

# Project 1: Audio Processing in the Time Domain

Andrew Zhang, Isaac Zhang, Owen Li

Group 9

November 27, 2022

SYDE252 Linear Systems and Signals

Charbel Azzi

## 1 Abstract

This report discusses a signal filtering project, which entails multiple parts including reading and processing audio files, implementing and tuning filters, and creating algorithms for various tasks using the filters.

## 2 Table of Contents

### Table of Contents

Abstract.....	2
Table of Contents .....	3
List of Figures .....	4
List of Tables.....	5
1 Introduction .....	6
2 Background.....	6
3 Methodology.....	6
3.1 Computing Experimental Acceleration.....	6
4 Results and discussion.....	7
5 Conclusions and recommendations.....	7
6 Acknowledgements .....	7
7 References .....	7
Appendix A: Place the title of appendix here.....	8

### 3 List of Figures

Figure 1: Unfiltered Signal $x[n]$ and Mean Filtered Signal $y[n]$ .....	8
Figure 2: Unfiltered Signal $x[n]$ and Median Filtered Signal $y[n]$ .....	9
Figure 3: Unfiltered Signal $x[n]$ and Gaussian Filtered Signal $y[n]$ .....	10
Figure 4: Speech.wav Plot .....	11
Figure 5: Birds.wav Plot .....	13
Figure 6: Drum.wav Plot .....	14
Figure 7: Mean Filtered Speech Waveform for Syllable Detection .....	34
Figure 8: Comparison of the waveforms of newBirds.wav, the waveform with a moving average filter applied ( $L = 3$ ), and where silent regions were removed from the filtered waveform. ....	35

## 4 List of Tables

Table 1: Window Size Tuning for newBirds.wav .....	16
Table 2: Window Size Tuning for newDrums.wav .....	22
Table 3: Window Size Tuning for newSpeech.wav .....	28

## 5 Introduction

Sounds are present in nearly every aspect of our lives. We communicate by speaking to one another, we listen to music, and we watch videos on the internet. With so much audio input, the technology related to audio processing is integral to the way we perceive sound. This project is an investigation into audio processing in the time domain. Specifically, it involves the creation and application of digital filtering systems in Matlab. The project is split into three parts. The first part saw our group create a Matlab script that read and processed the audio files to prepare them for filtering. In the second part, our group implemented three digital filtering systems in Matlab. The three digital filters were a 'Mean Filter/ Moving Average Filter' that takes an average around an input signal, a 'Median Filter' that replaces a value of a sample by the median value of the filter, and a 'Gaussian Weighted Average Filter' that takes a normally weighted average of samples around an input signal. Each of the audio files were filtered separately with the three filters. By adjusting their window sizes, the optimal window size was found through an ear-test of the output audios. A thorough analysis was conducted to explain the reasoning for the optimal window size of each filter, as well as explaining which filter was the best for each audio file. The analysis includes qualitative explanations and tabulated quantitative results and plots. Algorithms were also created for various tasks using the filters, including counting number of syllables, determining beats per minute, and detecting silent audio regions. Methodologies for the filters and the algorithms are included in this report.

## 6 Background

A filter is any medium through which an audio signal passes [1]. Even the air that our sound waves travel through when we are speaking to one another can be thought of as a filter, albeit a very weak one. However, when thinking of filters that serve a purpose for sound signals, one typically thinks of something that can modify sound. A digital filter is a filter that transforms digital signals like the sounds you might

hear from a smartphone or laptop. Digital filters use algorithms to transform samples of values representing sound into a filtered output.

It is important to learn about digital filters because they are present in most signals interfaced with through computers. For instance, modern musicians use digital filters in nearly every piece of music they create [1]. Some reasons for the prevalence of digital filters are that they are programmable, and they can be easily designed, tested and implemented on a computer. In comparison, analog filters require custom circuits, and can change properties over time or operating circumstances such as temperature [2]. Digital filters are commonplace in a world full of signals and understanding them will aid in the understanding of all technology associated with them.

## 7 Methodology

This section of the report discusses the design of each filter, using expressions and examples to explain the inner workings of the filters. The algorithms for counting syllables, estimating beats per minute, and finding silent regions in audio are also discussed and introduced.

### 7.1 Moving Average Filter Design

A moving average filter (also referred to as mean filter) uses a “window” which slides across an input signal  $x[n]$  and outputs a signal  $y[n]$  with each sample in  $y[n]$  being an average of the  $x[n]$  samples within the window. The equation for the moving average filter is shown in Equation 1.

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

*Equation 1*

$L$  is the window size for the filter. For a moving average filter with a window size of 4, the expression evaluates to  $y[n] = \frac{1}{4}(x[n] + x[n - 1] + x[n - 2] + x[n - 3])$ . Figure 1 shows an example input signal  $x[n]$ , and the resultant filtered signal  $y[n]$  assuming a window size of 4.

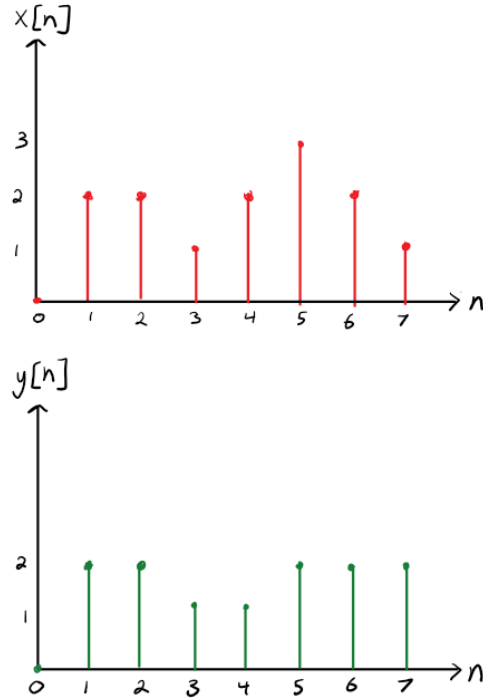


Figure 1: Unfiltered Signal  $x[n]$  and Mean Filtered Signal  $y[n]$

Note that  $y[n]$  and  $x[n]$  when  $n = 0, 1$ , and  $2$  are kept constant for this graphical explanation because there are not a sufficient number of samples to complete a window size of 4 according to Equation 1. The input value  $n = 5$  will be used for a deeper explanation of how the mean filter works. With a window size of 4,  $n = 5$  as well as the three samples before it ( $n = 4, 3$ , and  $2$ ) are examined. The sum across these four samples is 8 because  $x[5] + x[4] + x[3] + x[2] = 3 + 2 + 1 + 2 = 8$ . Since the window size is 4, the average across the four samples is 2 ( $8/2 = 4$ ). This procedure is equivalent to using Equation 1 when  $n = 5$  and  $L = 4$ . It is repeated across all samples in the input signal  $x[n]$ .

## 7.2 Moving Median Filter Design

A median filter replaces an input signal sample value with the median value of its window (which moves across the filter). The expression is shown in Equation 2.

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

Equation 2



$L$  is the window size. An example median filter with a window size of 3 results in an expression of  $y[n] = \text{median}(x[n], x[n - 1], x[n - 2])$ . Figure 2 shows an example input signal  $x[n]$  and the result of median filtering  $x[n]$  to create  $y[n]$ . A window size of 3 is used for the example.

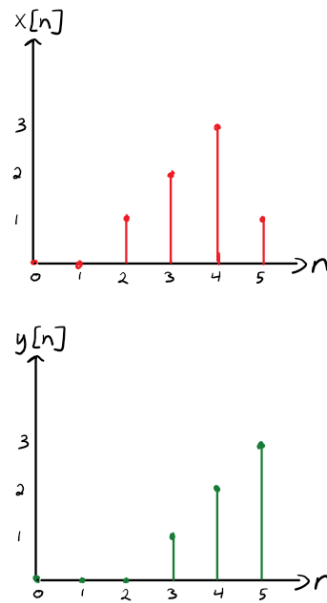


Figure 2: Unfiltered Signal  $x[n]$  and Median Filtered Signal  $y[n]$

Note that  $n = 0$  and  $1$  are kept constant between  $x[n]$  and  $y[n]$  because not enough previous values exist to satisfy a window size of 3 (which would require three samples). Considering the sample at  $n = 5$  (for example),  $y[5]$  becomes the median of  $x[5]$  and the two previous samples ( $x[3]$  and  $x[4]$ ). The median of those three  $x[n]$  samples is 3, and hence that is the value of  $y[5]$ . This approach is replicated for other possible values of  $n$ , to create a filtered  $y[n]$  signal.

### 7.3 Moving Gaussian Filter Design

The gaussian filter is a moving weighted average filter. It is similar to the moving average filter covered earlier in many ways, and the methodology is similar. The main difference is that while in the moving average filter, each sample within the window is multiplied by  $1/L$ , the gaussian filter contains different weight/multiplication values for each sample. The weights are determined using a gaussian

distribution, which means that samples towards the middle of the window have higher importance, while samples towards the edges of the window have lower significance. The expression for the gaussian filter is Equation 3.

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

Equation 3

L once again specifies the window size. The variable  $b_k$  represents the weight coefficient for each sample. For example, with a window size of 3, the weights become 0.0404, 0.9192, and 0.0404 respectively. It is important to note that the weights are averaged to always sum to 1. Figure 3 shows an example signal  $x[n]$  and its resultant filtered signal  $y[n]$  with a window size of 3.

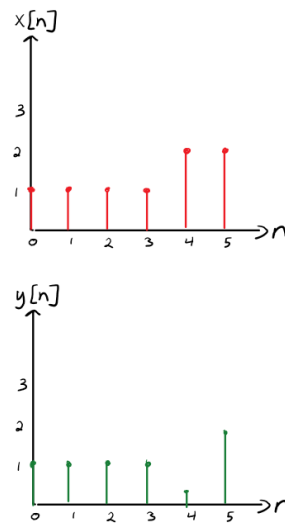


Figure 3: Unfiltered Signal  $x[n]$  and Gaussian Filtered Signal  $y[n]$

Examining the signals when  $n = 5$  gives insight into how the filter functions. With a window size of 3, the samples of interest include when  $n = 3, 4$ , and 5. Using the gaussian weights,  $y[5]$  becomes  $y[5] = 0.0404(1) + 0.9192(2) + 0.0404(1)$  which yields  $\sim 1.9$ , which is shown at  $y[5]$  in Figure 3. This process is repeated for individual samples.

