Musical Transposition for Monophonic Audio Files

Joe McInvale

EE 384 Project Report

Introduction

The goal of this project is to use the concepts of the Fast Fourier Transform (FFT), Inverse Fast Fourier Transform (IFFT), and frequency shifting in order to accurately transpose the key of a given "wav" audio file by a desired amount of semitones. However, due to limitations from the MATLAB *fft* function, it is important to state that this project will be limited to monophonic audio signals, which are further explained in the *"Theory"* section of this report. This report will include a detailed description of the procedure used to conduct the desired transposition, as well as figures and MATLAB code that outline the processing of the provided audio signal. Included deliverables include time-domain and frequency-domain representations of the audio signal both before and after transposition.

Theory

Musical transposition is the idea of converting a provided musical tune to an alternative key. This is accomplished by evenly adjusting the semitones of each pitch so that each note is now accurately adjusted to fit the new key. Observe the keyboard shown in Figure 1. Each labeled key represents a semitone as well as a key signature that a song can be composed in. For example, if a song is written in the key of G and it needs to be transposed to the key of C, it can either be transposed up by 5 semitones or down by 7 semitones, depending on what is preferred. In total, an octave is made up of 12 semitones.

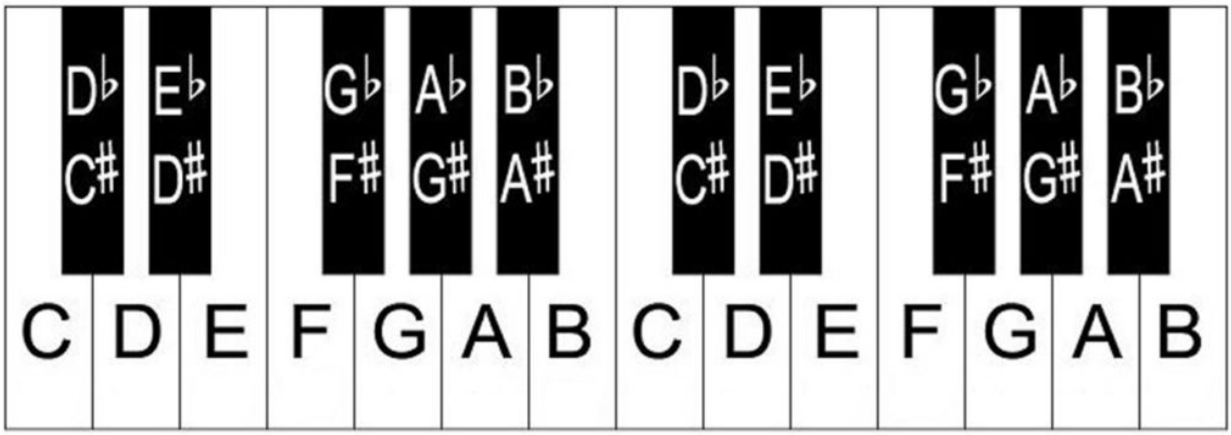

Figure 1. Standard piano keyboard with labeled note names

As mentioned in the *"Introduction"* section of the report, this transposition simulator will only operate properly for monophonic wav audio files due to limitations with the *fft* function in MATLAB. For an audio file to be monophonic, its tune must consist of a single, unaccompanied melodic line. For example, an audio file that consists of a tune coming from a single instrument or solo voice is considered monophonic, whereas an orchestral or choral arrangement is not considered polyphonic due to music being played or sung by concurrent instruments or vocalists, respectively. While this is a limitation to the transposition simulator, it still allows for composers to transpose individual melodies played by soloists within existing polyphonic music.

In order to accomplish musical transposition, we use the properties of the Fast Fourier Transform (FFT) in order to shift frequencies in the signal's frequency domain so that the pitches can be transposed to a new key. According to *The Scientist and Engineer's Guide to Digital Signal Processing*, by Steven W. Smith, Ph.D., "The FFT operates by decomposing an N point time domain signal into N time domain signals each composed of a single point. The second step is to calculate the N frequency spectra corresponding to these N time domain signals. Lastly,

the N spectra are synthesized into a single frequency spectrum." This process can be visualized using the "butterfly diagram" shown below.

