Department of Mechanical and Mechatronics Engineering

Audio Processing in Time Domain

A Report Prepared For:

SYDE 252 - Linear Systems & Signals

Charbel Azzi

Prepared By:

Group 13

Jolie Girgis

Shreya Singh

Kiran Surendran

November 25, 2022

Abstract

This report discusses the design and implementation of low pass filters. Low pass filters allow low frequencies to pass through while attenuating higher frequencies based on a defined cutoff frequency threshold. The filters discussed in detail are the mean (moving average) filter, median filter, and Gaussian (weighted average) filter.

Using mathematical equations and examples, the report elaborates on the design methodology of these filters in greater detail. Furthermore, the methodology to resample and mix the stereo signals of an input file are also covered. This is applied to raw audio signals and is then used as input for all the low pass filters. After implementation of the low pass filters, the report also analyzes the best filter for audio smoothing and window size for the respective filter. The two metrics used to determine the best window size and filter are the sum of absolute differences method and qualitative inspection – both of which are outlined in the report.

Another major aspect of the report explores the design and implementation of three different detection methods which are applications of the low pass filters. The beats per minute detection method is an application of the moving average filter and is applied to the drum audio file to count all the beats. The syllable detection method is also an application of the mean filter which is used to count the syllables in the speech audio file. Lastly, the silent region detection method is implemented using the median filter for the birds audio file to find all the silent areas in the audio sample.

Table of Contents

List of Figures

List of Tables

1. Introduction

Audio processing has been a core part of the human experience, especially since the twilight of the 19th century. In 1877, Thomas Edison recovered Mary's Little Lamb from a strip of tinfoil wrapped around a spinning cylinder [1]. This resulted in the creation of the phonograph, more commonly known as the record player. This invention kicked off an evolutionary moment in music and audio, as now one could replay audio after it occurred in real life. As the maturation of this technology (into telegraphs, radio etc.) increased, the need for filtering and sampling grew.

Today, filtering in audio processing is very important, especially in applications such as speaker systems, instrument amplifiers, music synthesizers (or any modern music software creation for that matter) and audial artificial intelligence such as Google Home. In many of these applications, without adequate filtering and noise reduction it would be impossible for the systems to function accurately and robustly.

2. Background

This project is centered around designing filters for audio processing. Thus, it is important to first define the purpose of filters, the different types of filters, and how filters work. Filters are used in audio processing for filtering out noise in an audio sample. There are four main types of filters, a low pass filter, high pass filter, band pass filter, and band reject filter. Lowpass filters are designed to cut out undesired frequencies that are greater than a threshold frequency and allow all frequencies below it to pass. High-pass filters perform the opposite function; they cut out frequencies lower than the threshold frequency and allow all frequencies above it to pass. Band-pass filters only allow frequencies between a specified range through and blocks all other frequencies. While a band-reject filter rejects frequencies that are between a specified range and allows all other frequencies through [2].

The filters implemented in this project are all low pass filters and are the mean filter (also referred to as the moving average filter), gaussian average filter (also referred to as the weighted average filter), and the median filter. Each filter is explained in detail in section 3. Methodology.

3. Methodology

The math behind the three low pass filters will be explained in the following sections. To see the code implementation of each of the filters, refer to Appendix 7.1 Code Implementation.

3.1 Moving Average Filter Design

The moving average (mean) filter function takes two inputs, the input signal (x) and the width of the window size (L). The filter itself is given by the following difference equation, which essentially takes the mean value of the window around point n (specifically the values before point n) and this becomes the new value at n.

$$y[n] = \frac{1}{L}\sum_{k=0}^{L-1} x[n-k] \tag{1}$$

Equation (1) is expressed as a rational transfer function of the form (2).

$$H(z) = \left(\frac{1}{L}\right) * (b(1) + b(2)z^{-1} + \cdots b(L+1)z^{-L}), \qquad where\ b(n) = 1\ for\ all\ n \tag{2}$$

Using the built-in filter function in Matlab, the transfer function is applied on the input signal x and the output signal y is produced.

Another way to describe the moving average filter is in terms of an impulse response function. Equation (3) shows the impulse response function in the continuous domain. It uses the unit step function to create the window of interest, L and also normalizes it (i.e. takes the mean) by dividing by L. Figure 1 shows the impulse response for a window of 5.

$$h(t) = \left(\frac{1}{L}\right) \cdot \big(u(t) - u(t - L)\big) \tag{3}$$

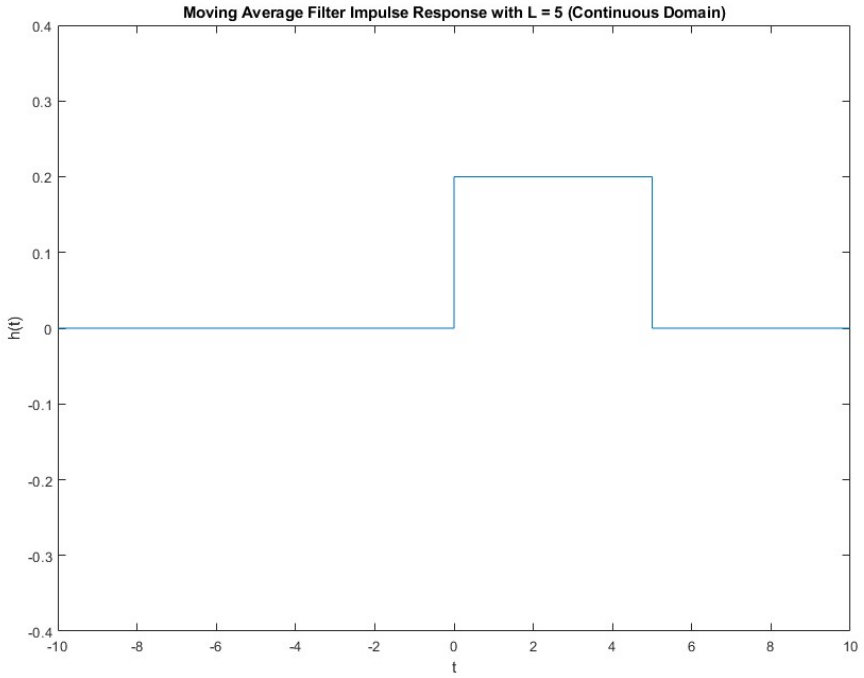

Figure 1: Moving average filter impulse response in continuous domain (L=5)

The impulse response function in the discrete domain is defined similarly as shown in (4). It uses the unit step function to create the window of interest, L, and also normalizes it (i.e. takes the mean) by dividing by L. Figure 2 shows the impulse response for a window of 5.

$$h[t] = \left(\frac{1}{L}\right) \cdot (u[t] - u[t - L]) \tag{4}$$

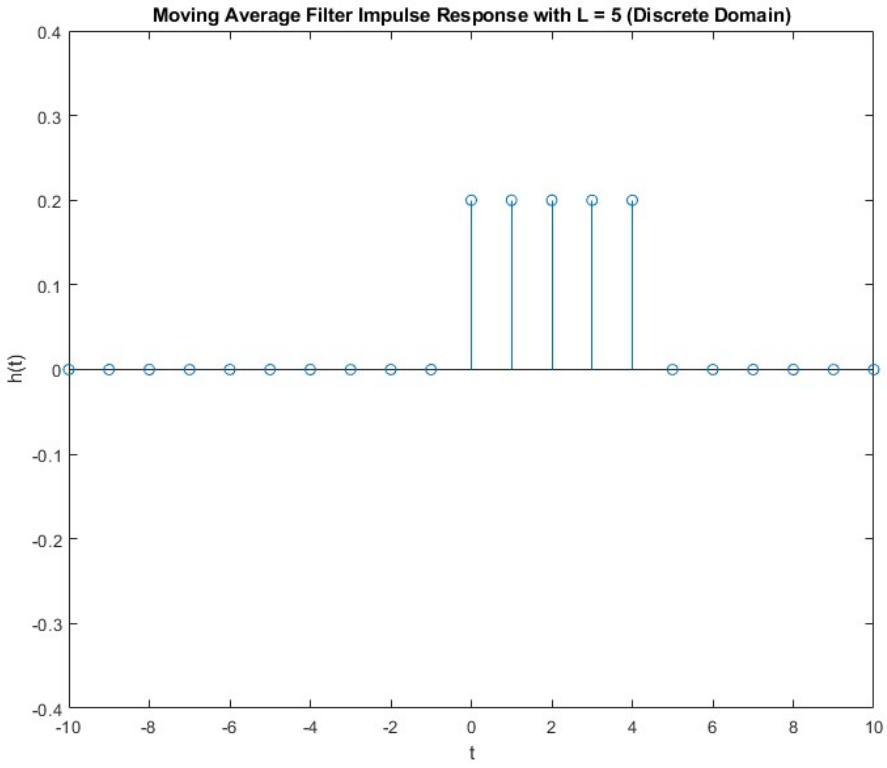

Figure 2: Moving average filter impulse response in discrete domain (L=5)

3.2 Weighted Average Filter Design

The weighted average filter applies weights to different sample values of the resampled input signal x[n] and gives more importance to the samples with higher weight. The applied filter weight to each sample is b_k , the weighted coefficient in (5).

$$y[n] = \sum_{k=0}^{L-1} b_k x[n-k] \tag{5}$$

A normal distribution of the window size is used to produce the gaussian coefficient, a_k which is then normalized using the sum of the distribution values to produce the final weighted coefficient, b_k. For example, for a window size of 3, the gaussian coefficient, a_k values are 1, 2, 1 based on the normal distribution. These values are then summed to provide the denominator. Each of the normal distribution

values are then divided by the sum to provide a b_k coefficient value for each sample. As a sanity check, all the weights should add up to 1. Lastly, all the values with applied weights are summed to produce the final output signal y[n].

To produce the gaussian coefficient, a_k, the gausswin function is used. It uses the window size as an argument due to the methodology explained above. Using the sum function, the coefficient values are summed and then the variable b_k stores the respective weight for each sample. Finally, the filter function is applied, which uses the weighted coefficient and input x[n] sample values, summed over the window size to produce the final filtered output signal y[n].

