Autotune Signal Analysis: Anyone Can Sing

Joseph Wihbey
Mech. Engineering Department
Columbia University
New York, New York
jgw2132@columbia.edu

Alex Thornton
Elec. Engineering Department
Columbia University
New York, New York
apt2141@columbia.edu

***Abstract*—Autotune, the digital signal processing technique developed in the late 1990s, significantly impacted the musical recording world with its ability to correct the pitch of audio recordings and enhance them with unique sound effects. Prior to this patented algorithm being released, there were numerous methods proposed to pitch shift audio signals, but none were able to do so without introducing some level of distortion. The purpose of this paper is to document the implementation of one of these more rudimentary pitch shifting algorithms in audio processing fields dominated by the modern Autotune algorithm. The two fields studied are pitch detection and correction and audio augmentation. Our algorithm is implemented via a GUI "slider" that allows the user to specify the desired musical note the audio sample should be shifted to. In so doing, the algorithm can correct any deviations from the intended pitch of the audio or it can drastically shift the pitch of the audio to simulate the vibrato effect that the Autotune algorithm can create.**

Keywords—Autotune, pitch shift, audio, pitch detection

I. Introduction

Following the release of Cher's Believe in 1998, the music industry was revolutionized with the advent of Autotune - a proprietary digital signal processing technique that corrects the pitch in a vocal recording to the idealized note the singer is trying to produce. From reviewing current uses of Autotune, there are two prominent applications in modern music. One is the heavily enhanced "robot" sound made famous by T-Pain, and the other is a more minimal pitch correction used by numerous performers to adjust their recording to match a desired note.

First off, what is pitch? Sound, most notably music, is characterized by a frequency related scale that attributes notes as sounding "higher" or "lower". The pitch is an aspect of this spectrum. Each note in the musical scale has a fundamental frequency, which is closely related to, but does not necessarily define the pitch of the note. For the purposes of this paper, the fundamental frequencies of musical notes are considered to be the pitch shift targets, as this is a more quantifiable, and thus manipulable, variable.

Modern pitch correction techniques have devolved into two primary categories. One involves using repeatable features of a signal to define a period, and thus measure a signal's frequency, which can then be transformed to a desired pitch. The other involves utilizing all of the sampled signal data through autocorrelation.

The proprietary Autotune algorithm that revolutionized the music industry corrects intonation errors of vocals in real time without distorting the output via autocorrelation. Traditionally requiring a high level of computation to generate, the algorithm utilizes mathematical tricks to make the autocorrelation computation more efficient in real time [2]. By using autocorrelation, the entire signal is analyzed at once, a robust technique for identifying the pitch at different time intervals.

In comparison, alternative pitch detection and shifting methods rely on the tracking of certain attributes of the signal to determine the instantaneous pitch of the signal. In [1], the pitch of a signal is identified by detecting when a signal amplitude rises above a certain threshold value after having crossed a prescribed lower threshold. By comparing the time when these two events happen, the fundamental frequency of the periodic signal can be measured. Similarly, in [3], pitch is measured by documenting zero crossings. In this case, the threshold is specifically when the signal goes from a positive amplitude to a negative amplitude. This period can be used to calculate a frequency. [3] then proposes a pitch shifting technique by extending or shrinking these periods, thereby changing the pitch.

The pitch identification and shifting techniques in [3] were the foundation for this paper's autotune algorithm because of the simplicity of using zero crossings to identify pitch and then windowing these segments to establish a new pitch. This technique relied on processes taught in a graduate level digital signal processing course, making it an accessible, yet still efficient algorithm choice. Using [3] we derived our own unique autotune algorithm and created an application to allow a user to interact with an audiofile of their choice.

II. Methods and Materials

A. Frequency Modulation Attempt

An early and primitive attempt at performing pitch correction involved determining the difference between the fundamental frequency and the desired frequency, and modulating by this frequency difference. Intuitively, the idea was to then low pass filter the lower portion of the spectrum leaving the pitch-shifted copy. This idea, however, was doomed to fail. Due to the relatively small delta between fundamental and desired frequencies to the total signal bandwidth, filtering out the lower and upper modulated copies is virtually impossible. The function BadTune.m explores this method through a simple example with a pure sinusoid, and the results are displayed in Figure 1.