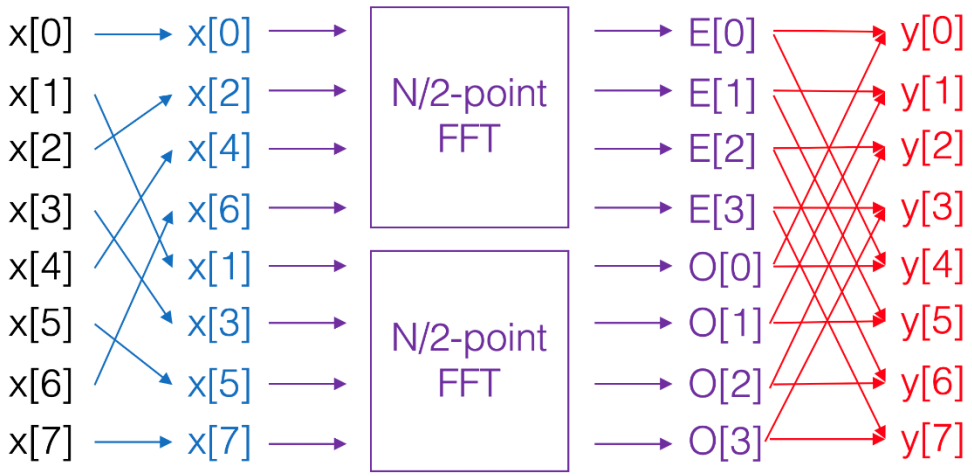

Figure 2. Decimation in Time FFT "butterfly diagram"

Computing these discrete time-domain and frequency-domain values allow for the adjustment of specific frequencies, which directly produces in the transposition of the given audio signal. Then, to convert back from the signal's adjusted frequency-domain back to the time-domain, the Inverse Fast Fourier Transform (IFFT), the reverse of the FFT, can be used.

Simulation

The transposition algorithm multiple parts, including the use of FFT, frequency shifting, and IFFT. The code first asks for user input for two values: the name of the audio file (variable name: *filename*) and the desired increase/decrease in semitones. The name of the file can be entered as a string, while the change in semitones is entered as a number. As explained in the prompt that is displayed to the user, an entry of "+3" would cause the provided audio signal to be transposed up by 3 semitones and then an entry of "-2" would produce a transposition of 2 semitones down.

The +/- variables indicate whether or not the transposition will move the key signature of the audio file up/down, respectively.

Once the user inputs are received, the wav audio file is read using the *audioread()* MATLAB function, which is a requirement in order to parameterize the audio file as a vector (in this case *y*) with a sampling frequency of *fs*. Then, this audio file "*y*" is converted from the time-domain to the frequency domain through the use of the *fft()* MATLAB function. This will compute the Fast Fourier Transform as described in the "*Theory*" section of the report. Then, the algorithm will produce two of the four plots seen in the end result: the time-domain and frequency-domain plots of the original audio file, before being transposed.

After the original audio file is converted to its frequency domain through the use of the *fft()* function, the signal undergoes frequency shifting in order to be transposed to the desired new key. This is accomplished by splitting the FFT into its lower and upper halves, and then shifting and resampling each of these halves to the appropriate new frequencies before getting stitched back together to make one signal. The shifting portion of the algorithm uses the relationship below:

$$New\ note\ frequency = (known\ note\ frequency) \times 2^{\frac{semitones}{12}}$$

In this relationship, the frequency of a note is determined by the frequency of a known note value as well as the number of semitones that exist between the known note and the desired note. A commonly known note frequency is A_4, which has a frequency of 440 Hz. Therefore, in order to find the frequency of C_5, which is 3 semitones above A_4, the above relationship would be used where the value for "*known note frequency*" would be 440 and the value for "*semitones*" would be 3, producing a new note frequency of 523.25 Hz. However, in the case of this algorithm, the

relationship has to applied more generally in order to read any given pitch provided by the *audioread* function and map that to its new, desired pitch based on the "*semitones*" user entry for the transposition. Therefore, interpolation is used to map these existing frequencies to their new, transposed frequencies. Through this use of interpolation and the existing relationship between known note frequency and new note frequency, the resampling and frequency shifting of each individual frequency within the audio file can be accomplished. Once the upper and lower halves of the original FFT are each resampled and shifted, they are stitched back together again to form the new, transposed audio file FFT.