## 7.4 Syllable Counting Algorithm

The syllable counting algorithm combines a mean filter along with Matlab's findpeaks function for detecting syllables. The original waveform for the speech audio contains a decent amount of noise which needs to be filtered out for detecting syllables. The original waveform is shown in Figure 4.



*Figure 4: Speech.wav Plot*

The syllables are generally visible in the plot, with most of the local maximums representing a syllable. However, there is still noise and random/abrupt spikes, which can be seen in the third sound section (which should not be identified as a syllable) and at other spots. The mean filter is leveraged to remove as much noise as possible while still retaining the original syllables. The mean filter is chosen because it has consistently good performance in filtering noise. More about the mean filter and its relative performance is mentioned in the results section of this report. A window size of 10 is used because experimentally it is found to filter the sound sufficiently for the purposes of syllable detection. After filtering, the findpeaks function is used with minimum peak distance and minimum peak height parameters to count the number of local maximums occurring, which corresponds to the number of syllables. The parameters are tuned by visually examining the plot and estimating a reasonable threshold

for peak height and peak separation/distance. The number of peaks detected is used as the number of syllables detected.

### 7.5 Beats Per Minute Algorithm

The beats per minute algorithm uses the same underlying technique as in the syllable detection algorithm. The drum audio file is first mean filtered with a window size of 3 (which experimentally is found to reduce noise while retaining significant peaks that correspond to drum beats). The `findpeaks` function is used with appropriate minimum peak height and peak distance parameters. The number of peaks found corresponds to the number of beats in the audio file. The total time in seconds of the audio file is calculated by taking the length of the signal and dividing it by the number of samples in the signal. The number of beats detected is multiplied by 60 divided by the length of the file in seconds. This gives a prediction for the number of beats in one minute (i.e. beats per minute).

### 7.6 Detecting Silent Regions Algorithm

Prior to processing and analyzing audio, any two-channel audio files were converted to single channel. This was done by adding the data in all columns of the  $m$  by  $n$  matrix outputted from Matlab's `audioread` function, then saving this as a new file. Within the  $m$  rows are data about the audio waveform, their indices being the samples taken in time domain.

When plotting wave forms, the relative loudness of the audio as it changes through time results in the characteristic shapes seen in Figure 4 and the window tuning results of the Results and Discussion section. Evident from these plots are dips near the horizontal axis, being the regions of audio with relatively quieter sound. Recall that information about the relative loudness of the waveform is outputted into the elements of a matrix by the `audioread` function. Then, to detect silent regions one possible method is with the use of a cutoff threshold, under which the waveform is considered to be silent.

For this project, the cutoff threshold was proportional to the magnitude of the highest peak.

## 8 Results and Discussion

This section discusses results from all parts of the project. This includes audio waveform plots from part 1, window size tuning from part 2 and a discussion surrounding the effectiveness of each filter, and algorithm results from part 3.

### 8.1 Part 1

Figure 4 shows the sound waveform for the Speech.wav file. Figure 5 shows the plot of Birds.wav (from step 5 of part 1).

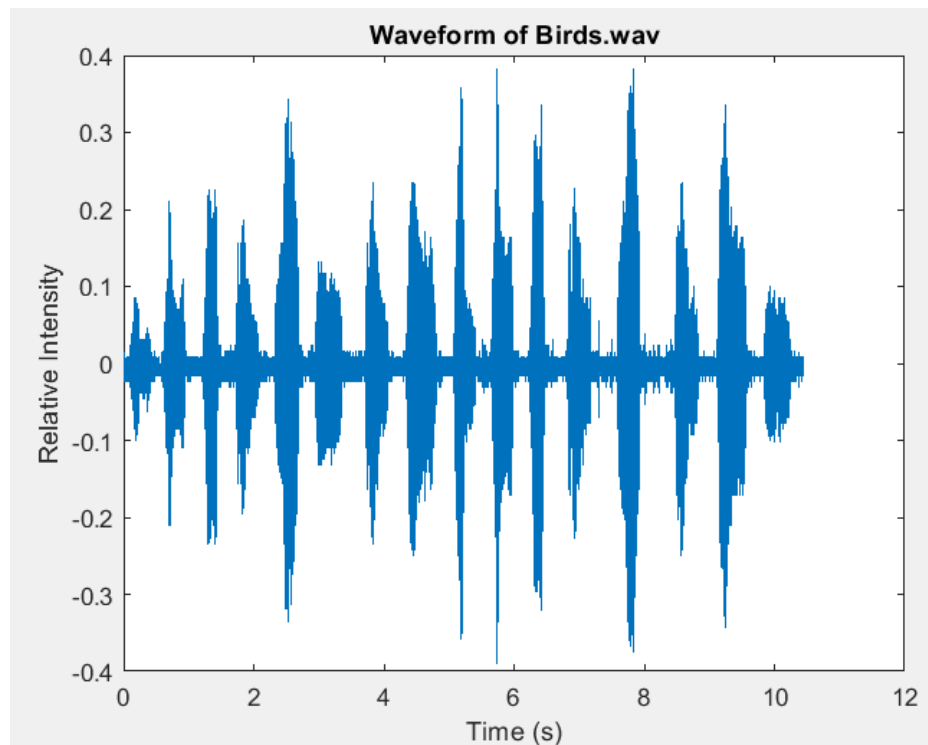


Figure 5: Birds.wav Plot

Figure 6 shows the plot of Drums.wav.

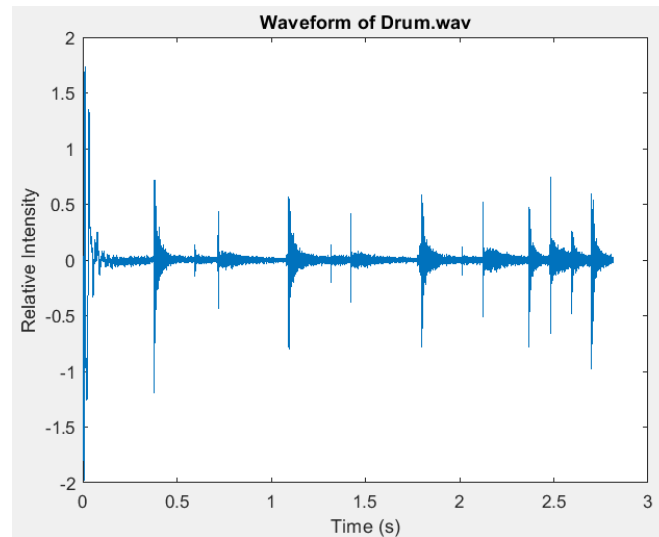


Figure 6: Drum.wav Plot

Please note that for part 1, two new audio files are created for each original audio file.

*newFilename.wav* is created during step 4, while *step6\_newFilename.wav* is created during step 6, after resampling is performed.

## 8.2 Part 2

Tuning the window size to obtain the optimal sound is very subjective and depends on the criteria used to assess the sound, therefore, for each sound the criterion will be explained. Sounds were assessed by ear. For each sound, each of the three filters were applied. For each filter, three window sizes were noted, the optimal window size, a window size that results in a slight distortion of the audio, and a window size that results in a high distortion of the audio. The use of distortion in this case is only related to a change in the audio sound, not necessarily a positive or negative change.

### Bird Audio File

The bird song had areas of very high intensity that were from the main bird, and a lot of background noise coming from other birds. Because the focus is on the main bird, the criterion for filtering this sound is how clearly the main bird can be heard.

Since one of the objectives of the filtering for this specific audio was to remove the background noise (noise from birds other than the main bird), in all 3 filters the optimal sound was a “middle” window size that was between a slightly distorted audio and a highly distorted audio. The effects of the filters were as follows:

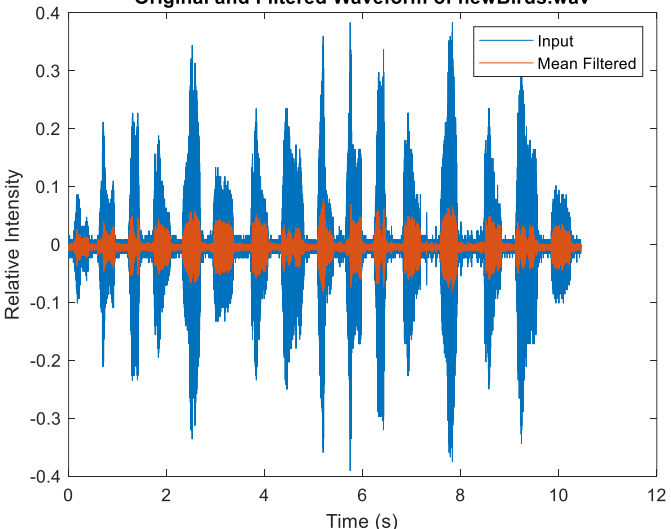
- Moving Average Filter:
  - Window Size 4 (slight distortion): The noise and volume/intensity of the audio were reduced slightly.
  - Window Size 5 (optimal distortion): The noise is reduced to allow for clear audibility of the main bird’s song. The volume/intensity of the audio is further decreased, but still loud enough for comfortable listening.
  - Window Size 24 (high distortion): The intensity/volume of the audio is decreased significantly to the degree that the main bird’s call is difficult to make out.
- Median Filter
  - Window Size 3 (slight distortion): The noise and volume/intensity of the audio were reduced slightly.
  - Window Size 4 (optimal distortion): The noise is reduced to allow for clear audibility of the main bird’s song. The volume/intensity of the audio is further decreased, but still loud enough for comfortable listening.
  - Window Size 5 (high distortion): The intensity/volume of the audio is decreased slightly, but a high-pitched static sound is introduced, disrupting the main bird’s song.
- Weighted Average Filter
  - Window Size 5 (slight distortion): The noise and volume/intensity of the audio were reduced slightly.
  - Window Size 6 (optimal distortion): The noise is reduced to allow for clear audibility of the main bird’s song. The volume/intensity of the audio is decreased more than the moving

average filter with optimal window size, but still loud enough for comfortable listening.