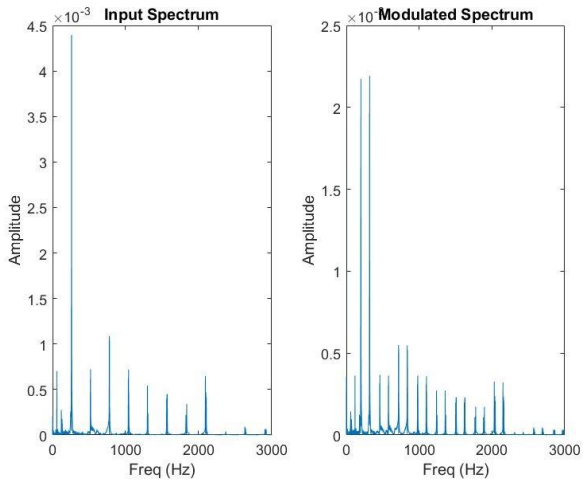

Figure 1: Crude autotune by frequency modulation

Pitch correction by the method presented in BadTune.m results in audio that has a strange oscillatory component to it and as a result the technique was quickly abandoned. A more refined approach is described in the following section.

B. *Overview*

The algorithm used is composed of a pitch tracking component and a pitch-shifting component. It is packaged within a matlab GUI that is launched through startup.m. Upon starting, the GUI allows the user to select an audio file and a corresponding note from the musical scale to tune to. With these two options selected, the algorithm can be run. See Figure 2 for an image of the GUI.

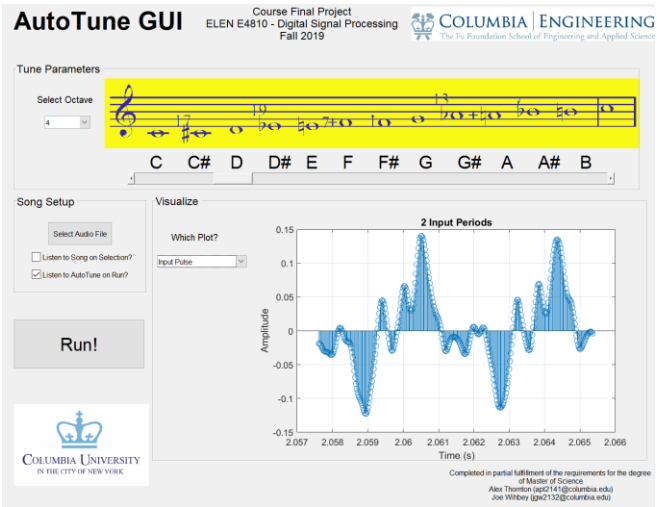

Figure 2: Snapshot of GUI

The algorithm is implemented through the function autotune_gui.m. The function inputs are the audio sample, a 1xN vector, the desired frequency to shift to (Hz), and the sampling frequency of the input signal (Hz). The function outputs the tuned input signal as well as the band passed input signal used for zero crossing prediction.

The two core components of the algorithm, pitch detection and pitch shifting, are described below.

C. *Pitch Detection*

As mentioned earlier, this algorithm relies on zero crossings to measure the pitch of a segment of the signal. The pitch detection stage of the algorithm determines the dominant pitch of the signal by finding the maximum value of the magnitude of the normalized fast Fourier transform. The length of the fast Fourier transform is rounded to the next nearest power of two to improve the algorithm's efficiency. This value is used to band pass filter the input signal at the dominant pitch frequency, as shown in Figure 3.