3.3 Median Filter Design

For each input value, the median filter replaces the input value with the median of the input value and values prior to the input value. The number of values that are considered when taking the median for an input value is dependent on the window size. The equation for the median filter can be seen in (6):

$$y[n] = median(x[n], x[n-1], x[n-2], \dots, x[n-k]), for\ k = 0,1,2, \dots, L \quad (6)$$

3.4 Syllables Detection

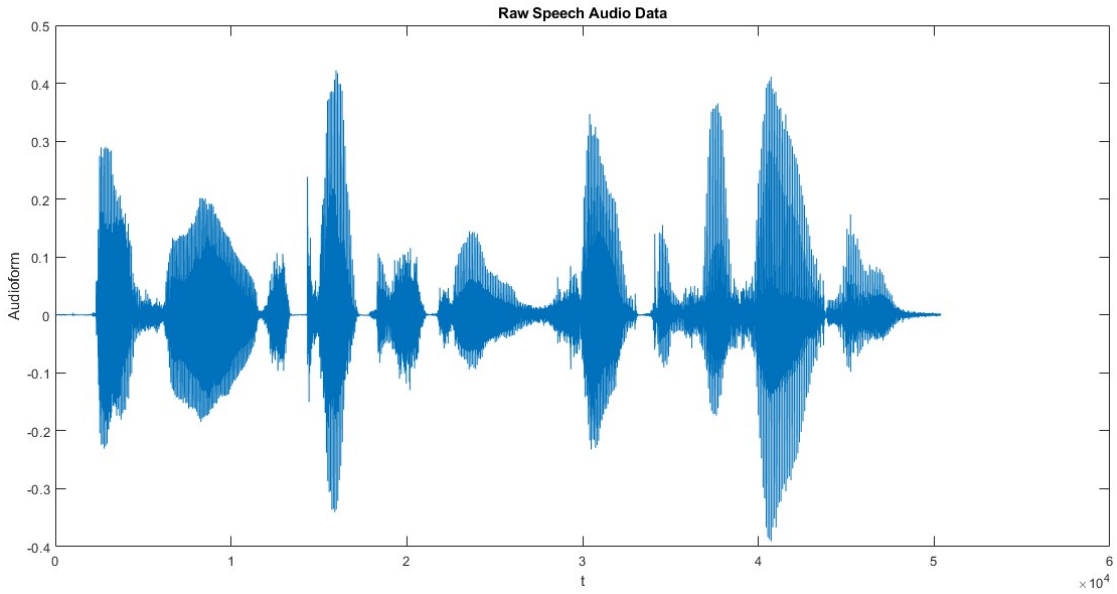

Figure 3: Raw speech audio data

The first step in the syllables detection algorithm was to reduce the noise in the raw audio file shown in Figure 3. The mean (moving average) filter was chosen as it was the most successful filter applied on the speech audio, based on the sum of absolute differences (SAD) metric. A value of 20 was chosen as the window size, based on Figure 19. The resulting data is shown in Figure 4.

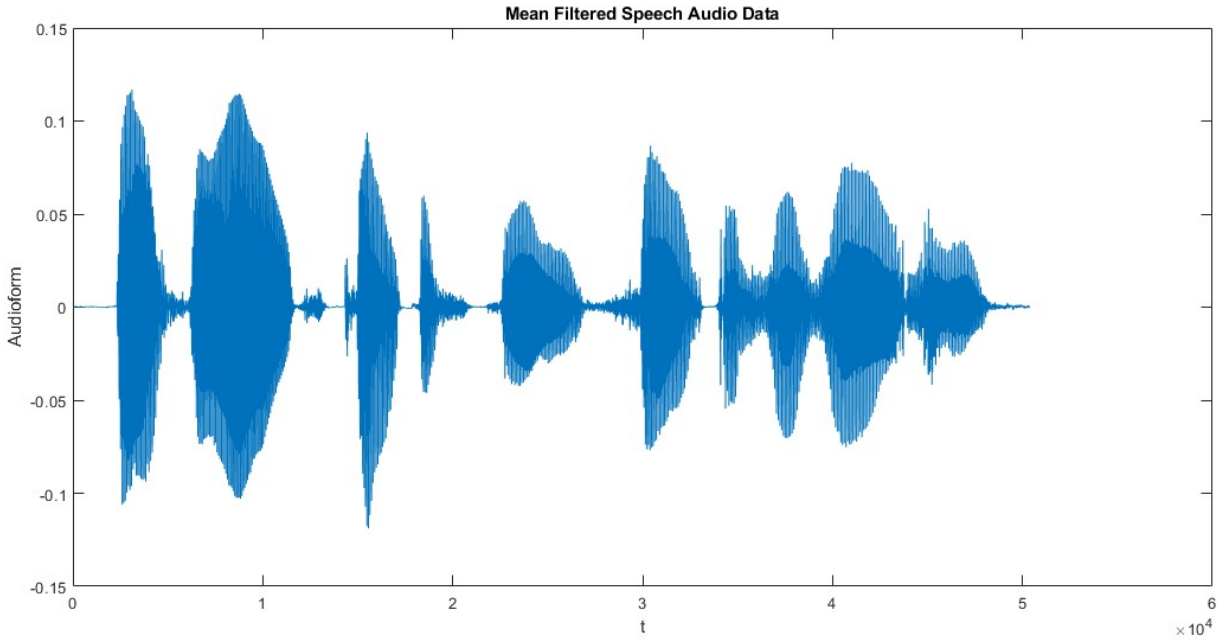

Figure 4: Mean-filtered speech audio data

The next step in the algorithm was to use a custom written peak detection function. This function works by finding the peak value in a sliding region across the input. A window size of 50 was chosen based on visual inspection of accurately picking out the peak regions of the audio data, with the resulting data shown in Figure 5. This step was required as the in-built MATLAB findpeaks function could not handle the mean filtered speech audio data with out large spikes. This step and the following step are used to smooth data further in preparation for the findpeaks function.

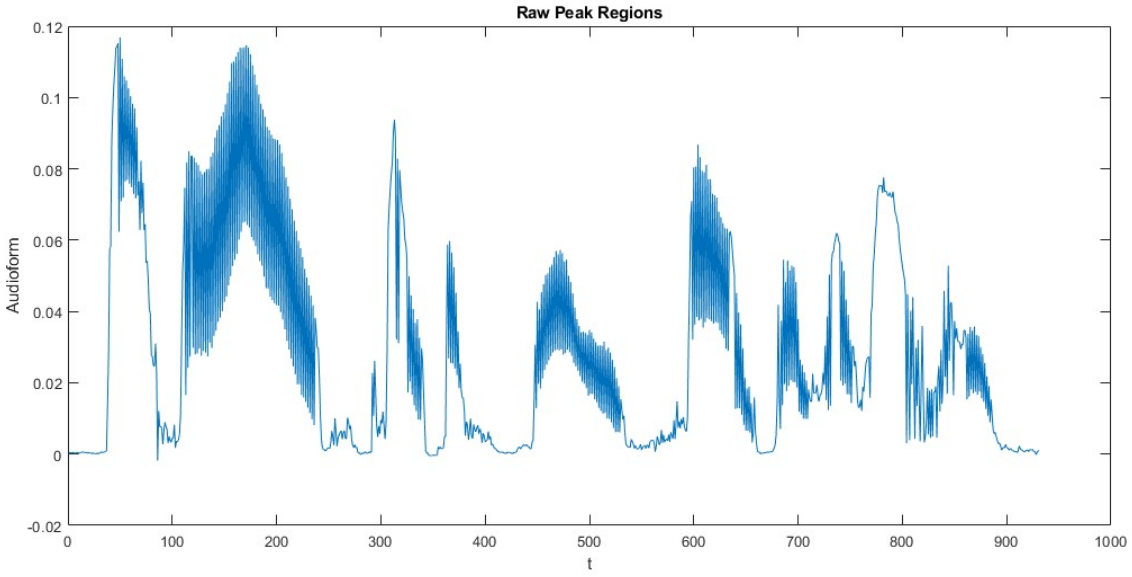

Figure 5: Speech data raw peak regions

The penultimate step is required due to the raw peak regions generated by the peak detection function being noisy. To smoothen this out, two cascading mean filters are applied (i.e. the signal is filtered and the result is filtered again). The first filter used a window size of 20 while the second filter in the cascade used a window size of 21. The window size was again chosen based on the optimal value shown on Figure 19, however a slight tweak (increase of the 2nd window size by 1) was required to remove a small amount of residual noise visible on the graph. The result is displayed in Figure 6.

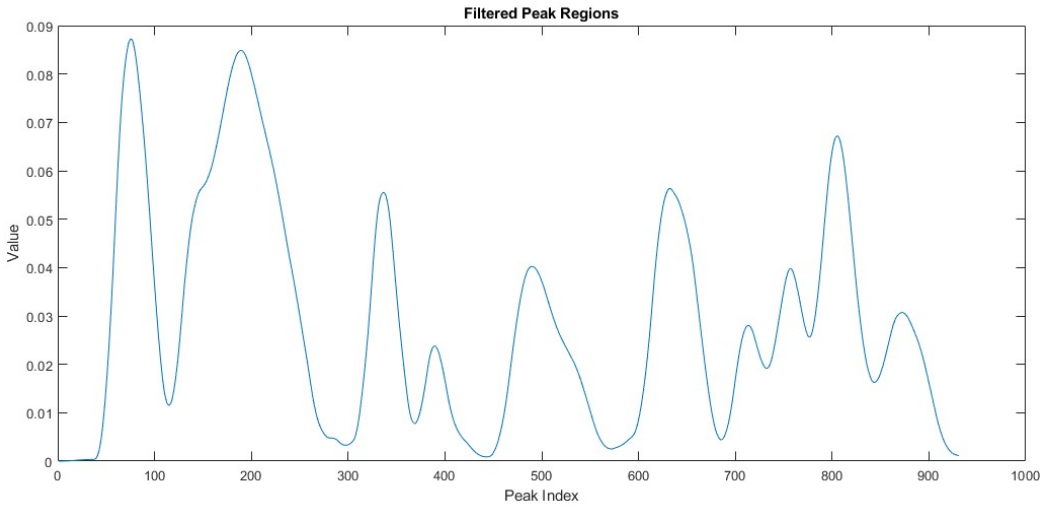

Figure 6: Speech filtered peak regions

The final step of the algorithm is to employ the use of the in-built findpeaks Matlab function to derive the number of peaks detected. As briefly mentioned earlier, the sliding peak detection function and cascading filters were used so the findpeaks function can easily detect the peaks without noisy peaks. The length of the findpeaks output vector is measured and represents the number of syllables detected. As shown in Figure 7, 10 peaks were detected thus 10 syllables were found for this sample.

