PITCH CORRECTION TOOL IN MATLAB

Benjamin Shafran, Joshua Miller, Brent Ikei

University of Rochester

ABSTRACT

For our final project, we were able to create a functional pitch correction or "auto-tune" software programmed in MATLAB. The three main components of this project are pitch detection (determining the pitch which is being played or sung), the pitch shifting algorithm used to change the pitch of audio without changing the speed at which it is played, and finally pitch correction: the "tuning" aspect of the auto-tune.

1. INTRODUCTION

Our goal for this project was to accurately receive audio, pitch shift, and pitch correct the audio file as stated above. One reason why we were interested in this research project is because we all have previous experience mixing music and designing filters using various programming languages however, we have not developed programs regarding pitch correction. We also wanted the project to be practical and able to be used after its completion. Auto-tune is often used in pop and hip-hop music, but consumer plugins are typically very expensive [1]. With our finished product, we planned to be able to achieve the same effect used in a wide variety of popular music today without purchasing expensive audio plugins. To do this, our intention was to have a sufficient user interface, where the user of the program could input 2-3 arguments of their liking and effectively use it in our code. To accurately determine the pitches of the audio sample, we used the YIN algorithm to split the audio into multiple frames and determine which fundamental frequency was present. To pitch shift and correct the audio, we used the PSOLA algorithm to extract the pitch periods of the audio and accurately remap them to increase or decrease pitch frame by frame. See "YIN" and "PSOLA" for more details.

2. OBJECTIVES

Our goal for this project was to make a pitch correction software using the YIN and Pitch Synchronous Overlap-Add (PSOLA) algorithms. The YIN algorithm was used to detect pitches. From there, the detected pitches were sent to the PSOLA function to do pitch correction. We set out to have minimal distortion in our output signal and an effect that could be easily applied to any signal. Additionally, we wanted to have flexible methods for choosing how pitches are corrected: either by a defined major or minor scale, or by a drawn in pitch curve. Although we did not aim for our software to be real time based, we also attempted to use algorithms that would let us achieve our goal of pitch correction as quickly as possible.

3. PITCH DETECTION

3.1. Frequency Domain Based Approach

One method for pitch detection which we initially began trying to implement was a frequency domain based approach which utilized the Harmonic Product Spectrum as a method of estimating pitch. By marking strong peaks above a given threshold in the frequency domain, one can guess the fundamental frequency and see how many of its multiples are also present in the signal. Whichever guess of the fundamental frequency has the highest number of harmonic multiples present in the signal is presumed to be the actual fundamental frequency. Although this method is simpler in theory, compared to the steps needed for the YIN algorithm, it does not yield as favorable results. To improve the speed and also accuracy of our program, we instead went with a time-based pitch detection method rather than this frequency based method, which would require taking

the Fourier transform of the input signal for every frame.

3.2 Time Domain Based Approach (YIN)

The YIN algorithm is essentially a modified version of the autocorrelation function used in the time domain. Using the difference function on frames of the input signal, YIN is able to accurately use the frames calculated to determine outlier dips which correspond to periods of the signal. After using the cumulative mean normalized difference function with absolute thresholding to find minima in the function (see Figure 3.2.1) and improve overall accuracy, parabolic interpolation is used to find the exact location of these dips. The dips that are calculated correspond to pitch periods of the input signal [2] and from each period, the estimated pitch can very easily be found on a frame by frame basis.

For our implementation of the YIN algorithm (adapted from Github User: orchidas' code [3]), we used a window size of three times the period of the lowest possible pitch we were trying to detect (65 Hz), a threshold of 0.1 for finding the dips in the cumulative mean normalized difference function, and an amplitude threshold for determining silence (0.05 for laptop webcam microphones) so that silent frames would not create problems for our pitch detection output. Additionally, we implemented a low pass filter at 1kHz before passing the input signal into the YIN function to limit the upper bound of pitches, so that it would properly detect frequencies within vocal range and further improve accuracy. These modifications to the YIN algorithm were obtained via suggestions of its author, Alan de Cheveigné, but also through experimental testing. The main changes consisted mostly of the window size and silence threshold, to determine which values for different parameters of the algorithm would yield the best results. The results of our YIN algorithm's pitch detection on a sung C major scale can be seen in Figure 3.2.2. As expected, the frequencies detected follow the frequencies of a C major scale starting from C3.

