**Harmonization Using A Phase Vocoder with Audio Effects**

Charles Huang, James Min, and Chris Woodard

ECE 485: Digital Audio Processing

Dr. Henry Pfister

**Introduction**

The inspiration for this project was taken from hardware and software harmonizers that take in audio recordings, typically recordings of music, and output harmonies of those audio recordings on top of the original track in real-time1. Effects such as delay, pitch correction, and distortion may be incorporated as well to mimic reverberation in a room of complex geometry or add instrumental timbre to one’s voice. These devices typically require users to select the key in which the music will be played, how many harmonies will be produced, and at what musical intervals above the melody. However, robust hardware and software harmonizers often cost hundreds of dollars or more, so the goal of our project was to create a rudimentary harmonizer with MATLAB for adding harmonies and effects to sound recordings using a phase vocoder.

**Background**

Several established digital audio processing techniques were utilized in the creation of the phase vocoder, including windowing and pitch shifting. Methods of pitch tracking were also analyzed and compared; different implementations included tools such as Fourier transformation and autocorrelation, two techniques for identifying a signal’s frequency content.

*Pitch Tracking*

Pitch tracking is completed by a pitch detection algorithm intended to estimate the pitch or fundamental frequency of an audio signal. For our project, pitch tracking allows for identification of unknown frequencies, which may be then shifted and resampled for harmonization using the phase vocoder. There are many different ways to accomplish pitch tracking, either in the time domain, as with autocorrelation, or in the frequency domain, as with Fourier transformation. Several algorithms take advantage of processing in both domains. Autocorrelation is the correlation of a signal with a time-shifted version of itself, and plotting this correlation as a function of the time shift allows for identification of frequency content. On the frequency side, a Fourier transform represents any signal as the sum of periodic sinusoidal signals, which may be plotted in order to determine primary signal frequencies. Throughout our project, we explored multiple options to fulfill this task and will discuss the differences below.

*Phase Vocoder*

A phase vocoder takes audio input files as input and performs pitch shifting as well as time compression and expansion. The following four major processes are completed in every phase vocoder implementation: windowing, Fourier transformation, pitch shifting, and inverse Fourier transformation2. Windowing is performed to break the audio signal of interest into small chunks for computational simplicity; a Hann window was used in each of our algorithms. For each window, a short-time Fourier transform is performed to determine its sinusoidal frequency. Next, pitch shifting occurs by mapping phase differences to a frequency interval, which corresponds to the magnitude of the shift. Resampling the shifted signal at a different rate balances the time expansion or compression accompanying the frequency shift, ensuring that the same amount of time elapses for the shifted signal as the original signal. Finally, the shifted frequency content is converted back to a time-domain signal using the inverse short-time Fourier transform.

**Materials**

*Pitch Tracking*

We came across a number of viable pitch detection algorithms in order to augment our project. The first one we looked at was named “Pitch Detection Algorithm” by Xuejing sun of Northwestern University3. This algorithm estimated pitch by spectrum shifting on the logarithmic frequency scale and then calculating the subharmonic-to-Harmonic ratio. The amplitude ratio between subharmonics with respect to harmonics reflects the deviation from the pitch of the original signal. This algorithm, however, came with many problems. It was created for the express purpose of calculating the pitch of speech instead of music. It also many times found two frequencies when there should only be one. These problems are evidenced in the following figure.