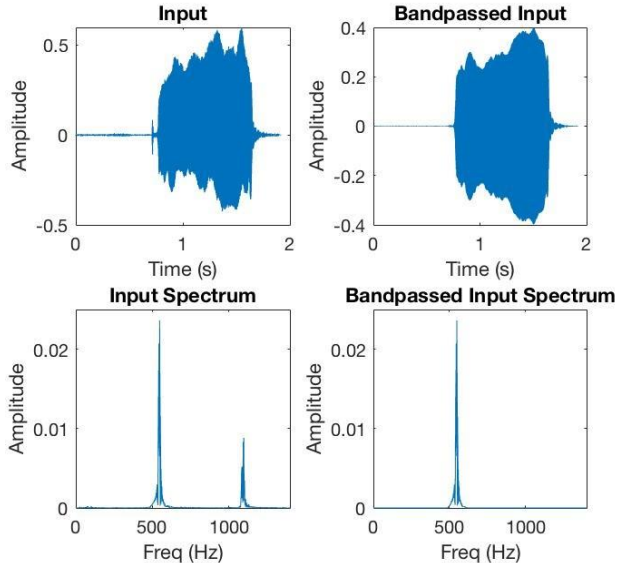

Figure 3: Sung C_5 (523.3 Hz) input file and band passed input. From inspecting the plots, the singer was singing 20 Hz too high.

The band passed input identifies the dominant fundamental frequency of the audio signal. This signal is then passed into a zero tracking loop, which increments through each point in the signal until the amplitude changes from positive to negative. The time duration between zero crossings is the period of the signal, the inverse of which is the signal's fundamental frequency. These sequences between zero crossings are transformed and inserted into the output audio file in the pitch shifting stage of the algorithm.

D. Pitch Shifting

In the pitch shifting stage, the fundamental frequency of the segment between zero crossings is scaled to a desired frequency by windowing the segment and placing it into an output array. The original, un-band passed filtered signal is windowed and inserted, rather than the band pass filtered signal used to identify zero crossings because the unattenuated signal of the

original audio needs to be shifted in order to preserve the characteristics of the original sound. Per [Lent Paper] a hamming window the length of the input segment is used to smooth out the edges of the window and minimize Gibbs effect errors. These windowed segments are inserted into the output array at the output array index. The output array index is incremented forward by the period of the desired output frequency and the windowed input array is filled in from this point. As shown in Figure 4, when the desired frequency is greater than the input frequency, the windowed segments overlap. When the desired frequency is less than the input frequency, the windowed segments spread out from one another, effectively zero padding the output.

By dividing an audio file with multiple notes into short chunks, the algorithm could be used to pitch shift multiple

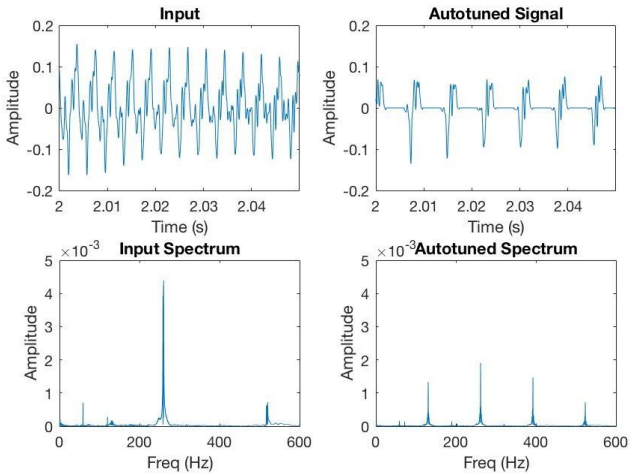

Figure 4: Piano C_4 (261.6 Hz) shifted to C_3 (130.8 Hz) to exhibit zero padding for low frequency shift.

segments of an audio file to different frequencies. The user could use the maximum value of each segment's frequency spectrum to identify a target pitch and then shift it to either the closest note or a desired frequency mapping input.

III. Data and Results

Four different types of signals were used to test the algorithm: a computer generated sine wave, a piano note, a recorded sung, note, and a sequence of notes. The different results are seen below.