After the transposed FFT is constructed, the overall algorithm generates a plot of this newly transposed audio signal in its frequency domain. IFFT is then used to convert this new audio signal back into its time-domain form, where the absolute value of the new signal is plotted in its time-domain form. It is important to use the absolute value of this IFFT, as this eliminates the imaginary components of the IFFT. Then, using the absolute value of the transposed IFFT, the new, transposed audio file is played using the *sound()* MATLAB function at its original sampling frequency *fs*.

Results

To showcase the four plots that are produced by the transposition algorithm, an example audio file has been tested. In this example, the user has requested to transpose the wav audio file 'happy_birthday_guitar.wav' up by 2 semitones from its original key. The wav file is a monophonic audio file in which the song "Happy Birthday" is played on the electric guitar.

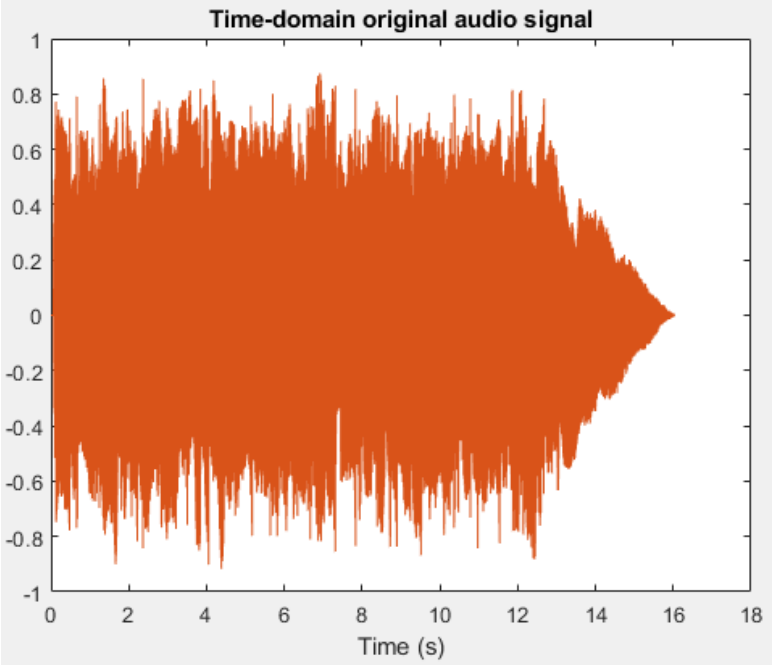

Figure 3. Audio signal in its time domain before transposition

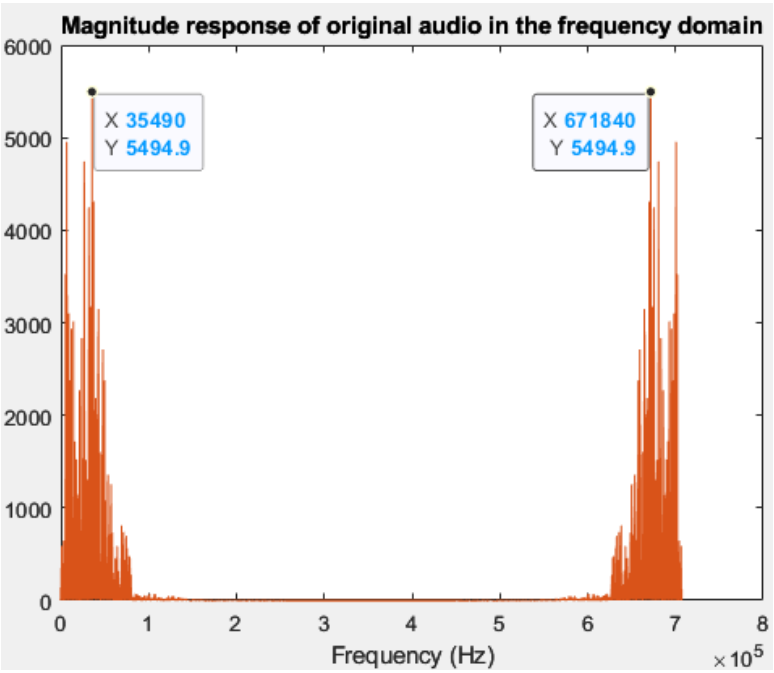

Figure 4. Magnitude response of audio signal in its frequency domain before transposition