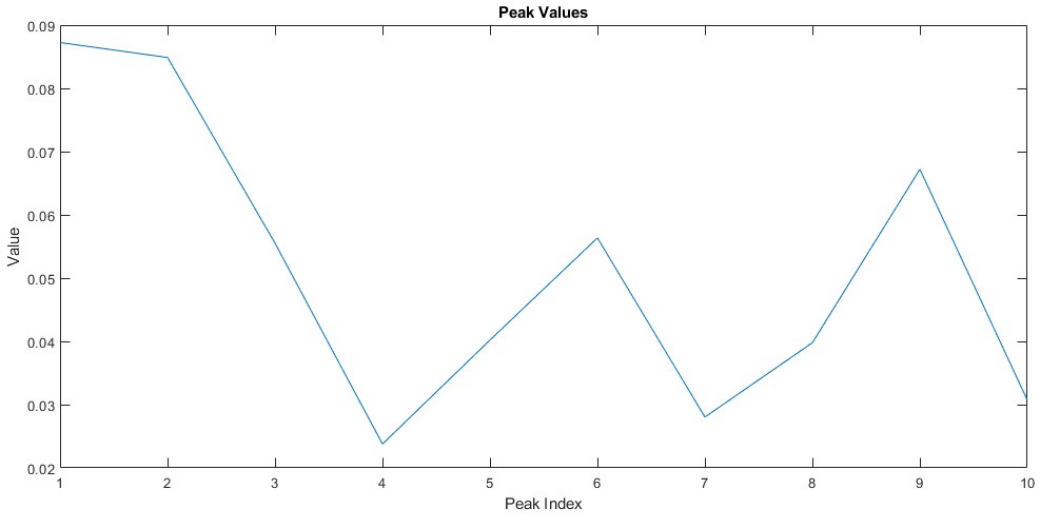

Figure 7: Speech findpeaks output

3.5 Beats Per Minute Detection

For detecting beats per minute of the drum, a function was developed which took two parameters in: the original raw audio file and a window size. The Init function is applied to the raw drum sound file which adjusts the sound to mono and sampling rate to 16KHz. Based on analysis in section 4.3 Comparison of Filters, the mean filter was chosen to be the best filter that smoothed out the drums audio file; whereas the median filter cut out parts of the audio file and the gaussian filter over-smoothed the drum beats. The analysis was done using the SAD method and qualitatively through listening as well – both of which are described in detail in section 4.3 Comparison of Filters. Once the mean filter is applied, the overall amplitude of all the beats is reduced significantly. But this depends on the window size chosen which is discussed next. Figure 8 below shows the audio file with the mean filter applied.

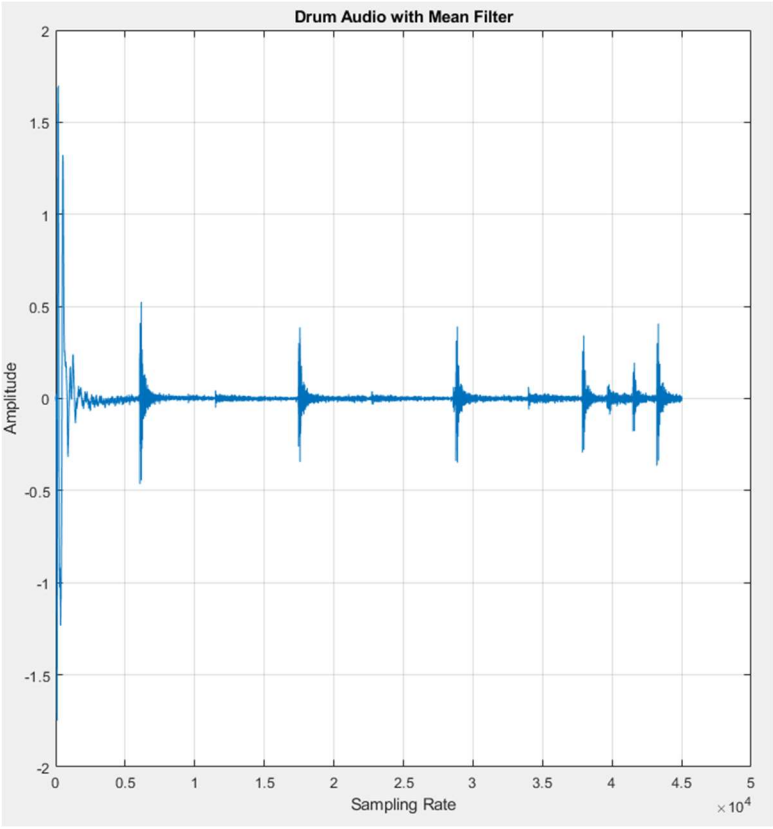

Figure 8: Mean filter applied to drum audio file with an arbitrary window size of 5

Furthermore, to choose the correct window size, the section 4.3 Comparison of Filters section was consulted. Based on that evaluation, 20 was the window size picked. Figure 9 below shows the mean filter applied to the sampled drum file with a window size of 20.

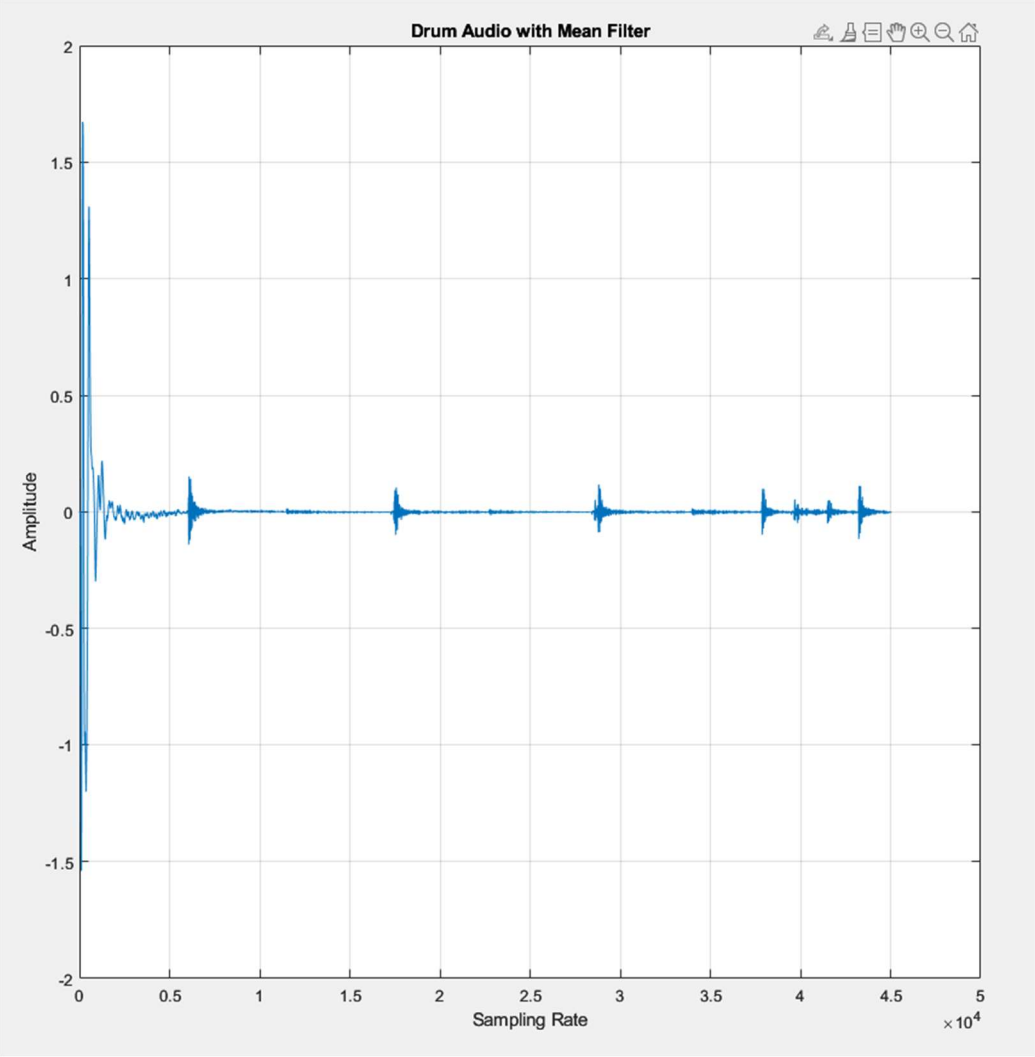

Figure 9: Mean filter applied to sampled drum file with a window size of 20

As can be seen, the amplitude of all the peaks is very small, hence it makes detecting the peaks slightly complicated. Decreasing the window size to 10 makes the peaks more prominent and allows for easier detection. This can be seen in Figure 10 below.

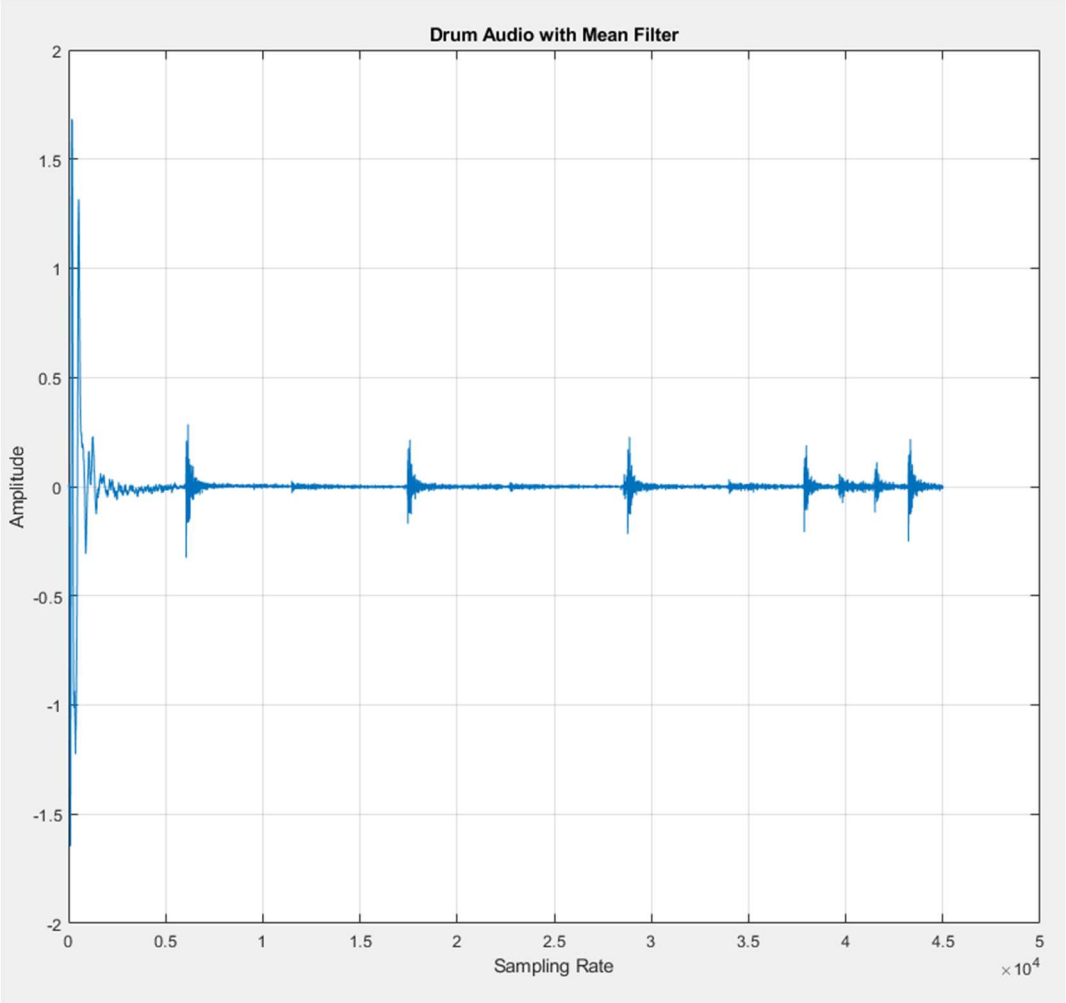

Figure 10: Mean filter applied to the drum audio file with a window size of 10

Once the filter and window size were selected, the function was implemented to use a threshold to cut off any peaks below a certain value thus counting only the beats of the drum. This value was visually inspected to be 0.16. Then the peaks above this value were counted and converted to only output beats per minute.