A. Window Shifting Perfect Sine Waves

A base objective of the algorithm was to demonstrate pitch shifting on a simple sine wave. Figures 5 & 6 show the results of pitch shifting a computer generated sine wave input of 3000 Hz. Note that in these instances there is clear variation between upshifting and downshifting. Upshifting results in a single frequency shift, with no corresponding harmonics. In contrast, downshifting resulted in two dominant frequencies: a frequency at 2000 Hz (the desired output) and at 4000 Hz, its first harmonic. To compensate for this, a band pass filter was applied to the output signal at the desired frequency for these simple sinusoidal inputs.

B. Window Shifting Recorded Audio

As demonstrated in Figure 4 and Figure 7, another goal was

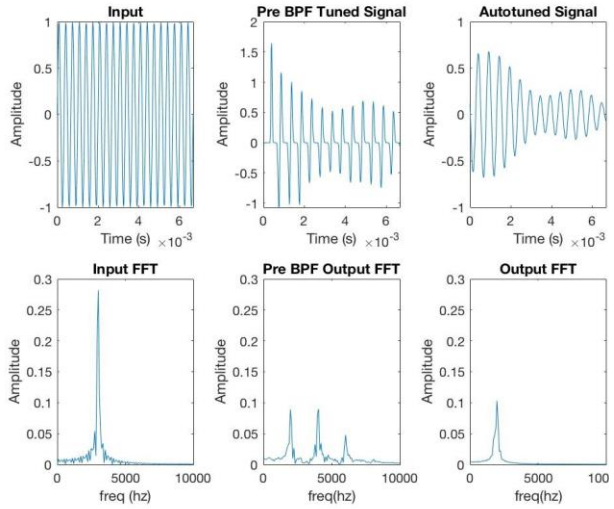

Figure 5: 3000 Hz input sine wave shifted to 2000 Hz

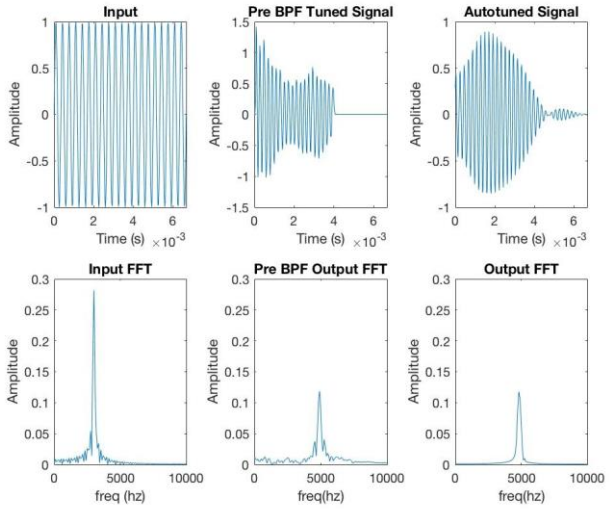

Figure 6: 3000 Hz input sine wave shifted to 4000 Hz

to implement the algorithm on recorded audio. Specifically, piano and sung notes were tested. At first, a band pass filter was applied to the shifted outputs, similar to the process used the sinusoids. However, the output signal sounded flat when compared to the original. Real signals have a much more complex frequency spectrum, which contributes to the overall uniqueness and sound quality of an individual's voice or the characteristics of a piano. As a result, the shifted output of real signals was not band pass filtered, in order to preserve its individualized qualities. Thus, the final algorithm, used for pitch shifting recorded audio, only uses a band pass filter to highlight zero crossings, and pitch shifts the raw input signal.