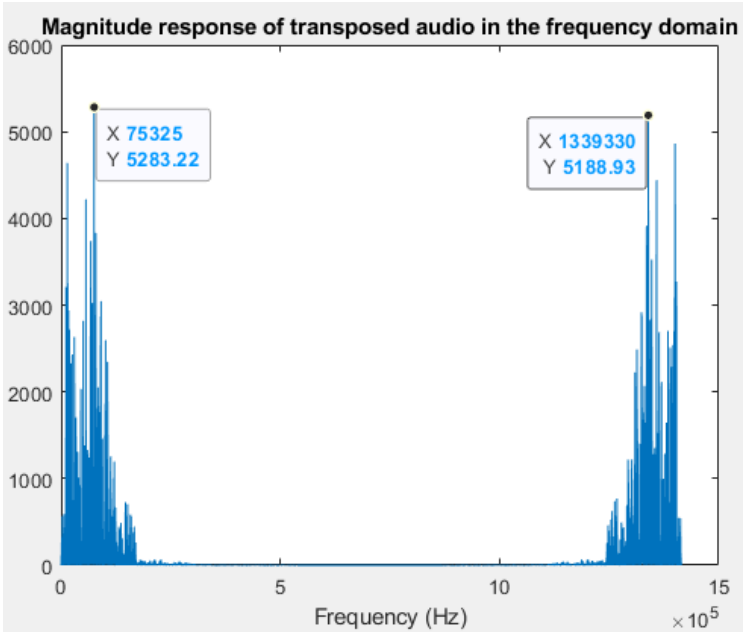

Figure 5. Magnitude response of audio signal in its frequency domain after transposition up by 2 semitones

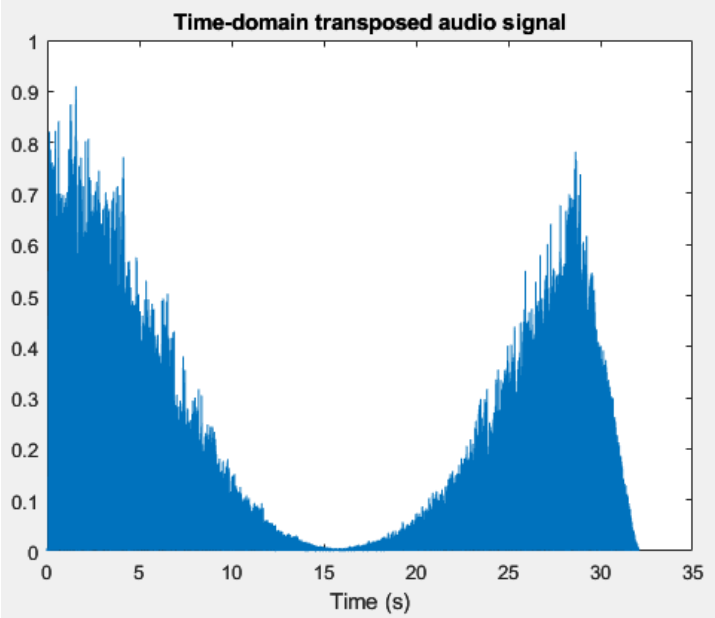

Figure 6. Absolute value of audio signal in its time domain after transposition up by 2 semitones

When comparing the peak magnitude values of Figure 3 and Figure 4, there is a clear shift in frequency as both the lower and upper halves of the FFT graphs have increased in frequency in correspondence with the requested +2 semitone transposition. Using this algorithm, the audio signal is duplicated and its tempo can speed up or slow down with dramatic changes in semitones. Such behaviors associated with filtering can be improved through the use of additional digital signal processing techniques. However, the algorithm presented produces an audio signal that has been transposed by a requested change in semitones, which satisfies the overall scope of this project.