3.6 Silent Regions Detection

To detect the silent regions in the birds audio, the median filter is first used to reduce the noise in the audio sample. The median filter was determined to produce the best results for detecting silent regions as when each filter was applied to the birds audio, then the resultant plotted, the median filter

reduced the noise the best. Each filter in Figure 11 was plotted with the optimal window size determined in section 4.3 Comparison of Filters.

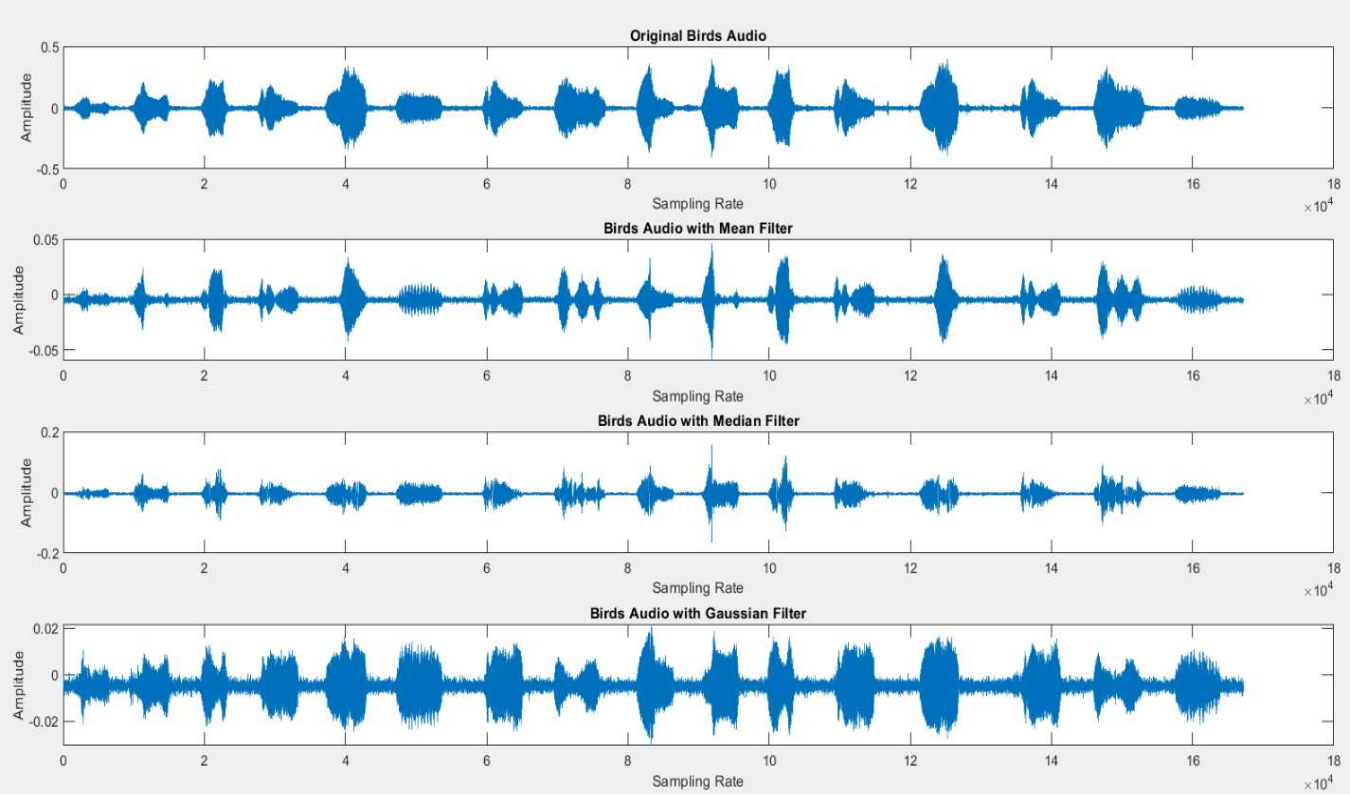

Figure 11: Birds audio with each filter applied

As can be seen, the mean and gaussian filters greatly reduce the amplitude of the overall birds audio. The median filter greatly reduces the amplitude of the audio in the silent regions, while keeping the main audio with a reasonable amplitude. The gaussian filter results in the silent regions being quite noisy and nearing the amplitude of the regions that are not silent. Thus, the median filter was chosen due to having the best reduction of the silent regions.

Once the median filter was chosen, the optimal window size for the filter was re-evaluated to determine the value that would work best for silent region detection. While the sum of absolute differences method, explained in section 4.3 Comparison of Filters, determined that a window size of 20 was the optimal window size for the median filter, a window size of 10 better preserved the amplitude

of the regions that were not silent (see Figure 12). Thus, the median filter with a window size of 10 was first applied to the birds audio to process it, before the silent regions could be detected.

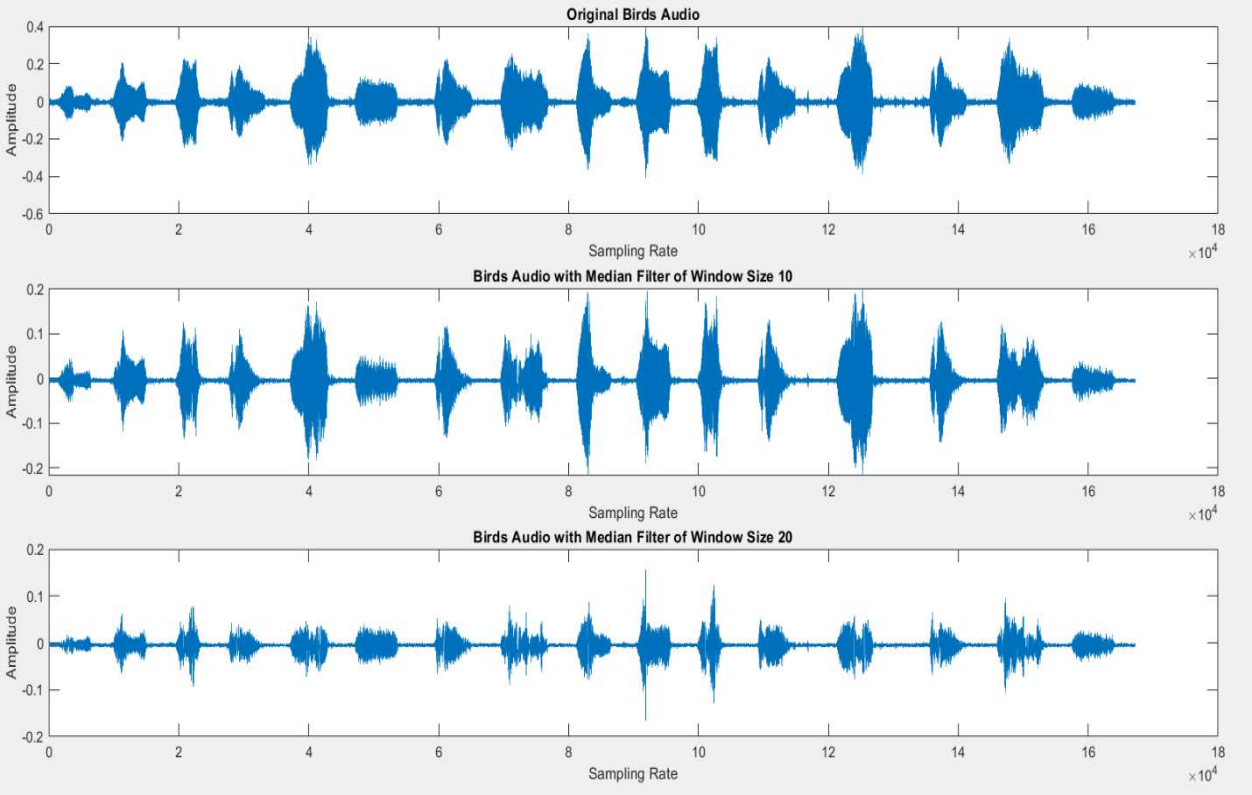

Figure 12: Comparison of window sizes for median filter on birds audio

After the birds audio has been processed, silent regions are detected by going through the processed audio sample and checking whether each value in the audio sample is less than the threshold value passed into the function or greater than the negative threshold value. If it is, then the output of the silence detection function for that index will be a 0, indicating that this index is part of a silent region. Otherwise, the output will be a 1, as it is not a part of a silent region.

4. Results and Discussion

4.1 Pre-processing Audio Samples Results

Before any filtering was accomplished, a preprocessing step was applied on all audio files. An initializing function was created to read in the filename of the relevant audio file then display the

sampling rate. It would then make the input audio into mono if it was stereo by adding the two channels together. After playing the sound for the user to hear, the new audio output was written to a new file with the name of the file pre-pended with the prefix “New”. Finally, the audio is resampled to 16KHz and the original (unsampled) audio signal is plotted as shown in Figure 13, Figure 14 and Figure 15.