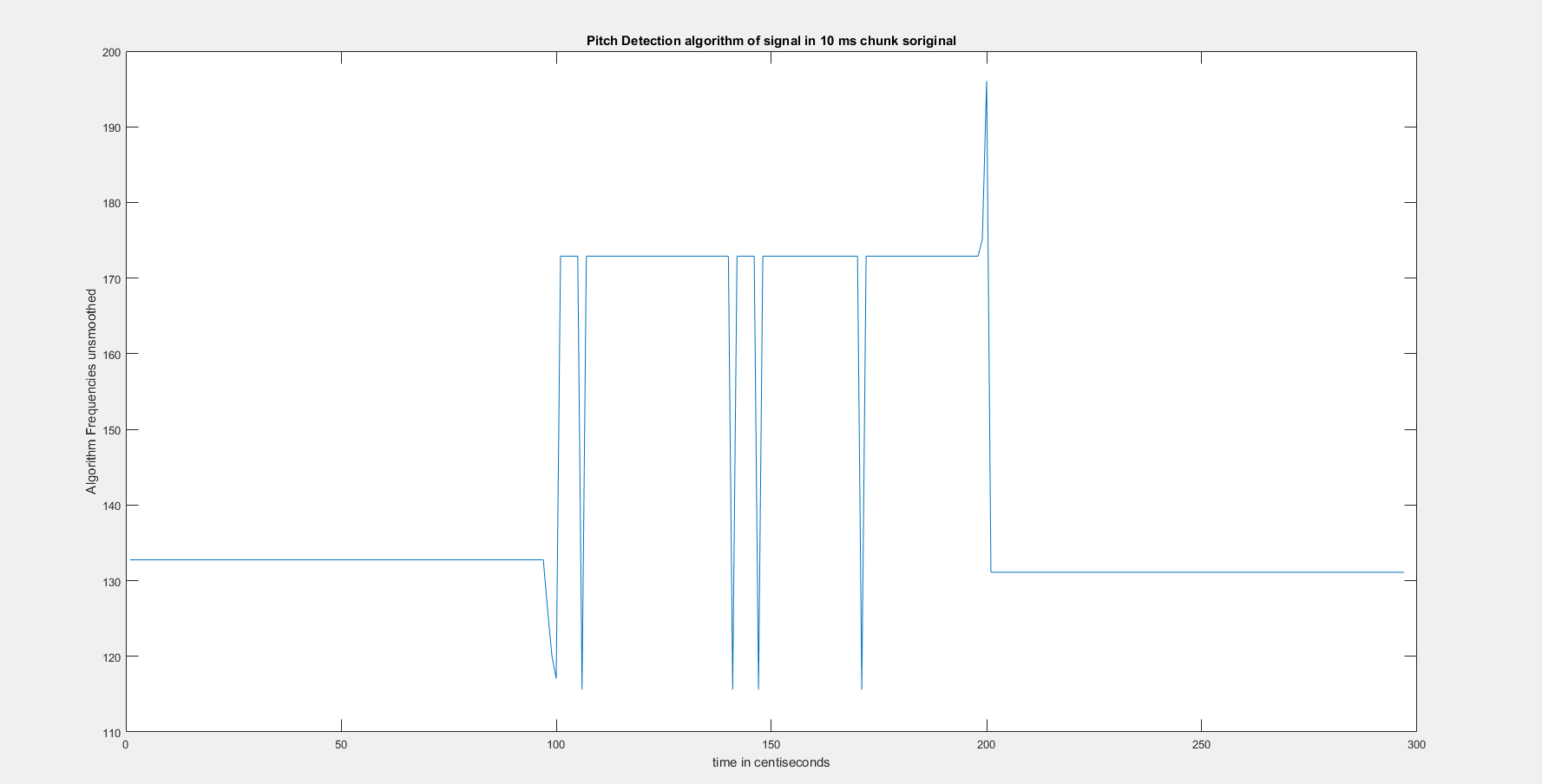


Figure 1: Pitch Detection fundamental frequencies

We also looked at an algorithm named YAAPT (“yet another algorithm for pitch tracking”) by Stephen A. Zahorian and Hongbing Hu of the Journal of the Acoustical Society of America4. This code, however, was also created for the purpose of pitch tracking high quality speech, so this algorithm was also incapable of determining certain frequencies throughout a song.

Next, we looked at an algorithm by Edgar Berdahl of Stanford University5. This code seemed promising and was written for the purpose of audio signals. Nonetheless, we ran into problems with the GUI and code documentation being entirely in German. After Google translating the entire GUI, we did get the code to run; however, we were unable to extract the frequency values into matrix form from the GUI.

Lastly, by augmenting the autocorrelation algorithm that Dr. Pfister uploaded to Sakai, we were able to partition audio signals into pieces and calculate each piece’s pitch via autocorrelation. This method worked out well and was our most promising algorithm. Our biggest problem with this algorithm was not the code itself, it was combining this code with the phase vocoder (discussed below) in order to correctly vocode each piece of the audio signal according to its pitch relative to the key of the signal.

*Phase Vocoder*

A phase vocoder takes audio input files as input and performs pitch shifting as well as time compression and expansion. Our phase vocoder algorithm consists of a top-level file (pvoc.m) and other smaller files needed for the analysis such as a short time Fourier Transform file (stft.m) and its inverse transform (istft.m). When running the code, an audio file is selected as input, as well as a default FFT size and a compression factor for each STFT. The input signal is first segmented using a Hann window function based on this factor; next, pvsample.m creates a new spectrogram to represent the amplitudes of all the frequencies. These estimates are made using phase differences between paired segments, and the corresponding frequency is calculated and incorporated into a new spectrogram. The new waveform is then created using istft.m to transform back to the time domain.