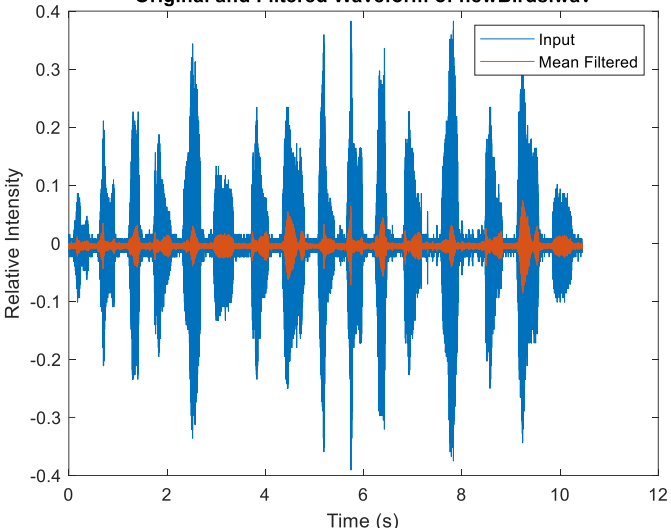
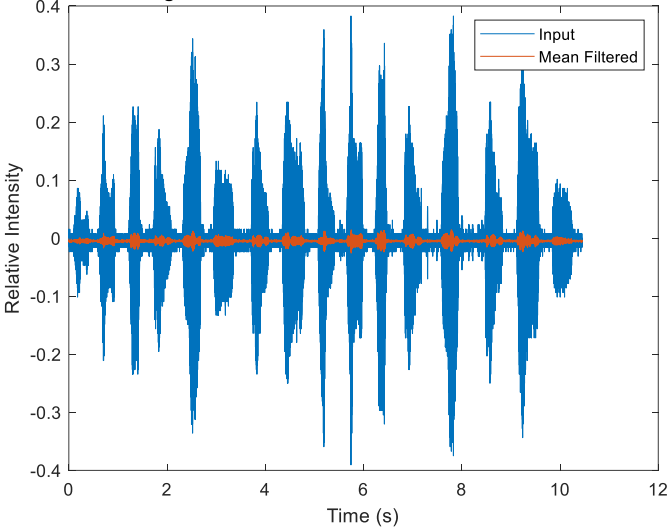
- Window Size 10 (high distortion): The intensity/volume of the audio is decreased significantly to the degree that the main bird's call is difficult to make out.

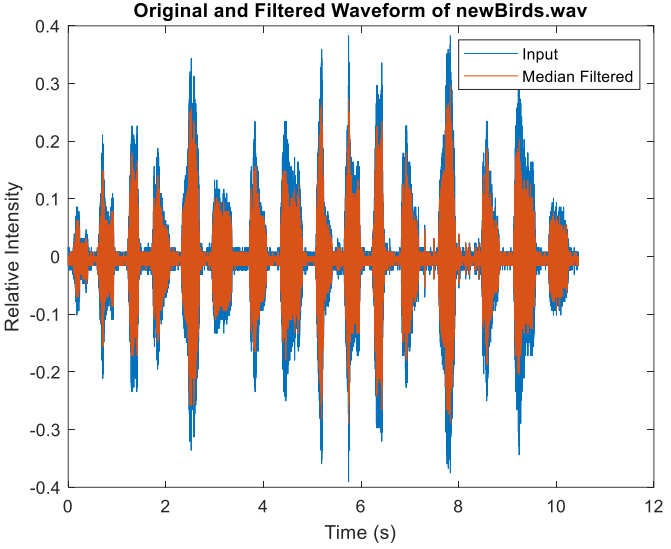
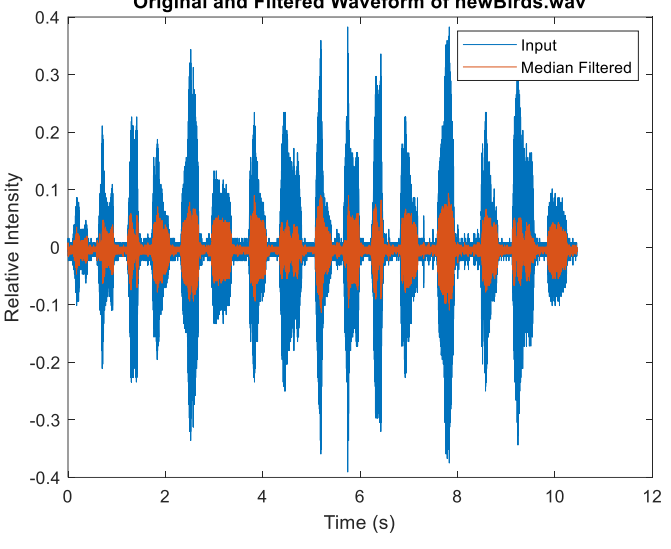
Of the three filters, the one with the best audio quality for hearing the main bird's song in the ear-test was the median filter at a window size of 4. This makes sense in relation to how the filters work. The background noise in this audio is consistent throughout the length of the waveform, therefore consistently taking the median will preserve the peaks during areas of high intensity and remove any noise in the sound, improving the clarity when the main bird is singing.

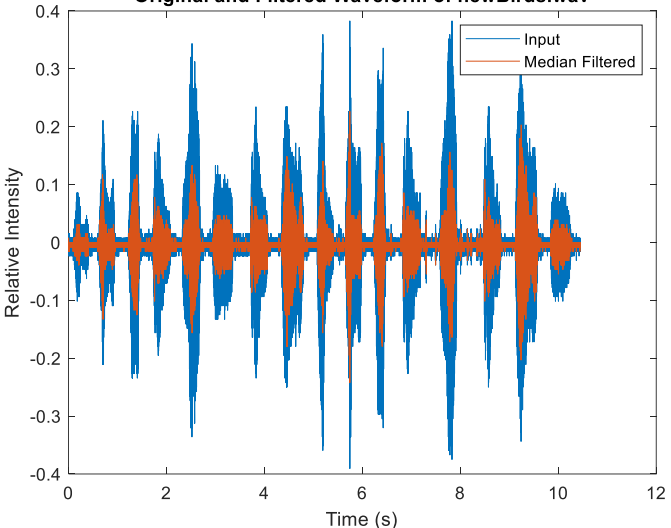
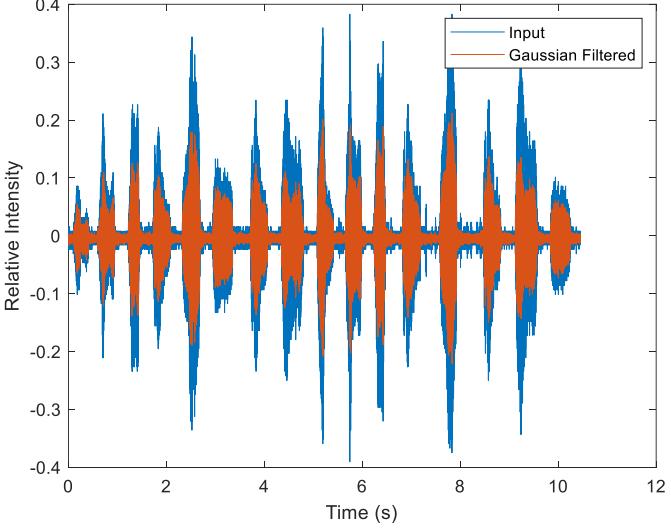
Table 1: Window Size Tuning for newBirds.wav

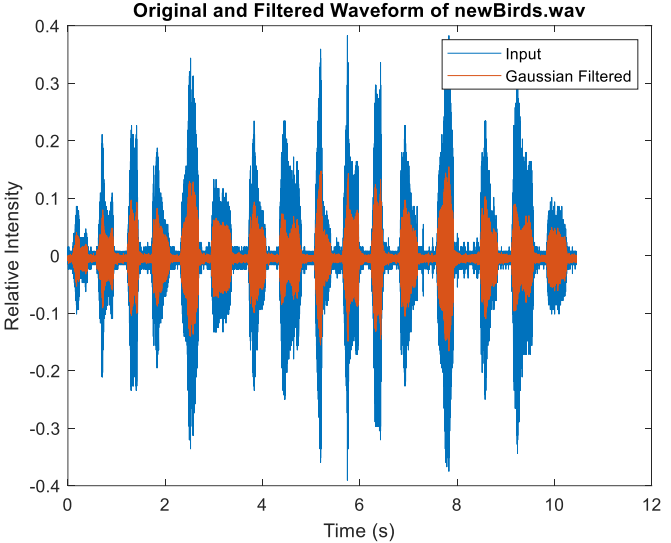
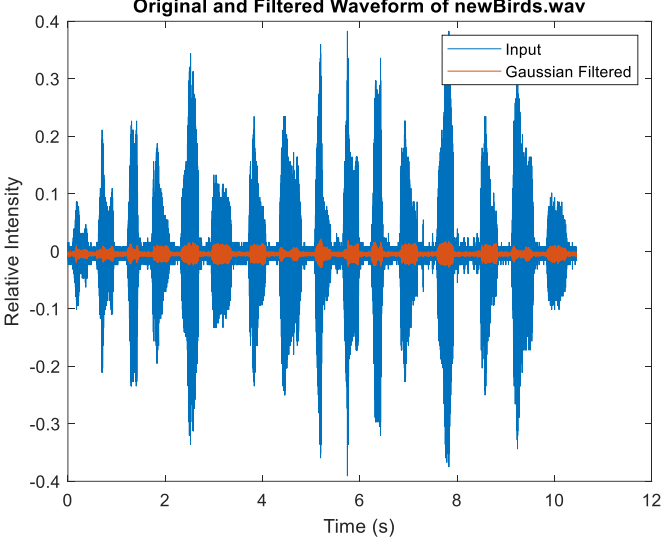
Filter	Window Size	Distortion Level	Plot
Moving Average Filter	4	Slightly	