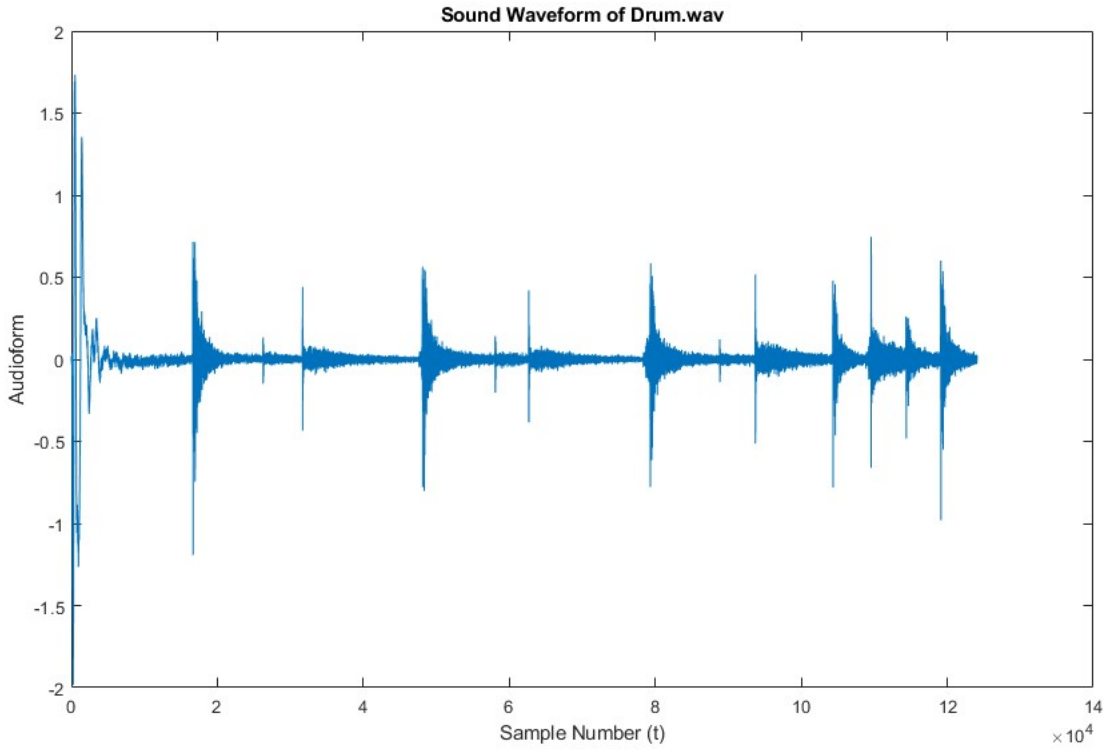

Figure 13: Sound waveform of Drum.wav

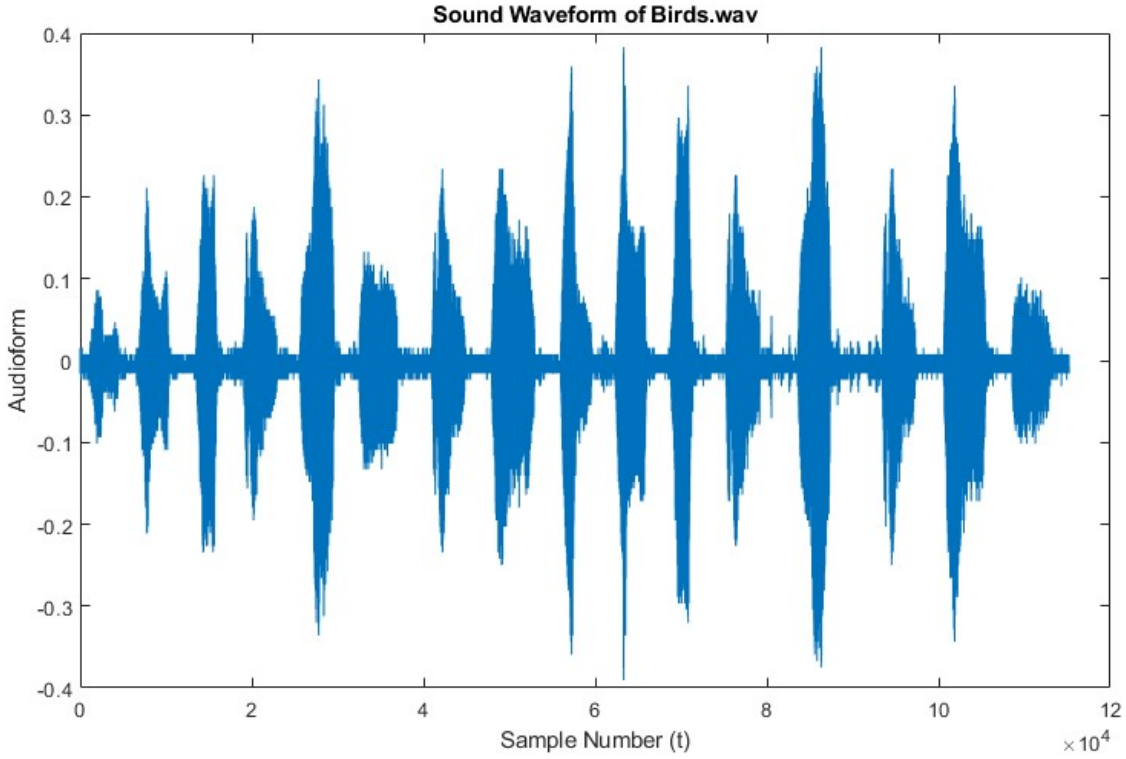

Figure 14: Sound waveform of Birds.wav

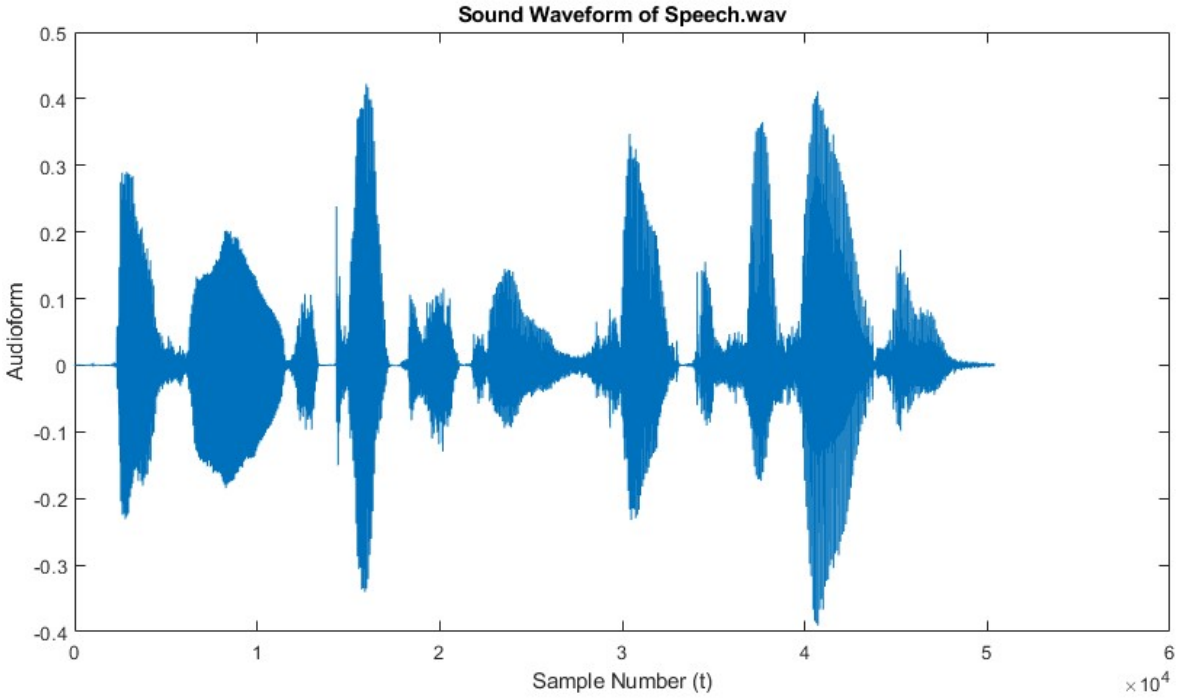

Figure 15: Sound waveform of Speech.wav

4.2 High Pass Filter

A high pass filter is used to pass any high frequency signals above a certain cutoff frequency. It implements this by attenuating the signals below the cutoff frequency. This can be demonstrated through the following equation where y[n] is the output signal and x[n] is the input signal.

$$y[n] = x[n] \; - \frac{1}{L}\sum_{k=0}^{L-1} x[n-k]$$

Thus, if the mean filter (low pass filter) is implemented on the input signal and those values are subtracted from the original input signal x[n], the output produced is a high pass signal that doesn't include any low pass frequencies of the input signal.

For example, with a window size of 3 (L is 3), the average over the current and previous two samples of the input signal will be calculated. This is the mean filter or moving average filter implementation which will then be subtracted from the original input signal. This leaves the high frequency aspect of the input signal which is essentially the output signal, as the low frequency signals of the input have been attenuated.

Once the high pass filter is implemented for the set window size, it slides over all the samples in the input signal to only allow the high frequencies of the entire input signal to pass.

4.3 Comparison of Filters

To determine the best window size for each filter, two metrics were used: a qualitative method and a quantitative method. The qualitative method involved applying different window sizes to a filter and comparing the resultant audio files. While the quantitative method involved plotting the SAD of the original 16 KHz resampled audio and the resultant audio signal from the applied filter. First, the

quantitative method was used and an example of the sum of absolute differences plot can be seen in Figure 16.

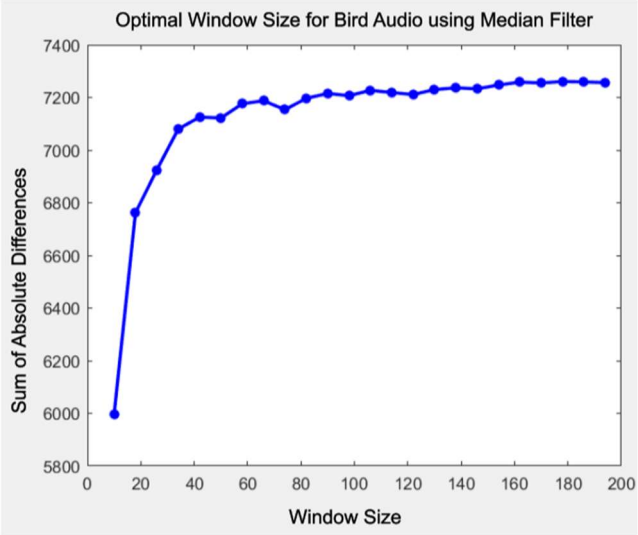

Figure 16: Optimal window size for birds audio using median filter through the SAD Method

Using the SAD plots, the smallest value at which the plot begins to stabilize or flatten out at was chosen to be the best window size for the audio sample [3]. As can be seen, the SAD plot for the median filter applied on the birds audio, begins to stabilize at a window size of approximately 80. To determine whether this value was acceptable, the qualitative method was used. It was determined that past a window size of 30, the audios were too distorted and quiet, and the low pass filters seemed to be filtering out more than just noise. Thus, the SAD plots were modified so that the maximum window size is 30. Then, the optimum window size was determined using the replotted graphs which can be seen in Figure 17, Figure 18, and Figure 19.

