Voice Modification

Vicki Chen
ECE 160
UCSB
vchen@umail.ucsb.edu

***Abstract*—To produce a voice modification program which has certain restraints such as constant speed and pitch, we must explore other variables of the sound component such as frequency, delay based audio effects, and other filters which can alter the input signal and produce an output signal that appears unrecognizable to one's ear. This program utilizes an echo filter, white noise, and a bandpass filter to create a final signal where the speech is still clear to the listener, but identity of the original speaker appears unfamiliar.**

I. Introduction

Voice modification programs are powerful tools which can mask the identity of the speaker by often altering the tone and pitch or creating a combinational distortion to the sound. The earliest applications of voice modulators were electronic devices used over the telephone for the purpose of disguise.

In this challenge, we are told to create a voice modification tool where the original speed of the audio should not be decreased or increased and the pitch should remain relatively the same. To create a program that fits these requirements, I created a simple voice modifier that utilizes several filters to produce an output signal which hides the speaker's identity.

II. Method

To read the sound file I recorded, I utilized the *audioread* function in Matlab:

```
[x,Fs] = audioread('speech.wav');
```
(1)

This reads the data from the file and returns sampled data *x* and a sample rate *Fs*.
Given that speed and pitch are fixed, I created and applied several filters to the modified audio file:

A. Echo

An echo is a repetition of the sound signal after a fixed amount of time period. Since the application of this is linear, we can use a finite-duration impulse response (FIR) filter. A FIR filter is a digital filter that can provide a linear phase response created by design methods that are generally linear. The Transfer function of a FIR filter with length M is defined as

$$y[n] = \sum_{k=0}^{M-1} h(k)x(n-k) \quad (2)$$

After loading the audio file, I created new variables *d* for delay and *a* for echo strength. The delay and echo strength can be set to any percentage between 0-100. For this example, I set $d = 0.15$ which indicates a 0.15 second delay, and $a = 1.0$, indicating 100% echo strength. A full echo strength will produce a sound identical to the original. Another variable we must obtain is *D*, the number of samples during the delay. This is calculated by multiplying sample rate *Fs* and delay time *d.* These variables will be used below in the echo equation.

After initialization the output music signal using:

```
echo = zeros(size(x));
```
(3)

To generate an echo filter, I created a FIR [1] equation:

```
echo(i) = a*x(i-D) + x(i);
```
(4)

B. Noise addition

The second modification I added to the audio is simply white noise. A white noise signal is constituted by a set of independent and identically distributed random variables. In more precise terms, white noise signals can be created using the *randn* function in Matlab which generates random numbers that follow a Gaussian distribution.

Using the equation below, we can generate Gaussian noise where the amount of noise can be controlled by increasing or decreasing the fraction size:

```
noise = echo + randn(size(echo))* (1/100);
```
(5)

C. Bandpass Filter

A bandpass filter is a device that allows signals between two specific frequencies to pass and rejects frequencies out of that range. To apply this filter, I used a *fir1* equation (4) which implements the classical method of windowed linear-phase FIR digital filter design. It designs filters in standard low-pass, high-pass, band-pass, and band-stop configurations. By default, the filter is normalized so that the magnitude response of the filter at the center frequency of the passband is 0 dB.

```
b = fir1(46,[100/Fn 700/Fn]);
```
(6)

b = fir1(n,Wn) returns vector b containing *n+1* coefficients of an order n lowpass FIR filter. This is a Hamming-window based, linear-phase filter with normalized cutoff frequency *Wn*. The output filter coefficients, b, are ordered in descending powers of z

$$B(z) = b(1) + b(2)z^{-1} + \cdots + b(n+1)z^{-n} \quad (7)$$

Wn = [w1 w2], fir1 returns a bandpass filter with passband w1 < ω < w2 where

$$w1 = \frac{low\ cut\ off\ frequency}{Sampling\ rate\ Fs} \quad (8)$$

$$w2 = \frac{high\ cut\ off\ frequency}{Sampling\ rate\ Fs} \quad (9)$$

For this setup, I chose *n,* the filter order specified as an integer scalar, to be 46 since you must use an even order for symmetric filters such as highpass and bandpass. *Fn* is the Nyquist frequency and is calculated by dividing the sample rate *Fs* by 2. The low frequency cutoff will be 100Hz and the high frequency cutoff is 700Hz. This will produce an effect similar to that of a lowered pitch.
After creating the bandpass configuration, we can apply it as a filter to the audio after noise addition:

```
y = filter(bandpass,1,noise);
```  (10)

Equation (5) follows the built in Matlab filter function *y = filter(b,a,x)* which filters the input data *x* using a rational transfer function defined by the numerator and denominator coefficients *b* and *a*. Since *x* is a matrix, the filter acts along the first dimension and returns the filtered data for each column.

III. Results

After running the program, the modified speech audio *y(t)* is played back and saved as *modified.wav*. To compare the modifications to the original audio, I plotted graphs of the sound after each filter is added.

