Introduction

By recent advances in signal processing, a new area of research has been opened termed as “Sound transformation”. The topic covers a broad range of transformations on different types of sound. One of the recent research directions is on transformations of voice. You have most likely heard the electronic change of voice pitch in pop music (a very good example is “Believe” by Cher). One of the applications of the pitch-change is that anyone can become a singer without having learned how to sing properly! One can sing and record a song and then the false notes can be corrected electronically by tuning them using pitch changing. More sophisticated signal processing methods allow voice morphing, where the voice of speaker A is changed in a way, such that it seems as if speaker B has said those things.

A simple approach to changing the voice is to change the playback speed. This however makes the voice sound funny (in fact this is referred to as Mickey Mouse effect). The approach used in this lab is a frequency shift method. Assume that the spectrum of the sound waveform to be processed is what shown in figure 1-a. Our goal is to change the spectrum to what is shown in figure 1-b.

![Figure 1](image)

**Figure 1** – (a) Spectrum of the input voice, (b) Spectrum of the processed voice, note that the spectrum is shifted by $\Delta f$.

Note that this cannot be done by simply modulating the signal by a cosine waveform for small amounts of shift.

Questions:
1. Assume you have a recorded sound waveform $x(t)$ with a bandwidth of 10 KHz. Derive the bandwidth of $x(at)$ as a function of $a$. This is the bandwidth of the signal when played back at a speed different from its original. Describe the effect of this when $a < 1$ and $a > 1$ both in time and frequency.

2. Draw qualitatively the spectrum of $x(t)\cos(1000pt)$ assuming that the bandwidth of $x(t)$ is 2 KHz. Do you see the problem why this won't shift the frequency content as depicted in figure 1-b?
**Objective**

The goal of this experiment is to make a system that shifts the spectrum of an input voice signal by a desired amount. Through this you will learn how to program filters on the DSP board and use MATLAB to design FIR filters.

**Procedure**

You’ll use the same example program as was used in experiment 3 as the initial program and add your own code to it to make the desired system. Refer to experiment 3 if you don’t remember where you should add your code.

Let’s now figure out the procedure for shifting the frequency. Assume that the signal you have has a spectrum as depicted in figure 2-a. We will first multiply the signal with a cosine waveform with a frequency at least twice the maximum frequency present in the signal to get the signal spectrum shown in figure 2-b. Now we need to get rid of the left half of the spectrum (in positive frequency). A well designed high pass filter will do this job and we get the signal depicted in figure 2-c. Now we multiply this signal with another cosine waveform. The resulting spectrum will be as shown in figure 2-d. Now we need to discard the higher frequency portion to get the desired shifted spectrum. This can be accomplished using a well designed low pass filter.

Now let’s look at the constraints that we have. The ADC has a limited sampling frequency and we’re working with samples of the signals, so when multiplying the signal with the cosine waveform in the first step above, the sampling frequency of the ADC must still be at least twice that of the maximum frequency of the resulting signal. This can not be accomplished if the input sound has high frequency content (between 10~20 KHz). So we need one more step at the beginning, which is to low pass filter the input sound to make sure its frequency content remains in the lower range.

![Figure 2– (a) Spectrum of the input voice, (b) Spectrum of the result of multiplication by \( \cos(2p f_1 t) \) (dotted lines show original spectrums), (c) HPF applied to the result, (d) Spectrum of multiplication result by \( \cos(2p f_2 t) \), (e) Final LPF result](figure2.jpg)
To summarize, these are the steps that need to be taken:
1. Low pass filter the input sound.
2. Multiply with a high frequency cosine signal.
3. High pass filter.
4. Multiply with a cosine waveform.
5. Low pass filter.

Extra Features
You may want to be able to vary $Df$, the frequency shift, while the code is running. This can be done by using the DIP switches on the board to select the value of $Df$. Having 4 different values will be enough. You can use whatever shifts you like. Some values could be 50, 100, 150 and 200 Hz.

Filter Designs
To design filters, use MATLAB’s signal processing toolbox sptool command. If this is new to you, review the filter design II lab instructions in EE2304.

Choose equiripple FIR as the filter type and minimum order from the left pane. You need to find the time domain samples of the FIR filters to be able to use them in your code. After exporting the samples to MATLAB’s workspace, you can create tables in your code that contain the sample values. You should then write a function to perform the convolution in time. Note that the convolution is performed on a frame of input sound, whose length is defined by the length of the input buffer. It is very important that the indices you use don’t exceed the lengths of the arrays.

Using high order filters produces better results, but you should also consider the processing time required for these filters. You may need to compromise quality for the sake of faster processing. Also have in mind not to use high order filters before making sure your design is proper. Changing the filter response sample values is tedious for higher order filters.

It is very important that you choose the filter parameters carefully. Also make sure that you specify the correct sampling rate when using sptool.

Cosine Multiplications
In order to improve the performance of the system and reduce computational load, you may want to create tables containing the samples of the cosine waveforms you are using at the beginning of the program, i.e. as global tables evaluated in the main function. You can choose the length of buffers in such a way that one of the cosines with frequency $f_1$ or $f_2$ will have all of its cycles completely in the buffer, i.e. no part of the cycle is cut off due to reaching the end of buffer. When creating the tables, multiply the values of one of the cosines by 4 (see question number 1).

The choice of frequencies $f_1$ and $f_2$ is very important and depends on the lowpass filter you’ve designed, as well as the sampling frequency. If you decided to implement the extra features, assign a constant $f_1$ and alter $f_2$, this way the filters need not be changed as the frequency shift changes.
Questions:
1. Derive the Fourier transform of $x(t) \cos(2\pi ft)$ in terms of $X(f)$, the Fourier transform of $x(t)$. Describe why you need to multiply the entries in your cosine table with 4.
2. What are the parameters for the filters you used? How did you choose these parameters?
3. Assuming you want to have a given frequency shift, $Df$, how do you choose the frequency $f_2$ of the second cosine waveform in terms of $f_1$ and $Df$?
4. Describe the quality of the processed voice, as you increase the frequency shift? Why do you think there is such a trend?
Instructor Sign-off Sheet

How well is the code written? (Commented, organized, etc.)

Are the filters designed properly?

Does the system work properly?

Are extra features present?