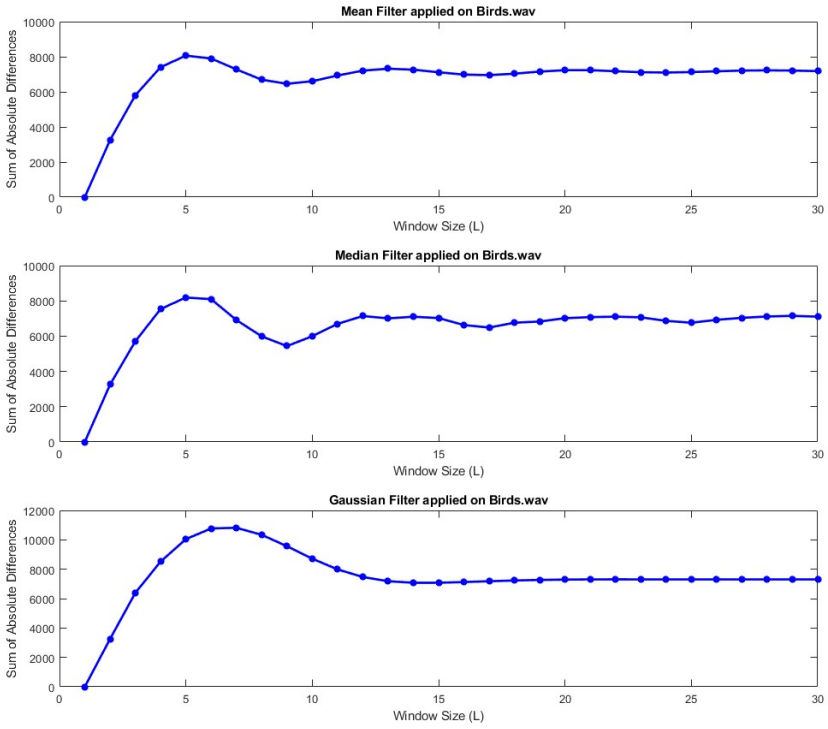

Figure 17: The sum of absolute differences for each filter for the birds audio

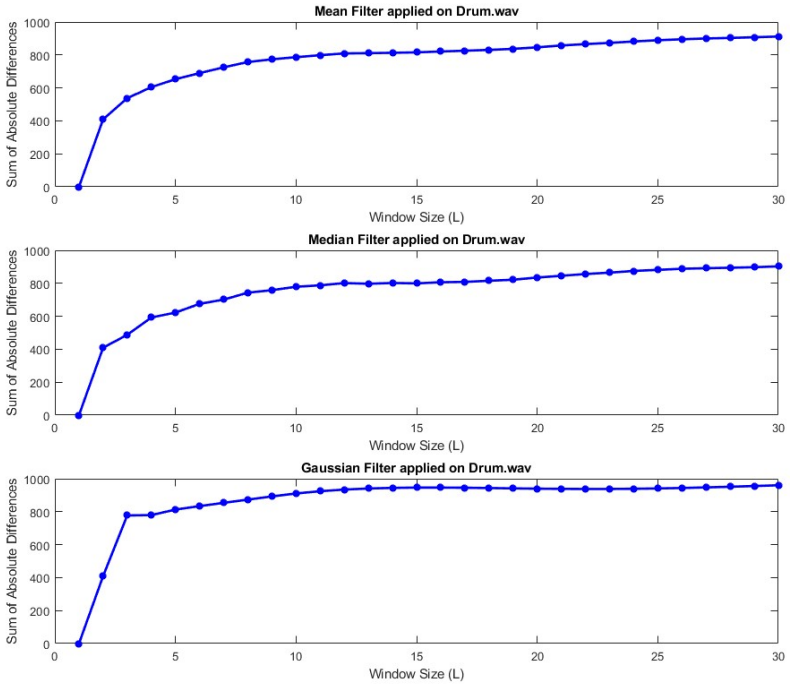

Figure 18: The sum of absolute differences for each filter for the drum audio

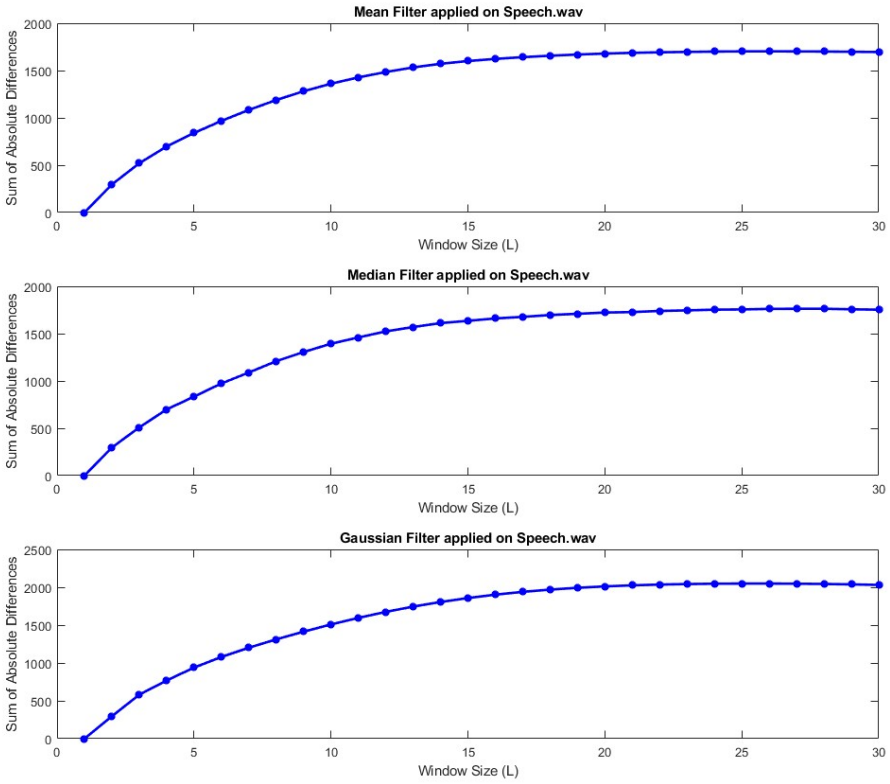

Figure 19: The sum of absolute differences for each filter for the speech audio

Table 1: Best window size for each filter applied on each audio sample

	Birds	Drum	Speech
Mean	15	25	22
Median	20	27	23
Gaussian (Weighted Average)	14	13	22

By inspecting Figure 17, Figure 18, and Figure 19, the best window size for each filter applied on each audio sample can be seen in Table 1. Then to determine which filter works best for each audio sample, the values for the SAD for each window size from Table 1 were tabulated in Table 2. The lowest SAD value was considered to be the best filter for that audio sample.

Table 2: The Sum of Absolute Differences for Each Window Size

	Birds	Drum	Speech
Mean	7114.33	889.335	1692.83
Median	7019	891.511	1745.95
Gaussian (Weighted Average)	7077.47	940.689	2035.63

Along with the SAD values, the output of a filter applied to an audio sample was listened to. For the birds audio, the sound was clear when the mean filter was applied, although quieter than the original birds audio. As for the median filter, the output audio was high-pitched, scratchy, and quiet. Thus, it was immediately eliminated as one of the options for the birds audio, even though it had the lowest SAD value. When the gaussian filter was applied, the output audio was slightly muffled, but otherwise was good. Although the gaussian filter produces a lower SAD value compared to the mean filter, the mean filter qualitatively sounded clearer and better than the gaussian filter. Thus, the mean filter was chosen as the best filter for the birds audio.

For the drum audio, the output audio for the mean filter was once again quiet but otherwise clear and had the lowest SAD value. The median filter resulted in an output sound that was heavily static, although had the second lowest SAD value. The gaussian filter resulted in an output sound that was very muffled and had the worst SAD value. Thus, the gaussian filter was eliminated from the options and the mean filter was selected as the best filter for the drum audio, despite having the lowest SAD value and due to its output audio sounding the best.

For the speech audio, the output audio for the mean filter was clear and the filter resulted in the lowest SAD value. The median filter once again caused the output audio to sound heavily static and had instances where the audio was high pitched, but had the second lowest SAD. The gaussian filter had the

worst SAD value and once again was muffled. Since the mean filter had the lowest SAD value and qualitatively sounded the best, it was chosen as the best filter for the speech audio.

A summary for the best filter and window size for each audio sample can be seen below in Table 3.

Table 3: Best filter and window size for each audio sample

Audio Sample	Best Filter	Best Window Size
Birds	Mean	14
Drum	Mean	25
Speech	Mean	22

Since the mean filter worked the best for all three audio samples, it can be concluded that the mean filter is the best low pass filter out of the three.

4.4 Syllable Detection Results

After applying the algorithm to the audio clip, an output of 10 syllables was detected. This result is consistent with the human derived solution after listening to the clip. However, there are some limitations to this algorithm. One major issue is the window size chosen for the cascading filter as the second level window size was tweaked visually to ensure the tested audio sample looked smooth. Each audio file might not be filtered enough or might be filtered too much depending on its frequency components. Similarly, the choice of peak detector window was narrowed down using visual observation.

Due to only tuning the algorithm on one audio file, the window sizes are likely to be biased to this audio input signal and may have issues processing other audio signals. The best way to alleviate this would be to do further tuning and testing with other audio signals to get a more generic solution.

4.5 Beats Per Minute Detection Results

An output of 190 beats per minute was produced based on the detection method implemented as per Section 3.5 Beats Per Minute Detection. This method produced a result that was within the expected range as per the lab manual but heavily depended on the threshold value implemented.

The method of detecting the peaks using a threshold value which is determined based on visual inspection is not a scalable way to implement any beats per minute detection system. This forces the code to be very suited to only one input audio file instead of being applied to multiple files.

Initially, the beats counting method was implemented on the sampled file and the amplitude of the peaks were very high, hence the threshold value chosen was also a very large number. After implementing the mean filter on the sampled file, the amplitudes were significantly reduced, which reduced the threshold value greatly. Thus, this explains the importance of building a detection method that does not solely rely on threshold values to detect beats. So, using this method for the beats per minute detection was its biggest limitation.

4.6 Silent Region Detection Results

As explained in section 3.6 Silent Regions Detection, a threshold value is required to determine whether a value should be considered silent or not. To determine the best threshold value for the birds audio with the median filter that has a window size of 10 applied to it, the output of the median filter was plotted and observed. It was determined that values approximately between -0.015 to 0.015 could be considered silent. To ensure that 0.015 was indeed the best threshold value, various threshold values were inputted into the silence detection algorithm and plotted to determine which resulted in the cleanest output. This can be seen in Figure 2020.

