Phase Vocoder Implementation with FLWT and TD-PSOLA

Terry Kong
Stanford University
Email: tckong@stanford.edu

***Abstract*—The following is a report written for EE 264's final project. The goal of the project was to realize a real-time implementation of a phase vocoder on a TI TMS320C5535 DSP chip. The theory behind the implementation can be divided into two sections: the Fast Lifting Wavelet Transform (FLWT) used for pitch detection, and the Time-Domain Pitch Synchronous Overlap and Add (TD-PSOLA) method for pitch correction. An implementation of the algorithm is provided that is optimized for processors that lack of a floating-point arithmetic unit. The libraries are available at https://github.com/terrykong/Phase-Vocoder.**

I. Introduction

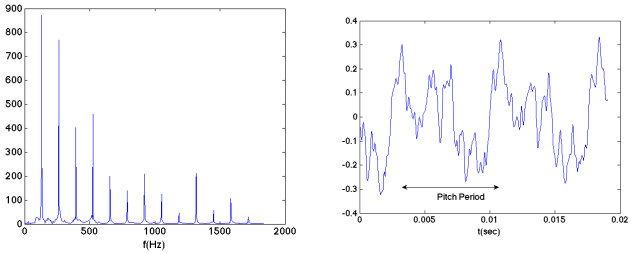
In the area of speech and audio processing, it is often desired to modify the properties of the original audio signal. These modifications include pitch shifting, time expansion and compression, and formant preservation. The focus of this project is to implement pitch shifting. Pitch is a property of sound that characterizes the perceptual ordering of frequencies. A higher pitch corresponds to a higher frequency and a lower pitch corresponds to a lower frequency. Pitch is usually determined by the fundamental frequency of a signal, which may not be the case in a signal with significant overtones or higher harmonics. It will be assumed for this project that the pitch will always be characterized by its fundamental frequency.

The process of pitch shifting, depending on how well it needs to be done, is usually an involved task. To correctly shift the pitch of a segment of audio, the pitch must first be identified. This motivates the need for a robust pitch detection algorithm. Once the pitch has been identified, the pitch can be shifted which may be followed by a few touch-up algorithms that abate the adverse effects pitch shifting introduces.

II. Pitch Detection

When analyzing the pitch of an audio signal, there is usually a choice in the flavor of the algorithm. The analysis can take place in the frequency domain, usually via the Short-Time Fourier Transform, or in the time-domain. The benefit of frequency domain analysis is that it can be interpreted graphically very easily. For example, Fig.1a shows the STFT of an audio sample of middle C on a piano. Notice the pitch can be easily extracted by looking at the peaks. We would call the peak at 130Hz the pitch of this signal.[1]

[1]The actual absolute frequency that middle C takes on is 261.626Hz. This is a prime example of the perceived pitch being disguised by overtones.



(a) Frequency Domain (b) Time-Domain

Fig. 1: Different Representations of Middle C from a piano

Pitch, in the time-domain, appears as periodicity. In Fig.1b, the pitch period is marked and it also delineates the difficulty in extracting the pitch in time-domain. Typical methods of extracting the pitch include the autocorrelation method or the average magnitude difference function.

There are many differentiators between time-domain approaches and frequency domain approaches, but the one that takes precedence in this project is latency. With the advent of the Fast Fourier Transform (FFT), it has become possible to take STFTs extremely fast even on CPUs with a single core. With the advantage of having an intuitive representation and robust FFT implementations, the frequency-domain methods seem to be a logical choice. However, it turns out that there is another algorithm that can compete with the efficiency of the STFT, namely the Fast Lifting Wavelet Transform.

A. FLWT

The FLWT is like the FFT in that it takes advantage of structure to speed up the calculation of the Discrete Wavelet Transform(DWT). The DWT can be implemented with a filter bank as depicted in Fig.2.