	5	Optimal	<p>Original and Filtered Waveform of newBirds.wav</p> 
	24	High	<p>Original and Filtered Waveform of newBirds.wav</p> 

Median Filter	3	Slightly	
	4	Optimal	

	5	High	<p>Original and Filtered Waveform of newBirds.wav</p>  <p>This plot shows the original input waveform (blue line) and the median filtered waveform (orange line) for the file newBirds.wav. The x-axis represents Time in seconds, ranging from 0 to 12. The y-axis represents Relative Intensity, ranging from -0.4 to 0.4. The median filter effectively removes the sharp, high-frequency spikes present in the original signal, resulting in a smoother waveform that follows the general envelope of the original signal.</p>
Weighted Average Filter	5	Slightly	<p>Original and Filtered Waveform of newBirds.wav</p>  <p>This plot shows the original input waveform (blue line) and the Gaussian filtered waveform (orange line) for the file newBirds.wav. The axes and legend are identical to the first plot. The Gaussian filter smooths the waveform by averaging the values over a neighborhood, which significantly reduces the high-frequency noise and sharp spikes seen in the original signal, creating a more continuous and rounded waveform.</p>

	6	Optimal	
	10	High	

### Drums Audio File

The drum audio is musical. Therefore, the way this sound file was assessed was in the context of a user listening to music. The scenario used to judge this audio was considering if it was coming from a music streaming service such as Spotify. At what point would the output audio be optimal for the listener?

Since the drum audio is of musical nature, almost all the sounds come from the instruments and add to the sound of the drums. In music production editing of the sound is specific to the intended result of the music. Thus, when tuning for the window size, the ear-test was mainly to find out when a noticeable negative

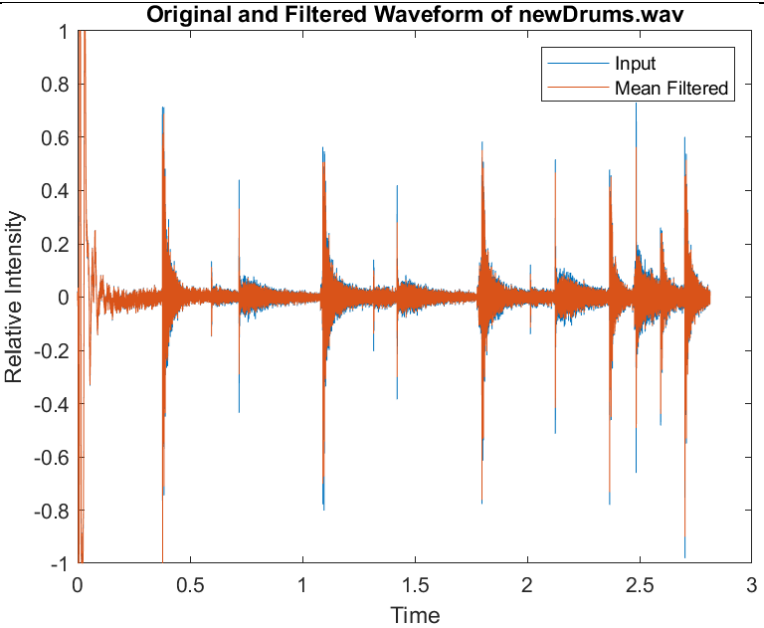
distortion appears (slight distortion), and when a severe negative distortion appears (high distortion). The criterion used for a high distortion is when the music is no longer satisfying to listen to.

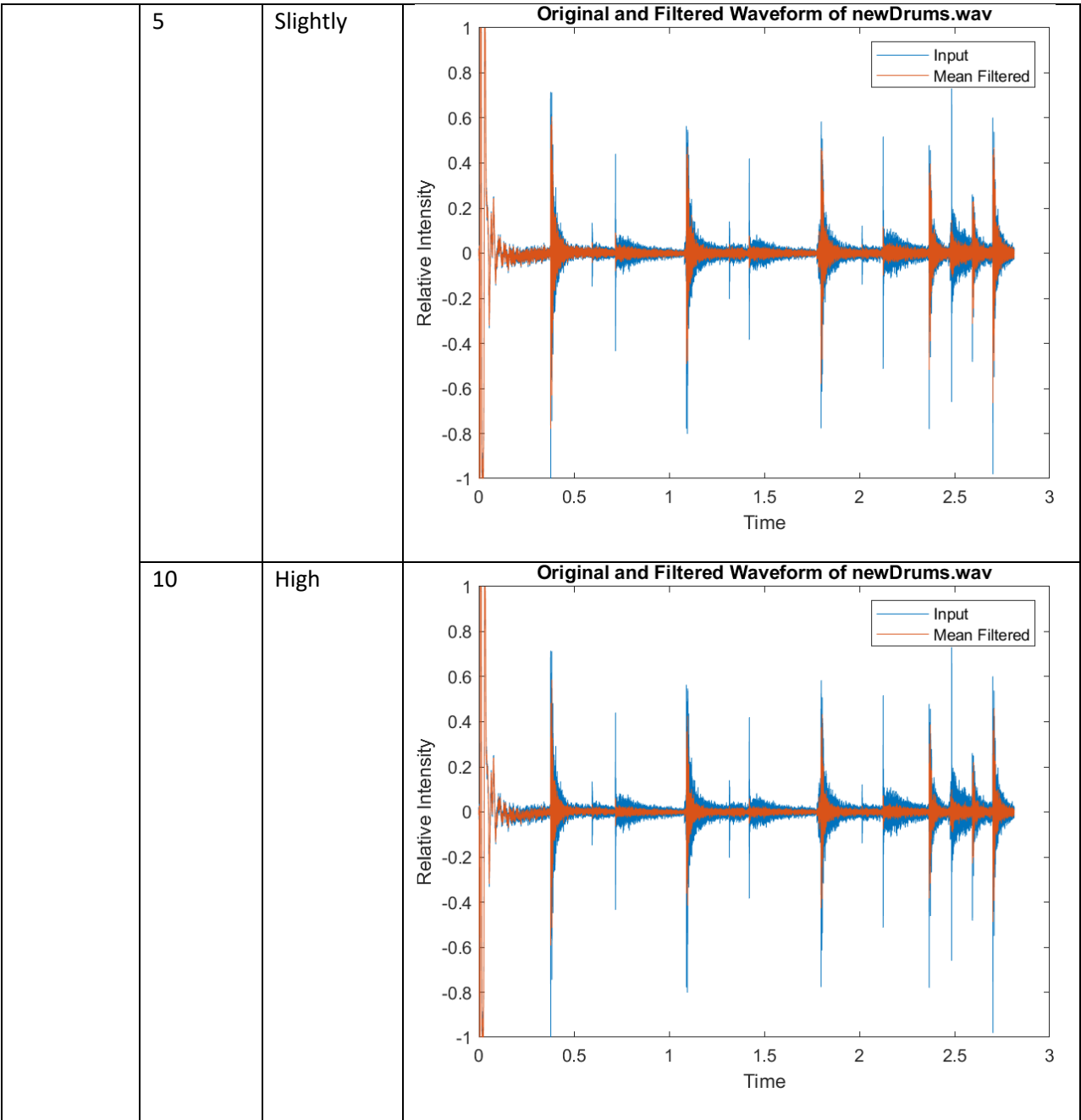
- Moving Average Filter:
  - Window Size 2 (optimal distortion): The noise is reduced to allow for clear audibility of the drums.
  - Window Size 5 (slight distortion): A slight echo effect is introduced to the drums.
  - Window Size 10 (high distortion): The intensity/volume of the audio is decreased significantly, and the echo effect becomes a muted effect.
- Median Filter
  - Window Size 4 (optimal distortion): The noise is reduced to allow for clear audibility of the drums.
  - Window Size 5 (slight distortion): A slight echo effect is introduced to the drums.
  - Window Size 12 (high distortion): The intensity/volume of the audio is decreased significantly, and the echo effect becomes a muted effect.
- Weighted Average Filter
  - Window Size 3 (optimal distortion): The noise is reduced to allow for clear audibility of the drums. The trailing sounds of previous bass drum and cymbals are retained.
  - Window Size 5 (slight distortion): A slight echo effect is introduced to the drums.
  - Window Size 10 (high distortion): The intensity/volume of the audio is decreased significantly, and the echo effect becomes a muted effect.