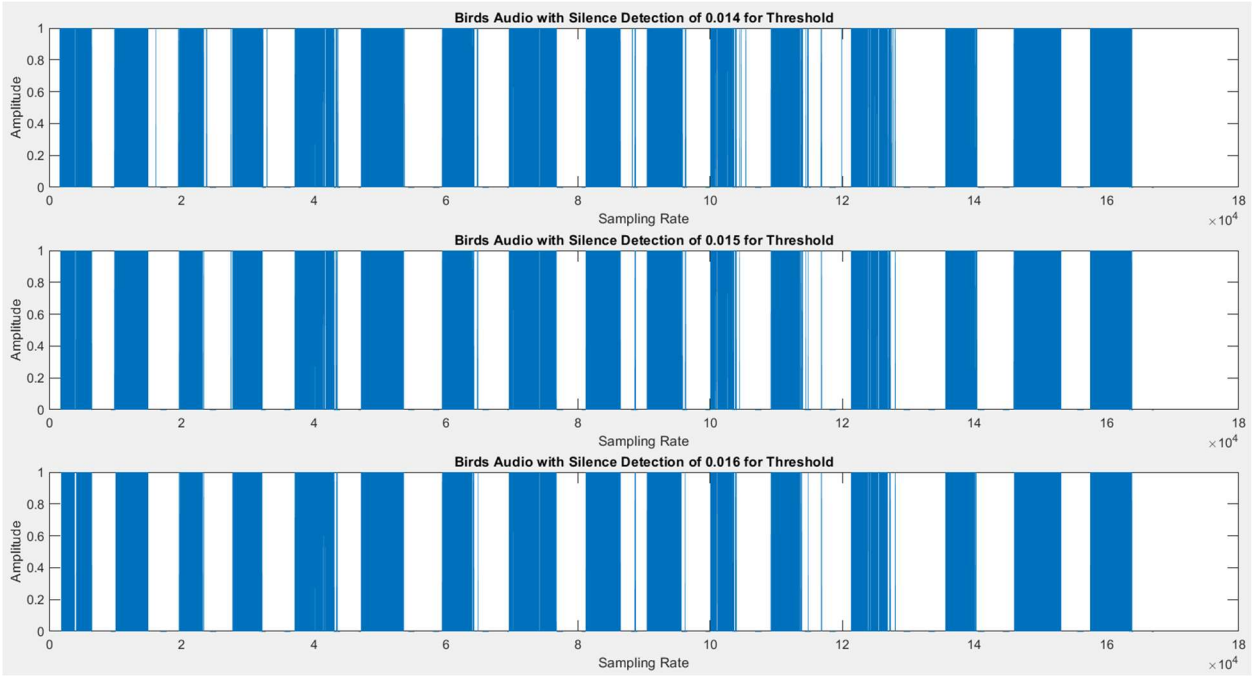

Figure 20: Comparison of Threshold Values for Silence Detection in Birds Audio

Thus, this confirmed that 0.015 was the best threshold value. Thus, the final result of the silence detection algorithm on the birds audio with a median filter that has a window size of 10 applied to it, can be seen in Figure 21.

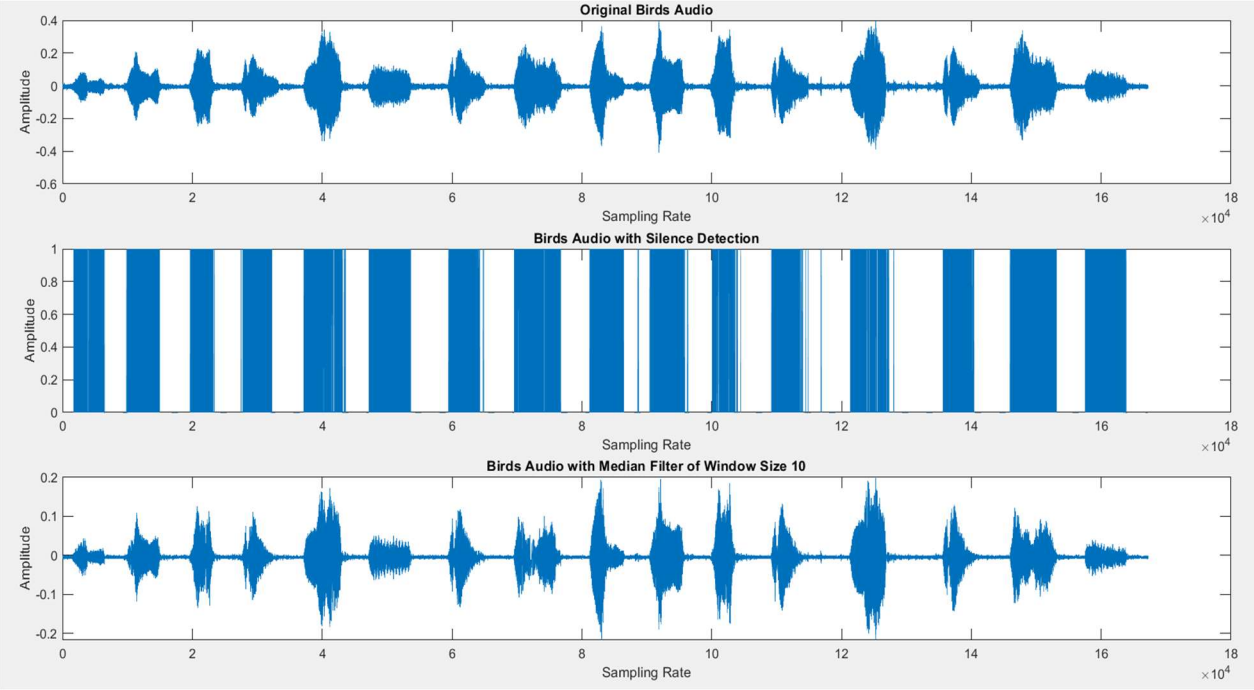

Figure 21: Silence Region Detection for Birds Audio

As can be seen, while the algorithm is able to detect the majority of silent regions correctly, it also mistakes some values as not being silent when they in fact are. It also detects some values as being silent when they should not be considered silent. Thus, it does not handle random peaks in silent regions well. As well as the median filter compressed some of the audio more than desired, which caused the algorithm to detect some values that should not be considered silent as silent.

This algorithm also can't be applied to any other audio file as it was designed specifically for the birds audio where the median filter of a window size of 10 was applied. A different audio file would result in a different output when the median filter is applied. Thus, the threshold value of 0.015 that was chosen for this project may not match where the silent regions are for another audio file. It is a manual process to come up with the threshold value as the output from the median filter must be examined to approximate the values of amplitude where the silent regions are. Therefore, while this algorithm works okay for the birds audio, it cannot be used for any other audio as another filter may work better, may require a different window size, and may use a different threshold value.

5. Conclusions and Recommendations

To summarize the report, the mean, median, and gaussian filters are three low pass filters that were created to filter out high-frequency noise in audio samples. All three filters depend on a window size and to determine this value, the sum of absolute differences method was used, as well as qualitatively evaluating the output audio samples by listening to them. The mean filter was determined to work best with a window size of 15 for the birds audio, 25 for drum audio, and 22 for the speech audio. For the median filter, it worked best with a window size of 20 for the birds audio, 27 for drum, and 22 for speech. For the gaussian filter, a window size of 14 worked best with the birds audio, 13 for drum, and 22 for speech. It was determined that the mean filter was the best at filtering out noise for all three audio samples, and thus was deemed the best filter out of the three.

The filters were then used for filtering the audio samples in order to perform three tasks, determining the number of syllables, beats per minute, and silent regions. The syllables algorithm used the mean filter multiple times in order to process the speech audio sample to a form that can be used to detect peaks in the sample. The beats per minute algorithm also used the mean filter to detect peaks in the drum audio sample so that the peaks can count as beats. The silent regions algorithm used the median filter to reduce the amplitude of the audio signal that was at a silent region to allow for easier detection of the silent regions. All three algorithms have the same limitations; they are all designed to only work with a specific audio file and thus the algorithms seem hard-coded. Algorithms that do not depend on specific threshold values would allow the algorithm to work on other audio samples.

6. References

[1] Audio Engineering Society, "AN AUDIO TIMELINE," Audio Engineering Society, 9 August 2013. [Online]. Available: https://www.aes.org/aeshc/docs/audio.history.timeline.html. [Accessed 24 November 2022].

[2] D. Wright, "3e - Frequency-Selective Circuits," [Online]. Available: https://learn.uwaterloo.ca/d2l/le/content/831396/viewContent/4646387/View. [Accessed 25 November 2022].

[3] MathWorks, "How to decide window size for a moving average filter?," 6 December 2016. [Online]. Available: https://www.mathworks.com/matlabcentral/answers/315739-how-to-decide-window-size-for-a-moving-average-filter. [Accessed 24 November 2022].