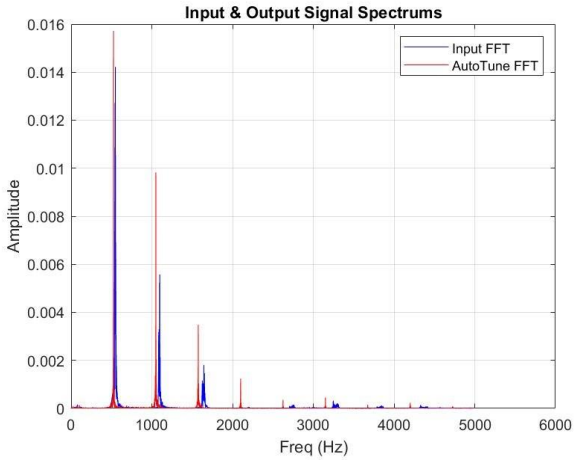

Figure 7: Sung C_5 (523.3 Hz) input overlaid with pitch corrected output

C. *Stretch Goal*

The final objective was to implement a 'karaoke' version of our pitch correction algorithm. Essentially, this program would accept a 'truth' audio file and an input audio file, and pitch correct the input file to match the truth file fundamental frequency at every point in time. Aligning the truth and input files proved to be more challenging than expected. As a compromise, we pursued partitioning the input into a finite number of 'chunks', running the algorithm to correct the fundamental frequency to the nearest note for each chunk, and then stringing the output chunks together. This method effectively replicates traditional autotune methods, but due to the slow computation time of MATLAB, this is more of a proof of concept than a finished product. Included in the final project code is a file, autotune.m, which performs this algorithm, but is not included in the GUI because of its impracticality.

IV. Discussion

In conclusion, the algorithm was able to accurately pitch shift audio signals containing one dominant pitch up or down one octave with minimal frequency distortion. In the case of the sung_highC.mp3 audio file (see Figure 7), the algorithm was able to correct the frequency of the sung note by the required 20 Hz. From reviewing the frequency spectra, it appears that the algorithm worked perfectly. However, the shift did introduce a subtle robotic quality to the pitch shifted output, more so than desired when compared to the autotune product on the market. Possible causes for this robotic quality could come from the rudimentary nature of measuring frequency from zero crossings. This calculation only uses a small subset of the data, rather than looking at the entirety of the signal. As a result, the algorithm could wash out the more complex nuances of a recorded voice or musical note. As suggested by [2] a more sophisticated method for pitch identification would be to use autocorrelation, which would examine the signal as a whole, rather than individual chunks. This would also increase the speed of the algorithm, as it would be analyzing the entire signal in real time, processing each sample as it was measured.

A further cause of audio distortion could be from the amplitude attenuation of the output signal, as observed in Figures 4, 5, & 6. The amplitudes of the output files are approximately one third of the input. The hamming window used to construct the output file could cause this attenuation. The hamming window is necessary to smooth out the edges of the overlapped segments in the output file, at the cost of signal amplitude. Future work could include testing different windows to optimize smoothing versus amplitude attenuation ratio.

Another unexplained aspect of the algorithm was the difference in upshifting and downshifting of the frequency. This was best seen in the case of frequency shifting a simple sinusoid (Figures 5 & 6). There are clear harmonics present in the down shifted output, whereas none are present for the up shifted output. The likely explanation for the harmonics are the empty space introduced between pulses when converting to a lower frequency. A future design could attempt to mitigate this effect, or research exactly why it is an inherent feature of this design.

An alternative design could include revisiting our first attempt to autotune via modulation. It might be possible to implement a technique which modulates the signal to a higher intermediate frequency, lowpass filtering the lower band, and then modulating down to the desired frequency. However, this process could potentially introduce undesirable audible effects. Nevertheless, it might be an interesting possibility to investigate.

Overall, our design is effective as an autotune tool. We were able to implement a simple yet efficient algorithm for our own application, and designed a useful GUI, which makes interacting with any audio signal easy and exciting. Since both of us were interested in understanding and exploring how autotune works, this has been a passion project for us, and an expansive learning opportunity. We look forward to further work on the project.

Acknowledgment

Thank you Mariam Avagyan and Professor John Wright for guiding our project and inspiring us!

References

[1] D. Slepian, E. Weldon, “Determination of pitch,” U.S. Patent 4 217 808A, July 18, 1977.

[2] H. Hildebrand, “Pitch detection and intonation correction apparatus and method,” U.S. Patent 5 973 252A, November 26, 1999.

[3] K. Lent, “An Efficient Method for Pitch Shifting Digitally Sampled Sounds,” *Computer Music Journal*, vol. 13, No. 4, pp. 65-71, Winter, 1989.