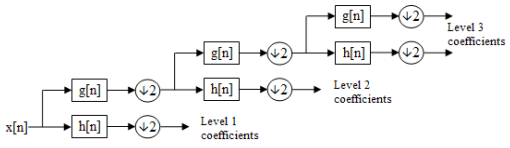


Fig. 2: Wavelet Filter Bank

The DWT shares many properties of the Discrete Fourier Transform. One property is that both transform the time domain signal into a different signal space with a different

basis. The basis that this algorithm uses is the Haar Wavelet (we refer to this particular Wavelet Transform as the Haar Wavelet Transform). This basis yields an interpretation of the filters that compose the filter bank.[2] At each stage of the filter bank, the signal is being both low pass filtered and high pass filtered simultaneously by each branch. The coefficients of each of the low pass and high pass filters are $\left(\frac{1}{2}, \frac{1}{2}\right)$ and $\left(-\frac{1}{2}, \frac{1}{2}\right)$, respectively. [3]

What we exploit from the FLWT is the low-pass branch at each stage of the filter bank. The idea is simple and intuitive: *low-pass filter the signal until it is smooth enough to detect the pitch period by peak tracking.* One way to determine the pitch is by finding the distance between the two highest peaks in a window. This can be made more reliable by taking the most common distance between any two peaks in a window, i.e., the mode.[4] Since the highest and lowest peaks hold information that is important for formant preservation, it is important to remove the local maxima and minima, which usually form due to high harmonics and noise. To remove these local peaks, we smooth (low-pass filter) the signal. This is the motivation behind taking the DWT of the signal. Since only the low-pass branch of each stage is useful, the high-pass filter branch does not need to be evaluated. This vastly simplifies the complexity of the algorithm.

Here is the structure of the pitch detection algorithm using the FLWT:

Algorithm 1 FLWT (Modified)

1: **for** $n = 1 : \frac{windowlength}{2}$ **do**
2: $\quad newwindow[n] = \frac{window[2n+1]+window[2n]}{2}$

Algorithm 2 Pitch Detection

Ensure: $windowlength = 2^k, k \in \mathbb{N}$
1: $levels \leftarrow$ Num of stages in DWT
Ensure: $\frac{length\ of\ signal}{2^{levels}} \gg 1$
2: **for** $i = 1 : levels$ **do**
3: $\quad window \leftarrow FLWT(window)$
4: Find the absolute value of the differences between all peaks
5: $pitch = mode($*peak differences*$)$

One thing to note is that if $levels$ is too large given the range of pitches you are trying to detect, you may filter out the frequencies you are trying to detect. Appropriate settings are covered in the *Results* section.

This approach of using the FLWT for pitch detection is inspired [2].

[2]Another interpretation of the Haar Wavelet Transform is that each stage is taking a weighted average and weighted difference of the input at that stage.

[3]The correct Haar wavelet coefficients are $\left(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}\right)$ and $\left(-\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}\right)$. However, we do not need to recover the signal, so removing the constant weighting of $\sqrt{2}$ is preferred since it reduces the number of computations.

[4]The event that occurs at the highest frequency.

B. Improving FLWT's Robustness

A proper implementation of this pitch detection algorithm should deal with the edge case where a segment of audio is pitch-less. It may even be possible that the signal does not have one dominant frequency, as is the case for white noise. If a threshold parameter is used to improve the confidence of correct pitch estimations, then the number of incorrect pitch estimations will also increase. For this reason, this project also explored different ways to improve the reliability of the pitch detection algorithm by augmenting different processes after the main algorithm. The two possibilities that were explored were:

1) Median Filtering: Using a median filter, rare anomalous pitch estimations can be removed. For example, instead of detecting the fundamental frequency, the pitch detection algorithm may detect an integer multiple of that fundamental frequency.[5] The median filter is actually a non-linear filter, so if the median filter is not low-order, it can become the bottleneck of the entire pitch-detection algorithm.

2) Zero-Order Hold: The zero-order hold process is simply the idea of returning the last reliable pitch estimation if the current pitch estimation is not reliable.

The operation of these augmented processes are included in the *Results* section.