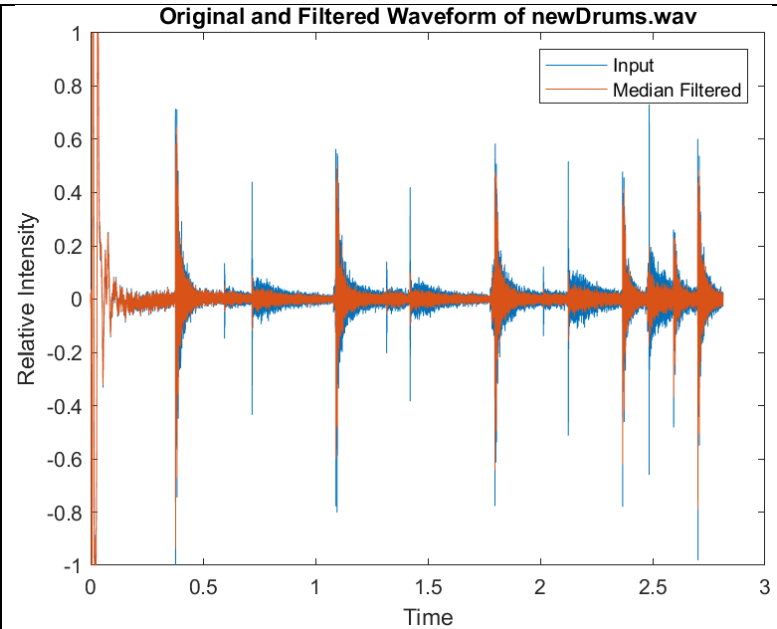
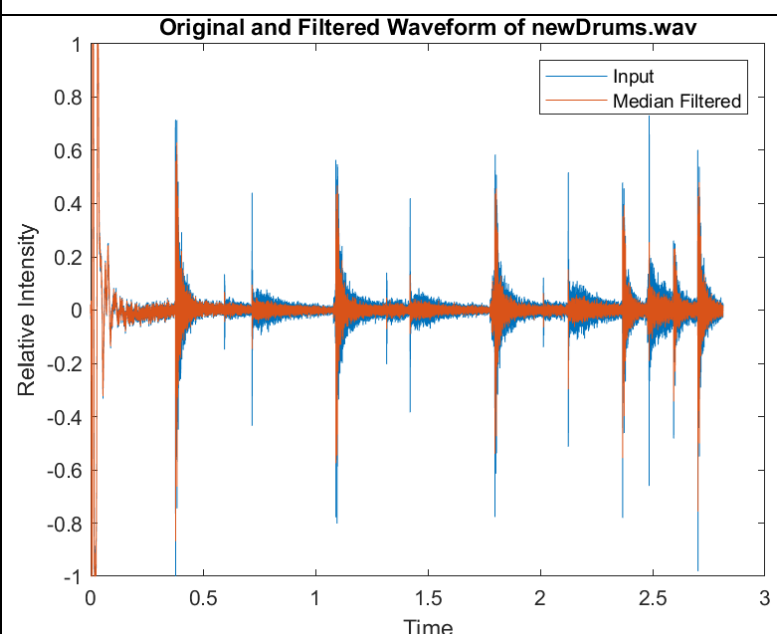
For this audio, the weighted average filter with a gaussian distribution performed the best in the ear-test with a window size of 5. When considering the method by which the gaussian weighted average filter works this makes sense. The two parts of the drum used in this audio are the bass drum (at the beginning) and repetitions of a cymbal. Both of those sounds have an initial spike of high intensity followed

by non-linear decreasing intensity. Since the weighted average filter allows for contributions from previous samples with a normal distribution, noise can be removed while still retaining the effect of previous intensities trailing into later sounds. For instance, the initial bass drum still has a trailing sound when the first cymbal strike occurs, which adds to the music. The weighted average filter does the best at retaining that effect.

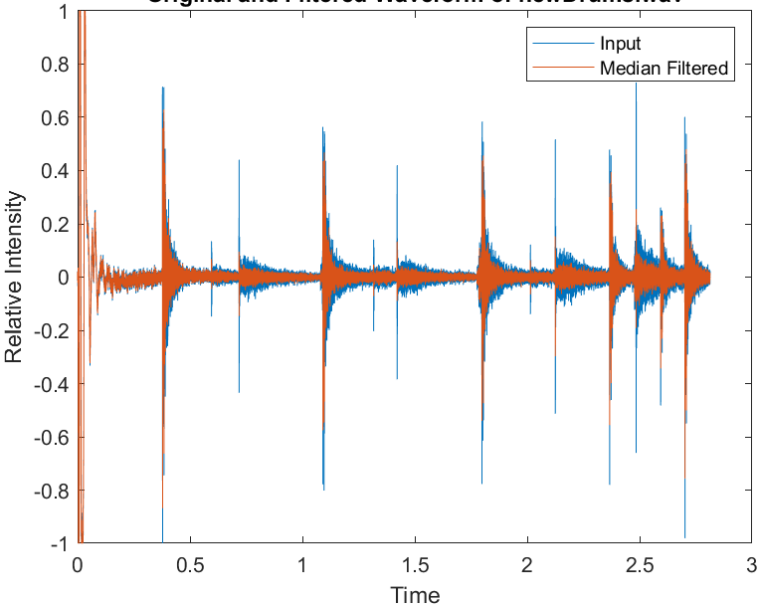
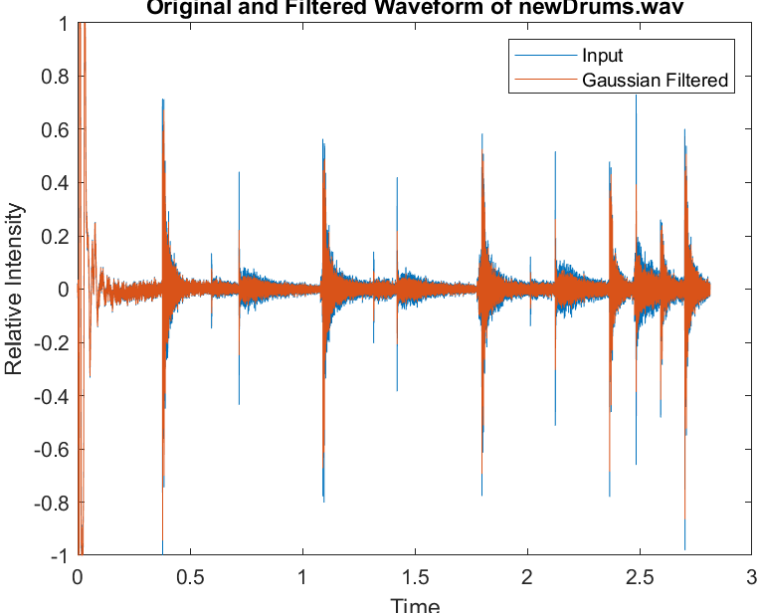
Table 2: Window Size Tuning for newDrums.wav

Filter	Window Size	Distortion Level	Plot
Moving Average Filter	2	Optimal	



Median Filter	4	Optimal	 <p>Original and Filtered Waveform of newDrums.wav</p> <p>Relative Intensity</p> <p>Time</p> <p>Input</p> <p>Median Filtered</p>
	5	Slightly	 <p>Original and Filtered Waveform of newDrums.wav</p> <p>Relative Intensity</p> <p>Time</p> <p>Input</p> <p>Median Filtered</p>



	12	High	<p>Original and Filtered Waveform of newDrums.wav</p> 
Weighted Average Filter	5	Optimal	<p>Original and Filtered Waveform of newDrums.wav</p> 

	7	Slightly	<p>The plot shows the original input waveform (blue) and the Gaussian filtered waveform (orange) for a 'Slightly' filtered audio. The x-axis represents Time from 0 to 3 seconds, and the y-axis represents Relative Intensity from -1 to 1. The filtered waveform (orange) is smoother than the input (blue), with reduced high-frequency noise.</p>
	13	High	<p>The plot shows the original input waveform (blue) and the Gaussian filtered waveform (orange) for a 'High' filtered audio. The x-axis represents Time from 0 to 3 seconds, and the y-axis represents Relative Intensity from -1 to 1. The filtered waveform (orange) is significantly smoother than the input (blue), with most high-frequency components removed, leaving only the low-frequency envelope.</p>

### Speech Audio File

The speech audio is verbal, in verbal audio it is most important that the words, pronunciation, and tone of the speaker are comprehensible. The scenario used to judge the speech audio was if this audio was the output from a voice call service like Zoom. At what point would the output audio be optimal for the listener?

In order to retain clarity and audibility of what the speaker is saying, the best filter should reduce noise but avoid effecting the intensity of the peaks of the audio, as the peaks occur when words are being spoken.

- Moving Average Filter:

- Window Size 3 (optimal distortion): The speaker can be heard clearly with little noise.
- Window Size 5 (slight distortion): The speaker can still be heard but a slight muted effect is present.
- Window Size 20 (high distortion): The speaker can be heard with a strong muted effect, conversations with someone with this level of distortion on a voice call would be undesirable.

- Median Filter

- Window Size 2 (optimal distortion): The speaker can be heard clearly with little noise.
- Window Size 5 (slight distortion): The speaker can still be heard but a slight muted effect is present.
- Window Size 20 (high distortion): The speaker can be heard with a strong muted effect and a static noise. Conversation with someone with this level of distortion on a voice call would be undesirable.