References

Dao, Tri. “Butterflies Are All You Need: A Universal Building Block for Structured Linear Maps.” *Butterflies Are All You Need: A Universal Building Block for Structured Linear Maps · Stanford DAWN*, 13 June 2019, https://dawn.cs.stanford.edu/2019/06/13/butterfly/.

“Documentation.” *Documentation - MATLAB & Simulink*, https://www.mathworks.com/help/.

Doering, E. (1999). Making music with MATLAB: An electronic music synthesis course for engineering students. 1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No.99CH36258). https://doi.org/10.1109/icassp.1999.757617

Li, H., You, H., Fei, X., Yang, M., Chao, K.-M., & He, C. (2018). Automatic note recognition and generation of MDL and MML using FFT. 2018 IEEE 15th International Conference on e-Business Engineering (ICEBE). https://doi.org/10.1109/icebe.2018.00038

Monnier, N., Ghali, D., & Liu, S. X. (2021). FFT and machine learning application on major chord recognition. 2021 Twelfth International Conference on Ubiquitous and Future Networks (ICUFN). https://doi.org/10.1109/icufn49451.2021.9528762

“Piano Notes and Keys – How to Label the Piano Keyboard.” (n.d.). https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.pinterest.com%2Fpin%2F268738302754148787%2F&psig=AOvVaw2YB2vteap4lWlfh1-JKLNx&ust=1670014512182000&source=images&cd=vfe&ved=0CA8QjRxqFwoTCJj-rICn2fsCFQAAAAAdAAAAABAF

“The Scientist and Engineer's Guide to Digital Signal Processing” by Steven W. Smith, Ph.D. *How the FFT Works*, https://www.dspguide.com/ch12/2.htm.

“Tuning.” *Formula for Frequency Table*, https://pages.mtu.edu/~suits/NoteFreqCalcs.html.

Appendix A. MATLAB Code for Transposition Algorithm

```
% Asks for user input for file name and amount of transposition
filename=input('What wav audio file would you like to transpose? ');
semitones=input(['How many semitones do you want to adjust by?'...
    '\nFor example:'...
    '\nWrite +3 if you want to transpose up by 3 semitones'...
    '\nWrite -2 if you want to transpose down by 2 semitones '...
    '\nYour transposition? ']);

% Reads audio file from user input
[y,fs]=audioread(filename);

% Plot of original audio signal in time-domain
figure()
plot((1:length(y))/fs,y)
xlabel('Time (s)')
title('Time-domain original audio signal')

% Use FFT to convert audio signal to frequency domain
F1=fft(y);

% Plot of original audio signal in frequency domain
figure()
plot(abs(F1))
xlabel('Frequency (Hz)')
title('Magnitude response of original audio in the frequency domain')

% Frequency shifting
N=numel(F1);            % the # of elements in the fft
F1a=F1(1:N/2);          % the lower half of the fft
F1b=F1(end:-1:N/2+1);   % the upper half of the fft
t1=1:N/2;               % indices of the lower half
t2=1+(t1-1)/(1+2.^(semitones/12)); % frequency shift
F2a=interp1(t1,F1a,t2); % sampling of lower half
F2b=interp1(t1,F1b,t2); % sampling of upper half
F2=[F2a F2b(end:-1:1)]; % put lower half and upper half together again
```

```
% Plot of transposed audio signal in frequency domain
figure()
plot(abs(F2))
xlabel('Frequency (Hz)')
title('Magnitude response of transposed audio in the frequency
domain')

% Use IFFT to convert transposed audio signal back to time-domain
transposedSignal=ifft(F2);

% Plot of transposed audio signal in time domain
figure()
plot((1:length(transposedSignal))/fs,abs(transposedSignal))
xlabel('Time (s)')
title('Time-domain transposed audio signal')

% Play the new, transposed audio file
sound(abs(transposedSignal),fs)
```

Published with MATLAB® R2022b