III. Pitch Correction

Like pitch detection algorithms, pitch correction algorithms also come in two flavors: time-domain methods and frequency domain methods. Frequency domain pitch correction algorithms are usually very robust, but the caveat is they suffer from very long latencies.[6] For this reason, the obvious choice is to use a time-domain implementation. The method chosen for this project is the Time-Domain Pitch Synchronous Overlap and Add method(TD-PSOLA). The benefit of this algorithm is that it is simple and one of the fastest implementation for pitch correction.

A. TD-PSOLA

The idea behind TD-PSOLA is to identify pitch periods, the reciprocal of the pitch frequency, within the time-domain signal, and shift copies of the pitch period around to change the fundamental frequency. Shifting copies closer together increases the fundamental frequency while spreading them farther apart decreases the fundamental frequency. The process effectively resamples the signal in the frequency domain. This is an astounding effect because this couples a simple time-domain operation to a very complicated one in the frequency domain. The algorithm is illustrated in Fig.3.

Here is another helpful interpretation of the TD-PSOLA method. If there is only one fundamental frequency and the harmonics of an audio signal are prominent, as in Fig.1a,

[5]Sometimes known as an octave error

[6]This is attributed to frequency domain interpolation and the need to calculate trigonometric functions.

[7]Image from: https://gtms1324.wordpress.com/2013/02/20/speech-synthesis-using-psola/

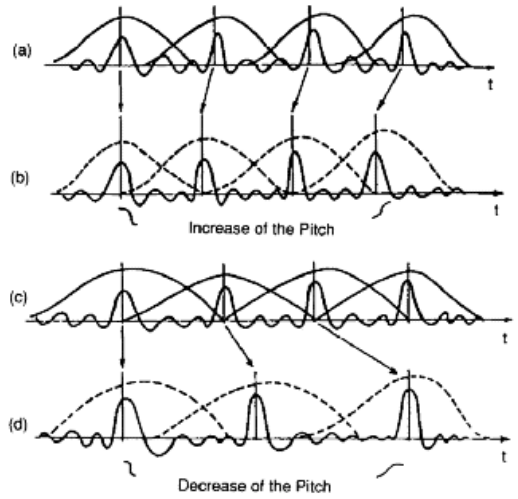


Fig. 3: TD-PSOLA Illustration[7]

then the spacing between each peak in the frequency domain corresponds to a particular pitch period. These peaks can be thought of as a delta train that is sampling the spectral envelope. The envelope holds the formant information, while the spacing of the delta train that samples the envelope holds the pitch information. According to Poisson's summation formula, a delta train in the frequency domain is related to a delta train in the time domain (with different spacing). This explains why moving copies of the pitch period changes the pitch. Because if we are decreasing the spacing between adjacent pitch periods, we are decreasing the spacing of the delta train in the time domain. Since there is an inverse relationship between time and frequency, the decrease in the time-domain delta train spacing corresponds to an increase in the frequency domain delta train spacing (which is equivalent to increasing the pitch).[8]

An important subtlety of the TD-PSOLA algorithm is the weighting window that needs to be applied to the analysis window. For this project, the bartlett window was used since its coefficients can be calculated very quickly. This weighting window effectively weights the importance of the values in the middle of the analysis window. The result minimizes the effect of the the phase inconsistencies when shifting and adding since the edge of each analysis window will be zero.

Here is an the structure of the pitch correction algorithm using TD-PSOLA:

Note that for a given window length and sampling frequency, there is a fundamental limit to how much the pitch can deviate. While this paper doesn't discus how to calculate this limit, a rule of thumb that seems to provide a reasonable effect is that there must be at least four pitch periods within one window length.