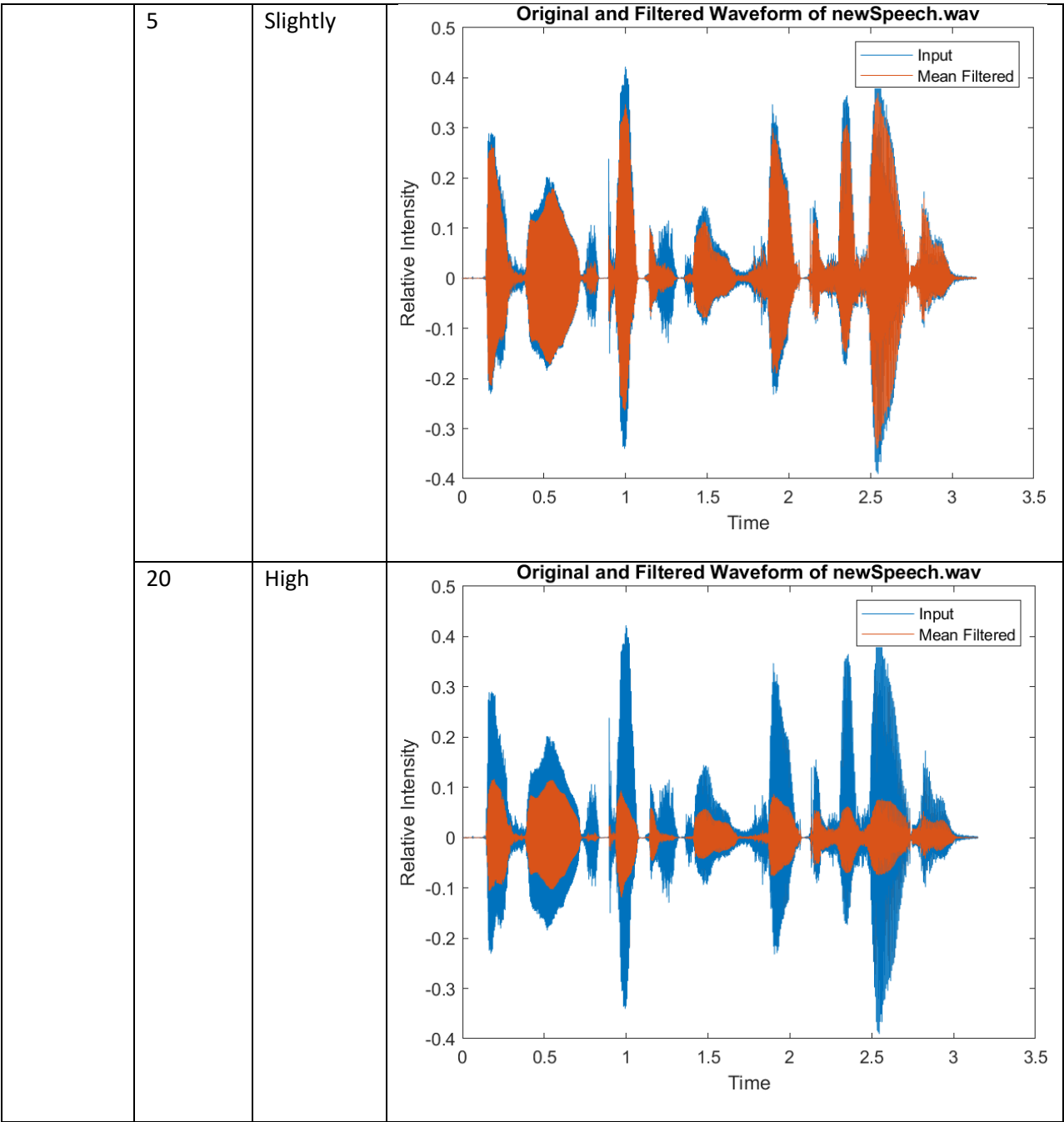
- Weighted Average Filter

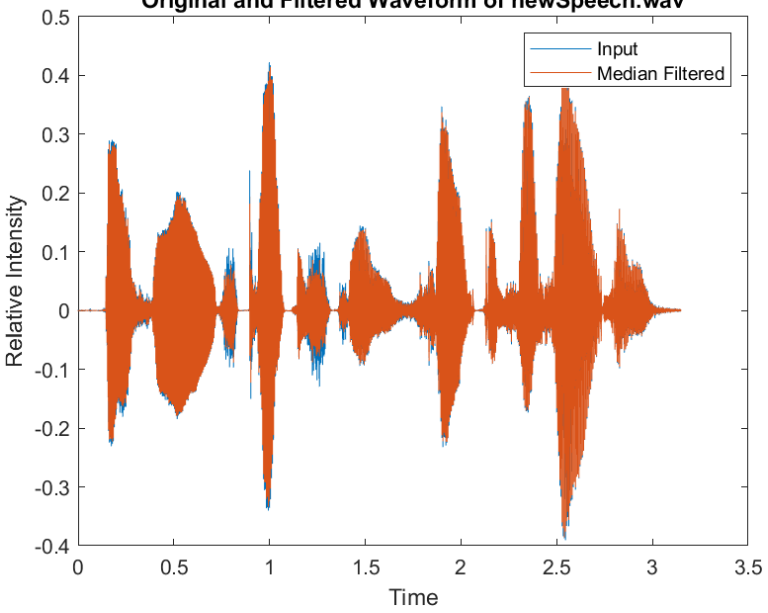
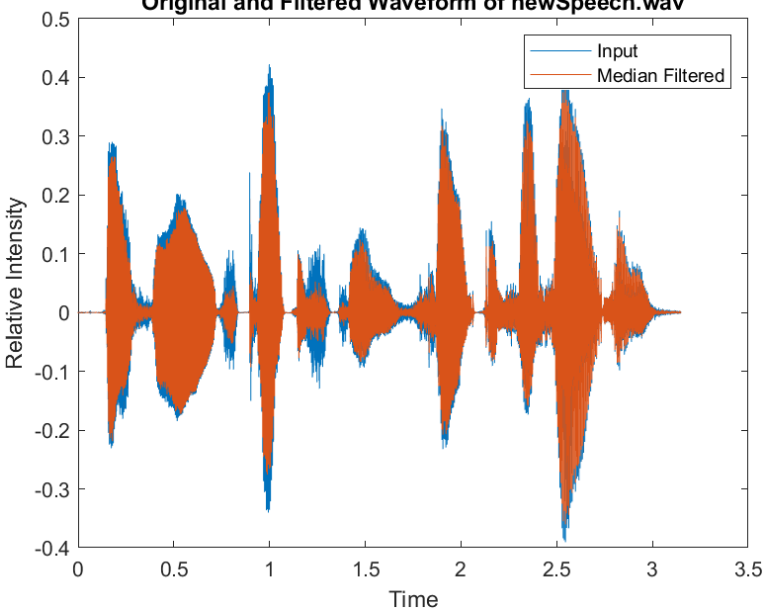
- Window Size 3 (optimal distortion): The speaker can be heard clearly with little noise.
- Window Size 10 (slight distortion): The speaker can still be heard but a slight muted effect is present.
- Window Size 18 (high distortion): The speaker can be heard with a strong muted effect, conversations with someone with this level of distortion on a voice call would be undesirable.

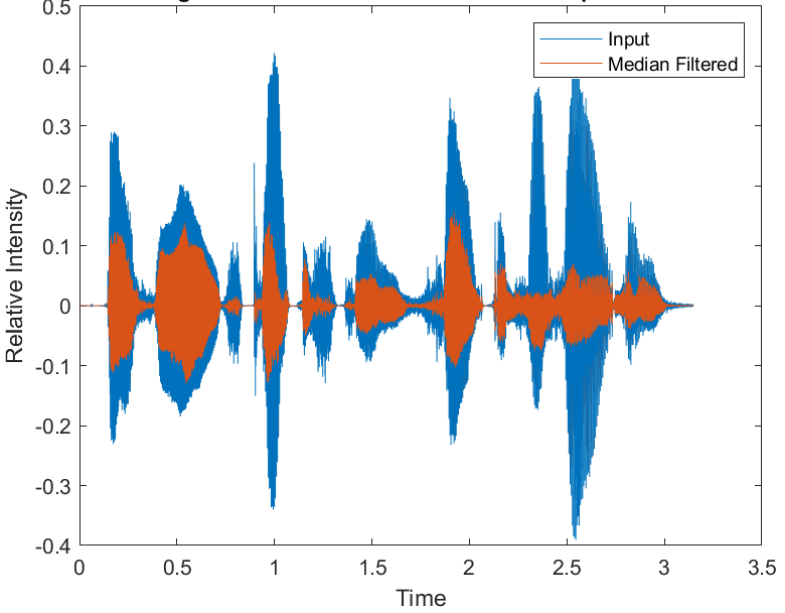
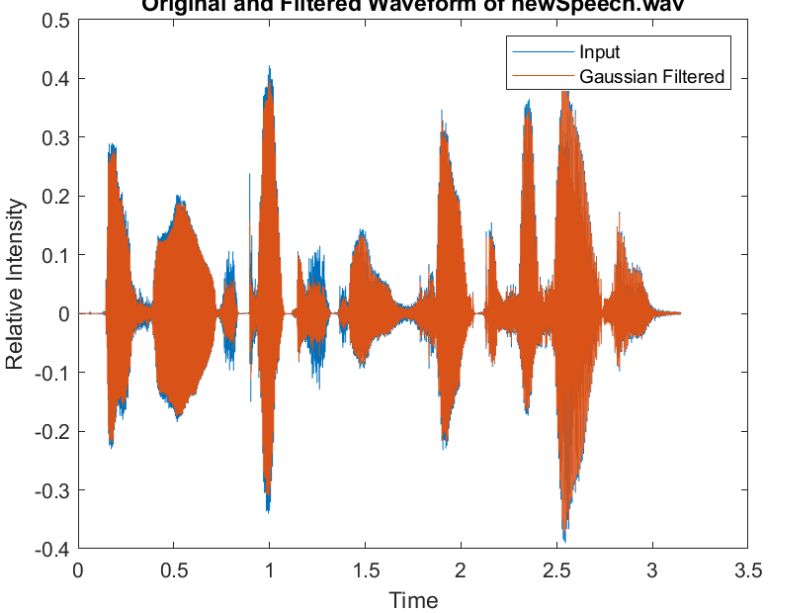
The best filter for this audio was the median filter with a window size of 2. Although at higher window sizes the median filter introduced a static noise that was not present in the other filters, at the optimal window size it was the clearest. This makes sense as the other two filters involve introducing components of nearby samples which would blur sounds together. The median filter chooses only one sample, which avoids a blurring effect and better maintains clarity.