7. Appendices

7.1 Code Implementation

7.1.1 main.m

```
clc % Clear command window
clear % Clear workspace
%% Pre-processing audio samples
%resampled_birds = Init("Birds.wav");
%resampled_drum = Init("Drum.wav");
%resampled_speech = Init("Speech.wav");

%% Applying each filter on each audio sample
% mean_birds = meanFilter(resampled_birds,15);
% mean_drum = meanFilter(resampled_drum,25);
% mean_speech = meanFilter(resampled_speech,22);
%
% median_birds = medianFilter(resampled_birds,20);
% median_birds = medianFilter(resampled_drum,27);
% median_birds = medianFilter(resampled_speech,23);
%
% weighted_birds = weighted_avg(resampled_birds,14);
% weighted_birds = weighted_avg(resampled_drum,13);
% weighted_birds = weighted_avg(resampled_speech,22);

%% Beats Per Minute
% bpm('Drum.wav', 10);

%% Find optimal window sizes for each filter
% % Calculate SAD for each filter applied on Birds.wav
% meanSADBirds = sad(@meanFilter, resampled_birds);
% medianSADBirds = sad(@medianFilter, resampled_birds);
% gaussSADBirds = sad(@weighted_avg, resampled_birds);
% % Plot SAD's for Birds
% subplot(3,1,1)
% plot(meanSADBirds,'b*-', 'LineWidth', 2);
% title('Mean Filter applied on Birds.wav')
% xlabel("Window Size (L)")
% ylabel("Sum of Absolute Differences")
% subplot(3,1,2)
% plot(medianSADBirds,'b*-', 'LineWidth', 2);
% title('Median Filter applied on Birds.wav')
```

```
% xlabel("Window Size (L)")
% ylabel("Sum of Absolute Differences")
% subplot(3,1,3)
% plot(gaussSADBirds,'b*-', 'LineWidth', 2);
% title('Gaussian Filter applied on Birds.wav')
% xlabel("Window Size (L)")
% ylabel("Sum of Absolute Differences")

% % Calculate SAD for each filter applied on Drum.wav
% meanSADDrum = sad(@meanFilter, resampled_drum);
% medianSADDrum = sad(@medianFilter, resampled_drum);
% gaussSADDrum = sad(@weighted_avg, resampled_drum);
% % Plot SAD's for Drum
% subplot(3,1,1)
% plot(meanSADDrum,'b*-', 'LineWidth', 2);
% title('Mean Filter applied on Drum.wav')
% xlabel("Window Size (L)")
% ylabel("Sum of Absolute Differences")
% subplot(3,1,2)
% plot(medianSADDrum,'b*-', 'LineWidth', 2);
% title('Median Filter applied on Drum.wav')
% xlabel("Window Size (L)")
% ylabel("Sum of Absolute Differences")
% subplot(3,1,3)
% plot(gaussSADDrum,'b*-', 'LineWidth', 2);
% title('Gaussian Filter applied on Drum.wav')
% xlabel("Window Size (L)")
% ylabel("Sum of Absolute Differences")

% % Calculate SAD for each filter applied on Speech.wav
% meanSADSpeech = sad(@meanFilter, resampled_speech);
% medianSADSpeech = sad(@medianFilter, resampled_speech);
% gaussSADSpeech = sad(@weighted_avg, resampled_speech);
% % Plot SAD's for Speech
% subplot(3,1,1)
% plot(meanSADSpeech,'b*-', 'LineWidth', 2);
% title('Mean Filter applied on Speech.wav')
% xlabel("Window Size (L)")
% ylabel("Sum of Absolute Differences")
% subplot(3,1,2)
% plot(medianSADSpeech,'b*-', 'LineWidth', 2);
% title('Median Filter applied on Speech.wav')
% xlabel("Window Size (L)")
% ylabel("Sum of Absolute Differences")
```

```
% subplot(3,1,3)
% plot(gaussSADSpeech,'b*-', 'LineWidth', 2);
% title('Gaussian Filter applied on Speech.wav')
% xlabel("Window Size (L)")
% ylabel("Sum of Absolute Differences")

%% Silent Regions in Birds Audio
% Determining which filter to use for silent regions
detection
% Selecting a filter
% figure;
% subplot(4,1,1)
% plot(resampled_birds);
% xlabel("Sampling Rate");
% ylabel("Amplitude");
% title("Original Birds Audio");
%
% subplot(4,1,2)
% result1 = meanFilter(resampled_birds,15);
% plot(result1);
% xlabel("Sampling Rate");
% ylabel("Amplitude");
% title("Birds Audio with Mean Filter");
%
% subplot(4,1,3)
% result2 = medianFilter(resampled_birds,20);
% plot(result2);
% xlabel("Sampling Rate");
% ylabel("Amplitude");
% title("Birds Audio with Median Filter");
%
% subplot(4,1,4)
% result3 = weighted_avg(resampled_birds,14);
% plot(result3);
% xlabel("Sampling Rate");
% ylabel("Amplitude");
% title("Birds Audio with Gaussian Filter");

% Selecting a window size for the median filter
% figure;
% subplot(3,1,1)
% plot(resampled_birds);
% xlabel("Sampling Rate");
% ylabel("Amplitude");
```

```
% title("Original Birds Audio");

% subplot(3,1,2)
% result4 = medianFilter(resampled_birds,10);
% plot(result4);
% xlabel("Sampling Rate");
% ylabel("Amplitude");
% title("Birds Audio with Median Filter of Window Size
10");

% subplot(3,1,3)
% result5 = medianFilter(resampled_birds,20);
% plot(result5);
% xlabel("Sampling Rate");
% ylabel("Amplitude");
% title("Birds Audio with Median Filter of Window Size
20");

% Using the silence detection algoirthm
% figure;
% subplot(3,1,1);
% plot(resampled_birds);
% xlabel("Sampling Rate");
% ylabel("Amplitude");
% title("Original Birds Audio");
%
% subplot(3,1,2);
% result6 = silenceDetection(resampled_birds,10,0.015);
% plot(result6);
% xlabel("Sampling Rate");
% ylabel("Amplitude");
% title("Birds Audio with Silence Detection");
%
% subplot(3,1,3);
% plot(result4);
% xlabel("Sampling Rate");
% ylabel("Amplitude");
% title("Birds Audio with Median Filter of Window Size
10");

% Selecting the best threshold voltage for silence
detection algorithm
% figure;
% subplot(3,1,1);
```

```
% result7 = silenceDetection(resampled_birds,10,0.014);
% plot(result7);
% xlabel("Sampling Rate");
% ylabel("Amplitude");
% title("Birds Audio with Silence Detection of 0.014 for
Threshold");
%
% subplot(3,1,2);
% plot(result6);
% xlabel("Sampling Rate");
% ylabel("Amplitude");
% title("Birds Audio with Silence Detection of 0.015 for
Threshold");
%
% subplot(3,1,3);
% result6 = silenceDetection(resampled_birds,10,0.016);
% plot(result6);
% xlabel("Sampling Rate");
% ylabel("Amplitude");
% title("Birds Audio with Silence Detection of 0.016 for
Threshold");

%% Finding number of syllables in speech audio
% y = meanFilter(resampled_speech, 20);  % reduce noise
with mean filter
%
% rawPeakList = peakDetector(y, 50)';     % sliding
window peak detection
% peakList = meanFilter(rawPeakList,20);  % cascading
mean filter level1
% peakList = meanFilter(peakList,21);     % cascading
mean filter level2
% finalList = findpeaks(peakList);        % final find
peaks calculation
%
% % Plot raw audio data
% subplot(5,1,1)
% plot(resampled_speech)
% xlabel('t')
% ylabel('Audioform')
% title('Raw Speech Audio Data')
%
% % Plot mean filtered audio data
% subplot(5,1,2)
```

```
% plot(y)
% xlabel('t')
% ylabel('Audioform')
% title('Mean Filtered Speech Audio Data')
%
% % Plot raw peak regions
% subplot(5,1,3)
% plot(rawPeakList)
% xlabel('t')
% ylabel('Audioform')
% title('Raw Peak Regions')
%
% % Plot filtered peak regions
% subplot(5,1,4)
% plot(peakList)
% xlabel('Peak Index')
% ylabel('Value')
% title('Filtered Peak Regions')
%
% % Plot final peak values
% subplot(5,1,5)
% plot(finalList)
% xlabel('Peak Index')
% ylabel('Value')
% title('Peak Values')
%
% % Get number of peaks fron final list
% numPeaks = size(finalList,1);
% fprintf("\nThe number of syllables detected in this
audio clip is %d\n", size(finalList,1));


%% Define SAD calculator function
% function sadOut = sad(func, rawData)
%     windowArr = [1:1:30];    % initialize windows to
test
%     for k = 1 : length(windowArr)
%         y = func(rawData, k);
%         sadOut(k) = sum(abs(y - rawData));
%     end
% end
```

7.1.2 Init.m

```
function [result] = Init(filename)
            % Read sound file and determine sampling rate
    [data, spl_rt] = audioread(filename);
    disp(spl_rt) % Sampling rate

    % Adjust input sound to mono if it is stereo
    [m,n] = size(data);
    if n==2
        new_data=data(:,1)+data(:,2);
    else
        new_data = data;
    end

    % Play modified sound
    sound(new_data);

    % Write modified sound to a new file
    newfilename  = "New" + filename;
    audiowrite(newfilename,new_data,spl_rt);

    % Plot audio
    %plot(new_data);
    %xlabel("Sample Number (t)")
    %ylabel("Audioform")
    %title("Sound Waveform of "+filename)
    %grid on

    % Adjust sampling rate to 16KHz
    [p,q] = rat(16e3/spl_rt);
    result=resample(new_data,p,q);
end
```

7.1.3 meanFilter.m

```
function y = meanFilter(x,L)
    b = (1/L) * ones(1, L); % construct transfer function
                            numerator
    a = 1;                   % construct transfer function
                            denominator
    y = filter(b,a,x);       % apply transfer function on
                           input signal
```

```
end
```

7.1.4 medianFilter.m

```
function F = medianFilter(x, L)
    y = [];
    for i=1:length(x)
        section = []; % Stores the section the median should be taken on

        % For an index value that doesn't exist, consider the value
        % as 0, otherwise take the value at the valid index
        if i < L
            for k=1:(L-i)
                section(end+1,1) = 0;
            end
            for m=1:i
                section(end+1,1) = x(m);
            end
        end
        if i >= L
            for j=(i-L+1):i
                section(end+1,1) = x(j);
            end
        end

        % Get the median for the section
        y(i,1) = median(section);
    end
    F = y;
end
```

7.1.5 weighted_avg.m

```
% Takes in input parameters of sampled audio file and desired window size
function[final] = weighted_avg(x, L)
    % Produces the gaussian coefficient ak based on normal distribution
    % of the window size
    w=gausswin(L);
    % Sums up all the gaussian values
    s=sum(w,'all');
    % Divides each gaussian value by sum total to produce final weighted
    % oefficient bk
    bk=w/s;
    % Produces final output signal based on input sampled signal and bk
    % coefficient for each sample value of the input signal
    final = filter(bk,1,x);
End
```

7.1.6 peakDetector.m

```
% Finds the peaks of an input signal around a certain window defined by L
```

```
function peakList = peakDetector(x, L)
    peakList = [];
    for j = 1+(L/2):size(x)-(L/2)
        isPeak = true;
        for k=(-L/2):(L/2)
            if x(j) < x(j+k)
                isPeak=false;
                break
            end
        end
        if(isPeak)
            peakList(end+1) = x(j);
        end
    end
end
```

7.1.7 syllablesSpeech.m

```
function numPeaks = syllablesSpeech()
    speechAudioData = Init('Speech.wav'); % Get raw speech data

    y = meanFilter(speechAudioData, 20);  % Reduce noise with mean filter

    rawPeakList = peakDetector(y, 50)';     % Sliding window peak detection
    peakList = meanFilter(rawPeakList,20);  % Cascading mean filter level1
    peakList = meanFilter(peakList,21);     % Cascading mean filter level2
    finalList = findpeaks(peakList);        % Final find peaks calculation

    % Plot raw audio data
    subplot(5,1,1)
    plot(speechAudioData)
    xlabel('t')
    ylabel('Audioform')
    title('Raw Speech Audio Data')

    % Plot mean filtered audio data
    subplot(5,1,2)
    plot(y)
    xlabel('t')
    ylabel('Audioform')
    title('Mean Filtered Speech Audio Data')

    % Plot raw peak regions
    subplot(5,1,3)
    plot(rawPeakList)
    xlabel('t')
    ylabel('Audioform')
    title('Raw Peak Regions')

    % Plot filtered peak regions
    subplot(5,1,4)
    plot(peakList)
    xlabel('Peak Index')
    ylabel('Value')
    title('Filtered Peak Regions')
```

```
    % Plot final peak values
    subplot(5,1,5)
    plot(finalList)
    xlabel('Peak Index')
    ylabel('Value')
    title('Peak Values')

    % Get number of peaks fron final list
    numPeaks = size(finalList,1);
    fprintf("\nThe number of syllables detected in this audio clip is %d\n",
size(finalList,1));
end
```

7.1.8 bpm.m

```
function [count] = bpm(x, L)
sample=Init(x);
% Final variable that stores the signal smoothed by the mean filter
final=meanFilter(sample,L);
% Variable that stores the count incrementer
count=0;
%for loop that detects peaks in all samples of the signal that are greater
%than 0.16 and for each peak detected, the count variable is incremented
for i=0.5e4:length(sample)-0.5e4
    if sample(i)>0.16
        count=count+1;
    end
end
%beats converted to beats per minute format
disp((count*60)/2)
%plots the signals
x=1:length(final);
plot(final);
xlabel("Sampling Rate");
ylabel("Amplitude");
title("Drum Audio with Mean Filter");
grid on
end
```

7.1.9 silenceDetection.m

```
function F = silenceDetection(x, L, threshold)
    % Apply the median filter
    y = medianFilter(x,L);
    for i=1:length(y)
        % For a value that is between the threshold, consider it as part of
        % a silent region (=0), otherwise give it a value of 1
        if y(i) < threshold && y(i) > -threshold
            result(i,1) = 0;
        else
            result(i,1) = 1;
        end
    end
```

```
    F = result;
end
```