The phase vocoder we used for our project was an adaptation of pvoc.m by D.P.W. Ellis of Columbia University in 20026. In pitch shifting, first, the STFT is applied to window the signal into short segments. Windowing allows for smoother reconstruction since it normalizes high frequency artifacts of the input at the ends where segments overlap and estimates should not be taken from. Frequency estimates are then calculated based on a new phase-time compressed or expanded scale determined by the factor r. The phase vocoder maps the frequencies to known phase shifts in a certain time interval. Between two windows, the maximum frequency peak is identified and the closest frequency estimate based off the phase difference is determined. Since the frequency estimate is inversely proportional to the time difference, if the compression factor is less than 1 then the frequency estimate will be increased, thus shifting the frequency upward. Essentially the original signal has amplitude peaks in set frequency bins that are now all mapped to larger phase shifts. The signal is then reconstructed using the original time scale to get an output the same length as the input but with systematically higher or lower pitch added on top, creating a harmony effect.

Next, we incorporated various audio effects onto the melody and harmonies, similarly to a commercial harmonizer, such as flanging, echo, and reverberation. Flanging is an effect that, physically, makes the audio file sound more muffled, resembling a “robotic” voice. The effect is achieved by taking an original input signal and adding delayed versions to it so that it has a time-delay effect. However, unlike reverb where the delay is constant, flange.m utilizes a basic flanging function with a cosine to create a time dependent, oscillating delay so that the delay between the two added signals is different at every point. The result is a more randomized, muffled sound. This effect may be observed by running flange.m in MATLAB.

echofunc.m is the main function used to create the echo and reverb effects. The function adds a constant time-delayed version of the input at a slightly smaller amplitude to each point after a certain sample delay, created the effect of hearing the voice again after it is initially said. The reverb effect is achieved by modifying these parameters so that the delay is much smaller and the amplitude added is smaller as well. This creates the effect of a voice resonating immediately after but not repeating in a distinct manner, as reverb peaks are close together and do not resolve like echo peaks do. With more time, a more complicated standalone reverb effect could be implemented using convolution or other sophisticated methods.

Note:All code files are saved in a zip file uploaded to Sakai instead of copied onto an Appendix for this report. Speech\_Decoder.m is the main top-level script and draft.m and testHarmonize.m are the additional scripts run for the presentation.

**Discussion**

There were many difficulties we ran into while completing this project; each, however, taught us much about the complexities of implementing a phase vocoder using different methods.

After getting Speech\_Decoder.m, in which an audio signal was run through a phase vocoder, to work properly, we had thought our project was complete. Yet, we realized that we had much to learn about the musical aspects of a harmony. Given that major and minor chords in a key require a harmony at a different number of half-steps, we learned that we could not simply vocode an entire song and expect to hear pleasant harmonies. Thus, we began to look for better ways to pitch track entire signals and then vocode each piece of a signal according to whether the pitch was major or minor, given a particular key. This task was also riddled with many complications, whether it was the accuracy of the pitch detection algorithms discussed above or combining pitch detection and phase vocoding into a single process.

In the end, we probably learned as much about music and harmonies as we did about pitch tracking and phase-time analysis of frequency spectrograms. The vocoder demonstrates a method of frequency estimation and analysis that other methods may not have the same success or resolution with, due to the use of phase differences as opposed to frequency bins as the main variable. This highlights the idea that digital audio processing is often influenced by perception in addition to algorithms; we must also understand the fundamentals of the system we are trying to augment and enhance.

**References**

[1] Good description of harmonizers:

<http://tweakheadz.com/harmonizers-and-more/>

[2] Mathematical explanation of phase vocoder:

<http://sethares.engr.wisc.edu/vocoders/phasevocoder.html>

[3] Pitch Detection Algorithm:

<http://www.mathworks.com/matlabcentral/fileexchange/1230-pitch-determination-algorithm>

[4] YAAPT:

<http://ws2.binghamton.edu/zahorian/yaapt.htm#ref1>

[5] Pitch Detection of Musical Signals:

<https://ccrma.stanford.edu/~eberdahl/Projects/Grundfrequenzanalyse/>

[6] Open source phase vocoder program:

<http://www.ee.columbia.edu/ln/labrosa/matlab/pvoc/>