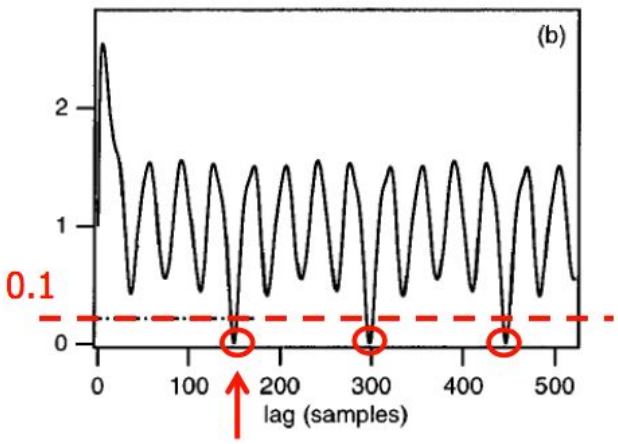

Figure 3.2.1: Detected dips using the cumulative mean normalized difference function [2]

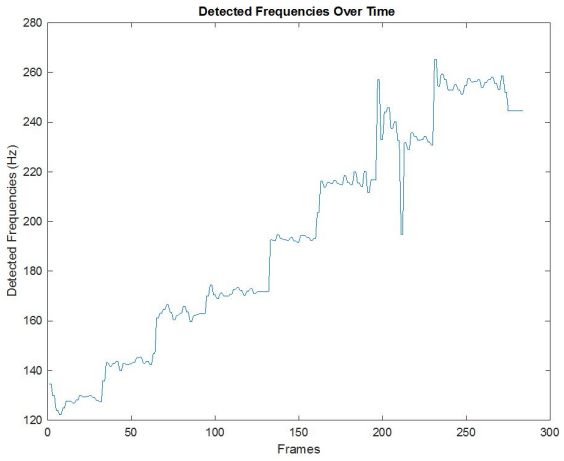

Figure 3.2.2: Detected fundamental frequencies of a sung C major Scale Using our implementation of YIN

4. PITCH SHIFTING

4.1. Phase Vocoding

At the onset of the project we tried to implement phase vocoding (a pitch shifting method which utilizes upsampling and downsampling) to achieve the pitch shifting portion of the software. After getting the algorithm to a point where it could repitch audio files without retaining the original file length, we noticed the algorithm produced a lot of noticeable artifacts. After some research we determined that a phase vocoding algorithm would not be a feasible option for our needs, especially considering our focus on pitch correcting the human voice.

4.2. Pitch Synchronous Overlap Add (PSOLA)

Pitch Synchronous Overlap Add (PSOLA) is the algorithm we utilized for pitch shifting and correction. Optimized for the human voice, this time domain method preserves the length of the original signal while shifting pitch on a frame by frame basis. PSOLA achieves pitch shifting by first marking and windowing pitch periods within a frame and then remapping them to contain either more or less periods within the same time range, making the pitch higher or lower [4]. This process can be seen in Figure 4.2. For this pitch period remapping we utilized MATLAB's k-nearest neighbor function. This algorithm provides very high quality pitch shifting within an octave range at relatively efficient speed. Our implementation of the PSOLA algorithm still has some minor artifacting, but it is a lot less abhorrent than the artifacting present with the phase vocoding algorithm.

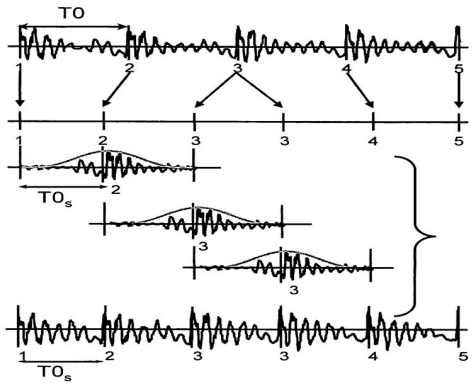

Figure 4.2: Example of Pitch Periods and their remapping in PSOLA Algorithm [4]

5. IMPLEMENTATION

Our program operates using a main script and separate functions for YIN and PSOLA, all of which are written entirely in MATLAB. Currently our program takes in a 10-second audio input from a microphone. Using YIN to detect the frequencies of the audio given, the user was able to either draw pitches that they would want to shift to on the graph of the frequencies given, or pick a predefined scale, e.g. G minor. The pitch curve drawing feature can be seen in Figure 5.1. Although this option is is not as precise as a scale, it allows for more gestural and coarse pitch correction such as a large upward slope of pitch over time. The alternative method: correcting pitches to a user-selected predefined scale can be seen in Figure 5.2, where the user inputs the scale they would like to correct their pitches to. This figure also shows the rest of the console output for a typical run of our program. The output of the YIN function: the detected pitches are used in conjunction with either the closest note in the scale, or the value of the line drawn for that frame to get a ratio of pitch desired to pitch detected. This ratio is sent to the PSOLA algorithm where through pitch period modification and signal reconstruction via overlap-add, the signal is pitch shifted to the desired pitch. After applying the PSOLA algorithm for pitch correction, the main function then outputs the pitch corrected signal for the user to hear using MATLAB's soundsc function.