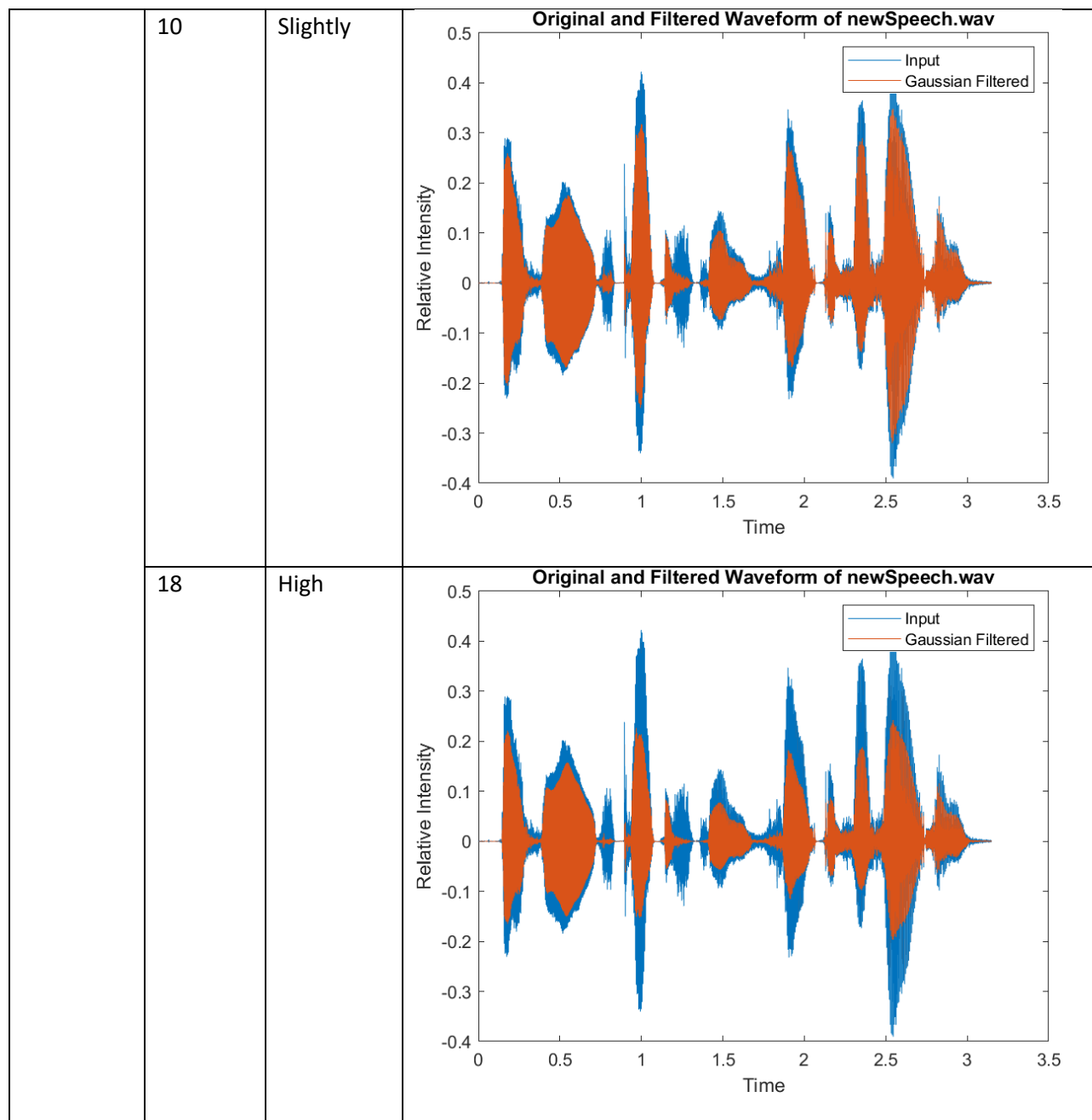
Table 3: Window Size Tuning for newSpeech.wav

Filter	Window Size	Distortion Level	Plot
Moving Average Filter	2	Optimal	<div>Original and Filtered Waveform of newSpeech.wav</div>



Median Filter	2	Optimal	<p><b>Original and Filtered Waveform of newSpeech.wav</b></p> 
	5	Slightly	<p><b>Original and Filtered Waveform of newSpeech.wav</b></p> 

	20	High	<p>Original and Filtered Waveform of newSpeech.wav</p>  <p>This plot displays the original speech waveform (blue line) and its median filtered version (orange line). The x-axis represents Time from 0 to 3.5 seconds, and the y-axis represents Relative Intensity from -0.4 to 0.5. The median filter effectively removes the high-frequency noise present in the original signal, resulting in a cleaner waveform.</p>
Weighted Average Filter	5	Optimal	<p>Original and Filtered Waveform of newSpeech.wav</p>  <p>This plot displays the original speech waveform (blue line) and its Gaussian filtered version (orange line). The x-axis represents Time from 0 to 3.5 seconds, and the y-axis represents Relative Intensity from -0.4 to 0.5. The Gaussian filter smooths the signal, removing high-frequency noise while preserving the overall shape of the speech waveform.</p>



The moving average filter can be portrayed in the form of an impulse response function. In both discrete and continuous time. The example will be given in discrete time, but it is the same for continuous time with the corresponding convolutional sums.

$$y[n] = x[n]h[n]$$

The system function can be written in the following form with the impulse function.



$$h[n] = \frac{1}{L}(u[n] - u[n - L])$$

$$y[n] = \int_{-\infty}^{\infty} x[\tau] h[n - \tau] d\tau$$

$$y[n] = \int_{-\infty}^{\infty} x[\tau] \frac{1}{L}(u[n - \tau] - u[n - L - \tau]) d\tau$$

For the two different impulse functions:

$$u[n - \tau] = 1 \quad \tau \leq n$$

$$u[n - L - \tau] = 0 \quad \tau > n - L$$

Therefore:

$$y[n] = \frac{1}{L} \int_{n-L+1}^n x[\tau] d\tau$$

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

*Equation 4*

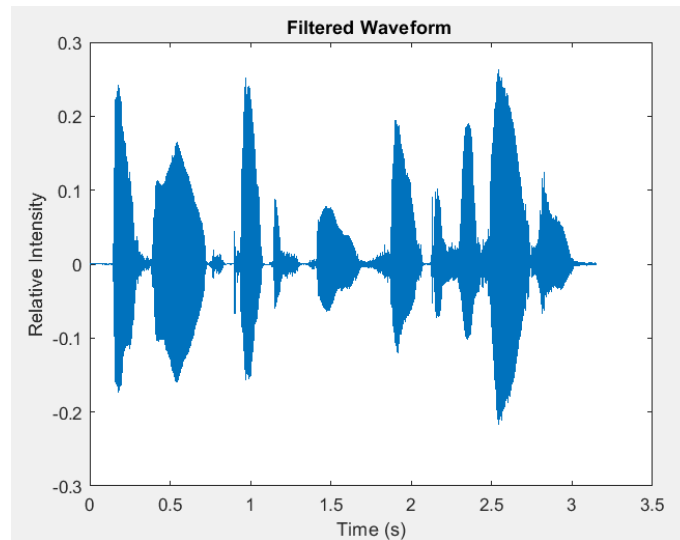
A high pass filter blocks low frequency signals while allowing high frequency signals to pass. The filter has a cutoff frequency and removes signals with frequencies lower than the cut-off.

By examining equation 1, the moving average filter can act as a low-pass filter by taking the average of many samples. However, at low window sizes, it will act as a high-pass filter and only have a significant effect on low-frequency signals.

### 8.3 Part 3

#### Syllable Counting Algorithm

The syllable counting algorithm produced an output of 10 syllables. This was manually verified by listening to the speech file, and is correct. Figure 7 shows the filtered signal that the findpeaks function is used on.



*Figure 7: Mean Filtered Speech Waveform for Syllable Detection*

As seen in Figure 7, there are 10 easily visible and separated local maximums, indicating 10 syllables in the file.

#### Beats Per Minute Algorithm

The beats per minute algorithm found the number of drum beats in the newDrums.wav file to be 14. Through manual counting, this was verified as the correct number of beats. The number of beats detected in the file was then extrapolated into the number of beats expected in one minute. This number ended up being 298.5 beats per minute. With this algorithm as well as the syllable counting algorithm, one of the drawbacks is that the algorithms assume a relatively consistent tempo/rhythm, which may not always be true. The algorithms also require individual tuning of minimum peak height and minimum peak distance depending on the application, which is also a drawback.

#### Detecting Silent Regions Algorithm

To detect regions of silence, a somewhat arbitrary threshold was defined in relation to the sample of sound with the greatest magnitude. Any samples with a lower magnitude than this threshold had its value set to 0, and thus no sound will be played when passing over these regions.

Its implementation in Matlab simply required iterating through all row elements of the matrix outputted by audioread and comparing their values to this cutoff threshold. A plot of the waveform with regions fully silenced is plotted alongside the original input and its waveform with a moving average filter applied; this is shown in Figure 8.

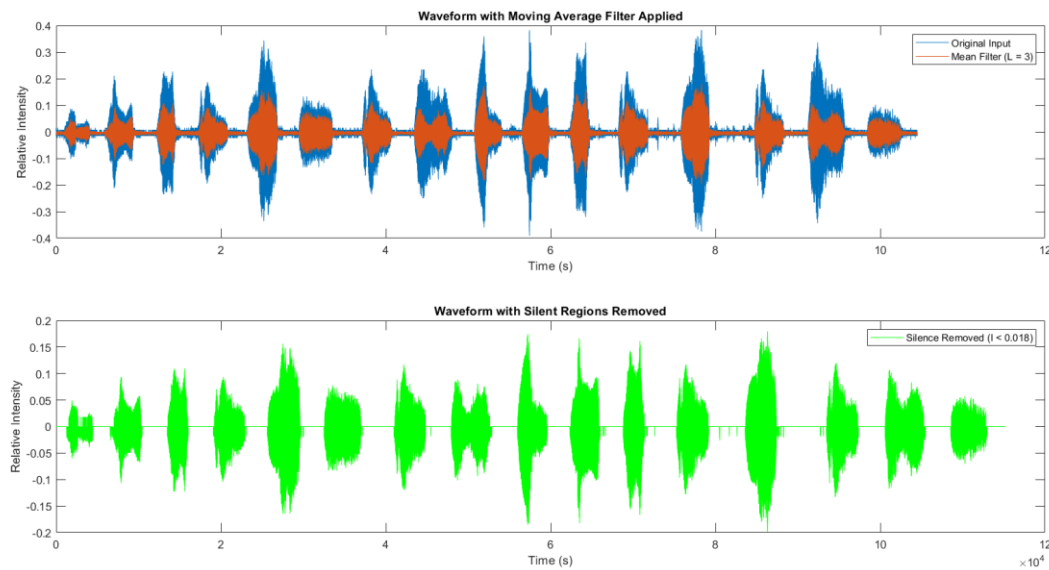


Figure 8: Comparison of the waveforms of newBirds.wav, the waveform with a moving average filter applied ( $L = 3$ ), and where silent regions were removed from the filtered waveform.

The silent regions are represented in the green waveform, at the regions where the waveform had no amplitude, collapsing onto the horizontal axis.