Modified Waveforms

I. Echo

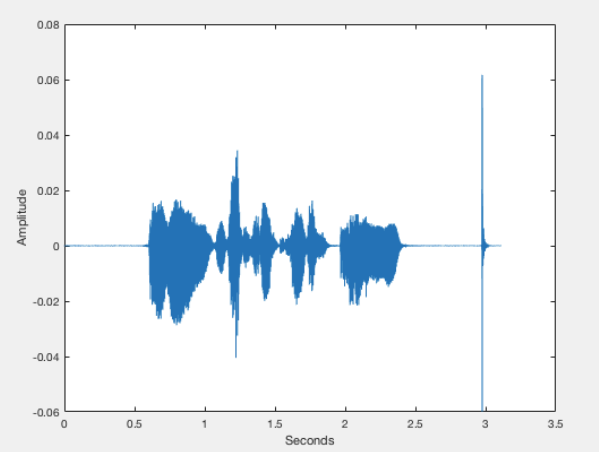

Fig. 1. Graph of Original Audio

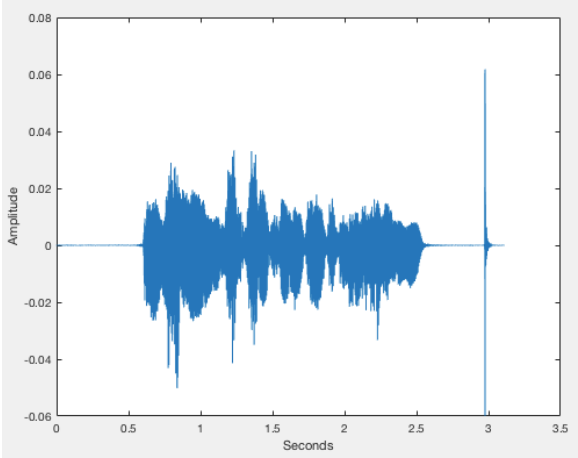

Fig. 2. Graph of Modified Audio with Echo

From the two figures, we can see that the amplitude of the sound is never zero. This is because we added in an echo which is played after 0.15 sec of delays, resulting in a sound wave that always has input and the amplitude remains greater than zero throughout the duration of the audio file.

II. NOISE

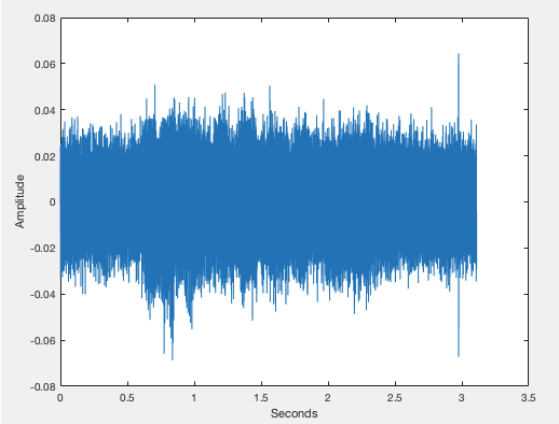

Fig. 3. Graph of Audio with Noise

TABLE I. ORIGINAL AUDIO MATRIX

x	y
0	0
0	0
0	0
3.05175781250000e-05	0
3.05175781250000e-05	0
3.05175781250000e-05	0
3.05175781250000e-05	0
0	0

TABLE II. AUDIO WITH NOISE

x	y
-0.0110152957303158	-0.00146727819991644
0.00591250369185888	0.0230717707141794
-0.00697952456691747	-0.00457719320313392
0.00178373259121270	0.0106413208059228
0.00551867274563880	0.00375116136571528
-0.0187983433525236	-0.00306758148137117
-0.00577331596140203	-0.0107665300604079
0.0137526081856429	-0.0111730622416703

Since the *randn* function generates a *n x n* dimension matrix full of random numbers, we can see from the comparison that although the original matrix contained 0's in certain *(x, y)* values, all the elements are filled in by a random number generated by the noise function in the new transformed matrix. As seen in fig 3, the output signal maintains a constant amplitude throughout the duration of time. This is defined by the function we generated (3) and the fraction size which produced a continuous number of random signals throughout the audio.

III. BANDPASS

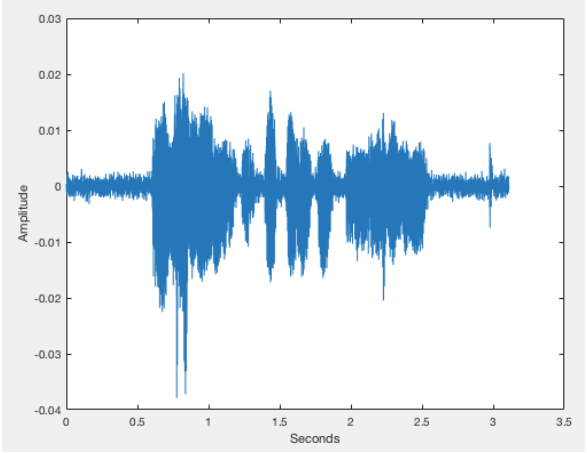

Fig. 4. Graph of Audio after Bandpass Filter Added

After applying the final filter, the resulting output signal did not experience any changes in speed or pitch.

IV. CONCLUSION

The modified audio correctly masks the identity of the speaker since the tones appear altered. Some of the things I noticed during testing out the code is that the order in which the filters are applied effects the output sound. By playing around with the order of the noise addition and bandpass filter, I realized to create a sound similar to an old static radio I had in mind, noise addition had to be applied in the bandpass filter as well to create the ambience.

REFERENCES

[1] Mathworks.com (2018). *Window-based FIR filter design – MATLAB fir1*. [Online]. Available: https://www.mathworks.com/help/signal/ref/fir1.html