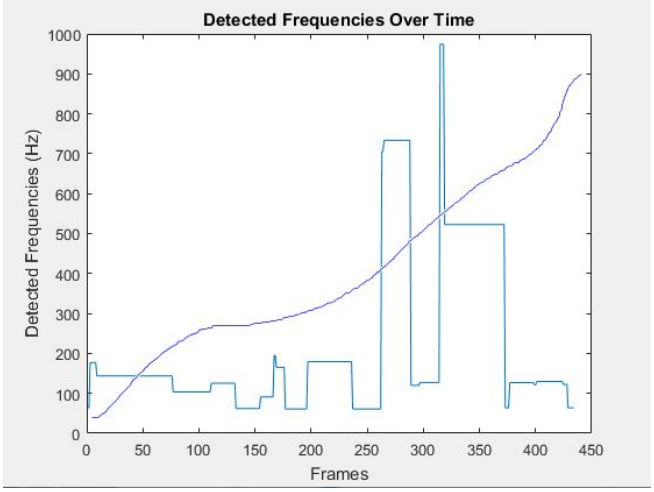

Figure 5.1: Example of user inputted pitch curve

```
>> main
Would you like to draw your pitch correction curve? (y/n): n
Please enter the key to retune to (E, G♯, Bb etc): Bb
Please enter the key tonality (major or minor): minor
Start singing.
End of Recording.
Estimating pitches sung...
Correcting Pitch...
Listen to your voice auto-tuned!
>>
```

Figure 5.2: Example of console output of a typical program run

6. CONCLUSIONS

6.1. Achievements

Our program is able to successfully pitch correct a given monophonic input signal. We were able to accomplish constructing a user interface that is able to ask if the user wants to shift frequencies to a predefined scale or if they want to draw the new pitches in. After that, the program is able to apply these pitch changes with the YIN and PSOLA algorithms and is able to output the newly corrected audio file for the user to listen to within a matter of about 20 seconds. Even though we used time trying to implement the phase vocoder instead of PSOLA, we managed to be able to still get a pitch correction program in a timely fashion. We were able to follow our planned schedule for the completion of this project and was able to manage our time efficiently.

6.2. Future Work

For future work, we would want to improve upon both algorithms used. For YIN, this would involve more effective thresholding for silence detection. For PSOLA, this would involve implementing epoch marking, which would create a more natural sounding output. Both algorithms could benefit from further experimentation with window size and filtering techniques. Additionally, there are some unexplained clicks in our autotune output which we would like to correct, either by way of filtering or improving either of our algorithms. Lastly, for verifying our entire process in a more concrete fashion, we would test our pitch detection algorithm with datasets where the pitch is definitely known, such as in the case of a laryngograph recording.

We would also like to implement some parameters of the pitch correction that the user could modify, e.g. attack time, pitch glide, or a simple dry / wet signal mix. We would want to be able to apply the YIN and PSOLA algorithms for longer audio files, without taking too much processing power or an excessive amount of time. We would like to be able to optimize our program so that it would be optimal for various instruments and not exclusively for vocals. Ideally, we would want a pitch correction program that is polyphonic and able to pitch correct specific instruments, without affecting other instruments within the audio file.

Regarding the user interface, we would like to incorporate a graphical user interface (GUI) into our program so that the user would have an easier time manipulating variables of their choosing in real time, such as the new pitch that they want to shift to. They would also be able to choose how long they would want to record their voice, instead of the predetermined 10 seconds.

7. REFERENCES

[1] Cnx.org. (2012). *Auto-Tune*. [Online] Available at: https://cnx.org/exports/22567958-1f9b-4426-8abe-b9e0736df034@1.1.pdf/auto-tune-1.1.pdf [Accessed 29 Apr. 2018].

[2] Alan de Cheveigné, "YIN, a fundamental frequency estimator for speech and music" *Acoustical Society of America*, vol. 111, no. 4, April, 2002. [Online serial]. Available: http://audition.ens.fr/adc/pdf/2002_JASA_YIN.pdf [Accessed 30 Apr. 2018].

[3] Orchidas, *yin_estimator.m* [Online], Github 2017, Available: https://github.com/orchidas/Pitch-Tracking/blob/master/yin_estimator.m [Accessed 30 Apr. 2018].

[4] Ricardo Gutierrez-Osuna, "L19: Prosodic modification of speech" *research.cs.tamu.edu*, [Online]. Available: http://research.cs.tamu.edu/prism/lectures/sp/l19.pdf [Accessed 30 Apr. 2018].