## 9 Conclusions and Recommendations

This project saw the design and implementation of a 'Moving Average Filter', 'Median Filter', and 'Weighted Average Filter'. These digital filters were applied to three different sound files and analysis on the optimal filter and window sizes were conducted based on the unique requirements of each sound type. For the 'birds' audio, the optimal filter was a median filter of window size 4, as it successfully

removed the noise from other birds in the background without decreasing the peaks of the main bird significantly. For the 'drums' audio, the optimal filter was a gaussian weighted average filter of window size 5, because the trailing sounds of the bass drum and cymbal were preserved. For the 'speech' audio, the optimal filter was a median filter of window size 2, because the output was clear, and noise had been filtered out. The project also involved developing algorithms for counting number of syllables of the speech file, determining beats per minute of the drums, and detecting silent regions in the birds audio. The algorithms were implemented with a combination of filtering and using Matlab's built-in findpeaks function. The syllable and beats per minute algorithm both produced accurate and correct results, and the silence detection algorithm correctly identified and displayed silent regions in a plot.

## 10 Acknowledgements

This work was conducted as part of the Linear Systems and Signals course at University of Waterloo. The authors are also particularly grateful to Dr. Charbel Azzi for instructing the course, and the teaching assistants for their efforts in grading this project.

## 11 Works Cited

- [1] J. O. S. III, "INTRODUCTION TO DIGITAL FILTERS," Center for Computer Research in Music and Acoustics (CCRMA), September 2007. [Online]. Available: <https://ccrma.stanford.edu/~jos/filters/>. [Accessed November 2022].
- [2] UC Davis, "INTRODUCTION TO DIGITAL FILTERS," [Online]. Available: [https://123.physics.ucdavis.edu/week\\_5\\_files/filters/digital\\_filter.pdf](https://123.physics.ucdavis.edu/week_5_files/filters/digital_filter.pdf). [Accessed November 2022].

```

clc;
close all;
clear all;

% PART 1

% process_file("Birds.wav");
process_file("Drum.wav");
% process_file("Speech.wav");

% For reading and processing audio
function [audio_data, sample_rate] = process_file(filename)
    % Read audio file and obtain sampling rate
    [audio_data, sample_rate] = audioread(filename);
    % disp(sample_rate)

    % Add the audio's two channels if the audio is stereo
    [~, cols] = size(audio_data); % or use
    audioinfo(filename).NumChannels
    if cols == 2
        audio_data = audio_data(:, 1) + audio_data(:, 2);
        % disp(filename + " is two channels.")
    end

    % Playing the audio file and creating a new file
    sound(audio_data, sample_rate);
    new_filename = 'new' + filename; % These are STEP 4
files
    audiowrite(new_filename, audio_data, sample_rate);

    % Plotting the audio waveform
    t = linspace(0, length(audio_data)/sample_rate,
length(audio_data));
    plot(t, audio_data);
    title('Waveform of ' + filename);
    xlabel('Time (s)');
    ylabel('Relative Intensity');

    % Resampling the audio to 16000 samples
    audio_data = resample(audio_data, 16000, sample_rate);

    % These are STEP 6 files
    new_filename = 'step6_new' + filename;

```

```
    audiowrite(new_filename, audio_data, 16000);  
end
```

```
clc;  
close all;  
clear all;  
  
% PART 2  
  
% Read the audio file(s)  
% filename = "newBirds.wav";  
filename = "newDrum.wav";  
% filename = "newSpeech.wav";  
[audio_data, sample_rate] = audioread(filename);  
  
% Call a filtering function  
% filtered_1 = mean_filter(audio_data, sample_rate,  
filename);  
% filtered_2 = median_filter(audio_data, sample_rate,  
filename);  
filtered_3 = gaussian_filter(audio_data, sample_rate,  
filename);  
% sound(audio_data, sample_rate);  
% sound(filtered_3, sample_rate);  
  
% Moving average filter  
function y = mean_filter(audio_data, sample_rate, filename)  
    % Setting window size  
    window_size = 13;  
  
    % Defining the 'window' for the filtering  
    % b = [1/windowSize, 1/windowSize, 1/windowSize]  
    b = (1/window_size) * ones(1, window_size);  
  
    % Using the filter function for filtering the audio  
    signal with the  
    % 'window'  
    % Denominator (second parameter) is set to 1  
    y = filter(b, 1, audio_data);  
  
    % Plotting the original unfiltered signal and the  
    filtered signal  
    label = "Average (L = " + window_size + ")";
```

```

    plot_audio(audio_data, y, sample_rate, label,
filename);
end

% Moving median filter
function y = median_filter(audio_data, sample_rate,
filename)
    % Initializing y equal to input signal for now
    y = audio_data;

    % Setting window size
    window_size = 2;

    for n = window_size : length(audio_data)
        % Note that window_array is an array of x[n], x[n-
1], ..., x[n-k]
        window_array = ones(1, window_size);
        for i = 0 : window_size-1
            window_array(i+1) = audio_data(n-i);
        end

        % Note that y[n] = median(x[n], x[n-1], x[n-2],
..., x[n-k])
        y(n) = median(window_array);
    end

    % Plotting the original unfiltered signal and the
filtered signal
    label = "Median (L = " + window_size + ")";
    plot_audio(audio_data, y, sample_rate, label,
filename);
end

% Moving weighted average filter
function y = gaussian_filter(audio_data, sample_rate,
filename)
    % Setting window size
    window_size = 3;

    % Setting weight coefficients using gausswin
    w = gausswin(window_size);
    w = w ./ sum(w);
    disp(w);

    % Filtering using the filter function and the weights

```



```

        y = filter(w, 1, audio_data);

        % Plotting the original unfiltered signal and the
        filtered signal
        label = "Gaussian (L = " + window_size + ")";
        plot_audio(audio_data, y, sample_rate, label,
filename);
end

% For plotting original and filtered waveforms together
function [] = plot_audio(x, y, sample_rate, label,
filename)
    t = linspace(0, length(y)/sample_rate, length(y));
    plot(t, x, t, y);
    legend('Original Input', label);
    title('Original and Filtered Waveform of ' + filename);
    xlabel('Time (s)');
    ylabel('Relative Intensity');
end

```

```

clc;
close all;

% find_syllables();
% beats_per_minute();
find_silent_regions();

% findSyllables reads in the newSpeech file and find number
of syllables in
% the file
function [] = find_syllables()
    % Read in audio data
    [audio_data, sample_rate] = audioread('newSpeech.wav');
    disp(sample_rate);

    % Calling the mean filter
    y = mean_filter(audio_data, 10);

    % Finding peaks with minimum peak distance and minimum
    peak height
    % Minimum distance and height are tuned based on the
    filtered waveform

```

```

    pks = findpeaks(y, 'MinPeakDistance', 2500,
'MinPeakHeight', 0.075);

    % When min distance and height are tuned correctly,
number of peaks
    % detected is the number of syllables
    disp(length(pks));

    % Plotting the filtered waveform for visual analysis
    t = linspace(0, length(y)/sample_rate, length(y));
    plot(t, y);
    title('Filtered Waveform ');
    xlabel('Time (s)');
    ylabel('Relative Intensity');
end

% Finds beats per minute from drum file
function [] = beats_per_minute()
    % Read in audio data for drum file
    [audio_data, sample_rate] = audioread('newDrum.wav');

    % Filter audio with mean filter
    y = mean_filter(audio_data, 3);

    % Get number of beats in the audio file
    peaks = findpeaks(y, 'MinPeakHeight', 0.1,
'MinPeakDistance', 1575);
    num_beats = length(peaks);
    disp("Number of beats in file: " + num_beats);

    % Convert number of beats in audio file to beats per
minute using
    % length/time (s) of the file
    time_length_seconds = length(y) / sample_rate;
    bpm = (60 / time_length_seconds) * num_beats;
    disp("Predicted beats per minute: " + bpm);

    % Plotting the filtered waveform for visual analysis
    t = linspace(0, length(y)/sample_rate, length(y));
    plot(t, y);
    title('Filtered Waveform ');
    xlabel('Time (s)');
    ylabel('Relative Intensity');
end

```

```

% Finds silent regions
function [] = find_silent_regions()
    % Read in audio data for birds file
    [audio_data, sample_rate] = audioread('newBirds.wav');

    % Setting window size
    window_size = 3;
    legend_for_filtered = "Mean Filter (L = " + window_size
+ ")";

    % Filtering audio with mean filter
    y = mean_filter(audio_data, window_size);

    % Removing audio of relative intensity below a certain
threshold
    z = y;
    silence_threshold = round(abs(max(z)) * 0.1, 3);
    legend_for_silence_removed = "Silence Removed (I < " +
...
        silence_threshold + ")"; % here 'I' represents the
intensity
    for i = 1 : length(z)
        if (abs(z(i)) < silence_threshold)
            z(i) = 0; % pulling the intensity to 0 if it
considered silent
        end
    end
    sound(y, sample_rate)
    %sound(z, sample_rate)

    subplot(2, 1, 1);
    t = linspace(0, length(audio_data)/sample_rate,
length(audio_data));
    plot(t, audio_data, t, y);
    title('Waveform with Moving Average Filter Applied');
    xlabel('Time (s)');
    ylabel('Relative Intensity');
    legend('Original Input', legend_for_filtered);

    subplot(2, 1, 2);
    plot(z, 'g');
    title('Waveform with Silent Regions Removed');
    xlabel('Time (s)');
    ylabel('Relative Intensity');

```

```
    legend(legend_for_silence_removed);  
end  
  
% Moving average filter (taken from part 2, but without  
audio plotting)  
function y = mean_filter(audio_data, window_size)  
    % Defining the 'window' for the filtering  
    % b = [1/windowSize, 1/windowSize, 1/windowSize]  
    b = (1/window_size)*ones(1, window_size);  
  
    % Using the filter function for filtering the audio  
    signal with the  
    % 'window'  
    % Denominator (second parameter) is set to 1  
    y = filter(b, 1, audio_data);  
end
```