Algorithm 3 Pitch Correction

1: $T_s \leftarrow \frac{1}{sampling\ rate}$
2: $T_{analysis} \leftarrow \frac{1}{pitch\ period}$
3: $T_{synthesis} \leftarrow \frac{1}{new\ pitch\ period}$
4: $N_{analysis} \leftarrow \frac{T_{analysis}}{T_s}$ (analysis spacing)
5: $N_{synthesis} \leftarrow \frac{T_{synthesis}}{T_s}$ (synthesis spacing)
6: initialize $window_s$ (synthesis window)
7: $i = N_{analysis}/2$
8: $j = N_{analysis}/2$
9: **for** each pitch period in the analysis window **do**
10: $window_s[j - N_{analysis}$ to $j + N_{analysis}]$ += $window_a[i - N_{analysis}$ to $i + N_{analysis}] * w_{bartlett}$[9]
11: $i+ = N_{analysis}$
12: $j+ = N_{synthesis}$

B. Differences between typical TD-PSOLA Implementations

The implementation of the TD-PSOLA differs from the typical TD-PSOLA method that is usually encountered. Most of the time, TD-PSOLA methods will have a pitch marking preprocessing step that looks at the time-domain signal and marks prominent peaks, and uses the distance from each adjacent peak as the local pitch period. For this project, this approach doesn't make much sense since there is already a pitch detection step that estimates the fundamental frequency across the entire window. Since the pitch period is uniform over the window, the implementation of this specific TD-PSOLA is much faster than the garden variety implementations since the pitch markings are known at the start of the pitch correction algorithm. The caveat is this approach doesn't guarantee formant preservation because there is no guarantee that each pitch period in the analysis window will be centered at a prominent peak.

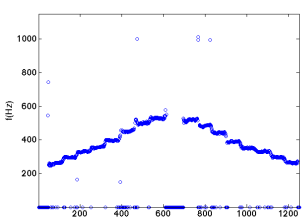
Another caveat to this implementation is imposed by the TI TMS320C5535 DSP chip. The C5535 is limited to 512 samples per window at its fastest sampling rate: 48kHz. This implementation imposes an interested constraint on the pitch correction problem. Since only a very small portion of the signal is being analyzed at a time, phase inconsistencies manifest at the edge of each window, which becomes a nontrivial issue. The crux of the issue is that the TD-PSOLA method attempts to leave the spectral envelope unchanged, but to do so introduces changes in the phase of the signal. Hence, TD-PSOLA method does not guarantee the continuity in phase between different windows. Since the operation on the signal is local, a discontinuity in the phase manifests at the edge of each analysis window and between adjacent windows. This issue is perceived as a humming in the synthesis.

IV. Results

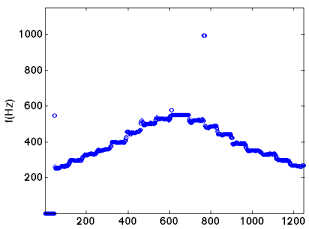
All algorithms previously discussed were implemented for real-time operation. The figures that follow are all generated by sending data back to MATLAB to display.

[8] A similar argument can be made for decreasing the pitch.

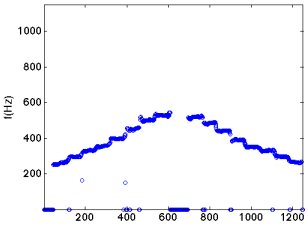
[9] $window_a$ is the analysis window, which is the original window, and $window_s$ is the synthesis window, which is the window where the result of the TD-PSOLA is formed. This is the common definition of synthesis and analysis in the literature.



(a) No Additional Process



(b) Zero-Order Hold



(c) 5th Order Median Filter

Fig. 4: Performance of Pitch Detection Algorithm

The settings that the chip used to produce reasonable audio with negligible latency were: a sampling rate of 32kHz, an analysis window size of 512 words[10], and a 5th order median filter. The FLWT used 6 levels.

A. Median Filter vs. Zero-Order Hold

The median filter resulted in the best performance. Figure 4 is a collection of results using a recording of an individual singing a C-scale. Notice the improvement that using either a zero-order hold, Fig.4b, or a 5th order median filter, Fig.4c, has compared to the unprocessed result, Fig.4a.

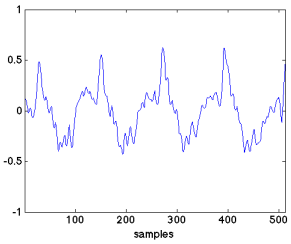
This performance improvement can be qualified in two different ways. Firstly, the median filter removes most of the octave errors which if fed to the pitch correction algorithm, will change the frequency by a significant erroneous amount. Secondly, for periods when no one is speaking or singing, the zero-order hold method will assume that even silence has a pitch, which may be an issue for signals that have non-negligible energy, e.g., white noise.

Although a median filter is non-linear, the implementation of the 5th order median filter is done by using the nine comparisons, the minimum amount, instead of sorting each block of five data.

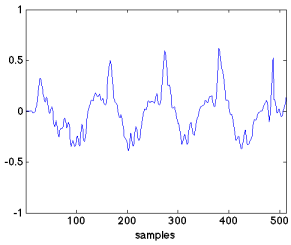
B. Pitch Detection Tradeoffs

The pitch detection algorithm was very effective above the range of 32kHz. The algorithm usually pinpointed the pitch within a few hertz. The algorithm began to operate unreliably with high amounts of error when the sampling rate was below 24kHz. This makes sense since the sampling frequency is directly related to the granularity of frequency.

[10]int16 or two bytes

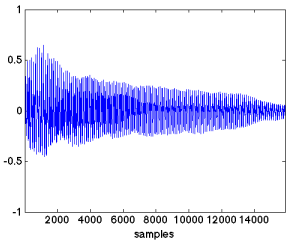


(a) Before TD-PSOLA

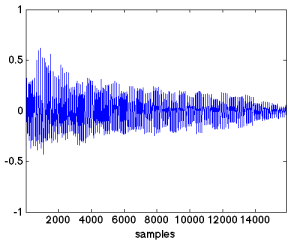


(b) After TD-PSOLA

Fig. 5: Local Effect of TD-PSOLA: Middle C being shifted three semitones[11]higher



(a) Before TD-PSOLA



(b) After TD-PSOLA

Fig. 6: Global Effect of TD-PSOLA: Middle C being shifted three semitones higher

C. Pitch Correction

The local effect of the pitch correction algorithm can be seen in Fig.5. Notice the tapering at the edge of the synthesis window after the TD-PSOLA algorithm has been applied. This is the effect of applying a weighting window. The signal after 512 samples is also at a different amplitude after applying the TD-PSOLA. This is the phase inconsistency that was discussed earlier that results in a humming when played back.

The global effect of the TD-PSOLA algorithm can be seen in Fig.6

Interestingly, this implementation of the TD-PSOLA did not always work for all sampling frequencies and all window sizes. For example, the TD-PSOLA would work very poorly if the sampling rate was too high. Without exhaustive testing, my hypothesis is that the phase inconsistencies become more significant as the window size represents a smaller and smaller time period. However, the sampling frequency cannot be lowered too much because the pitch correction may not have enough copies of the pitch period to adequately change the pitch. This is a very important tradeoff to keep in mind since it will largely effect performance.

The ideal choice of sampling frequency for the TD-PSOLA algorithm was a low frequency, while the pitch detection algorithm required a higher frequency. This tradeoff is interesting to note since it is a common theme in signal processing to trade performance and speed.

[11]An change of k semitones is equivalent to multiplying the frequency by $2^{k/12}$. A positive k is pitch increase, and a negative k is a pitch decrease

V. Conclusion

Implementing a robust real-time phase vocoder is a very difficult task. As this project has demonstrated, it is possible to create a real-time phase vocoder, but making its effect natural and perceptually pleasing requires more time than a short project allows. During the process of writing the software, I realized that certain parts of the pitch detection algorithm could be made more sophisticated given more time. I encourage anyone that finds this interesting to visit my github, and download the software and try it yourself. The libraries are not dependent on any system specific library or operating system.

VI. Future Work

There is a lot more to do to make this implementation more robust, which is exciting because this algorithm can actually become better. It was written with best code practice in mind so it shouldn't be hard to continue working on this, or for someone else to pick it up.

Here is a list of the things that could be done to improve the functionality assuming this will be run on the TI TMS320C5535 chip:

- Include buffered memory to hold past windows to provide a smoother transition between adjacent windows. This will minimize the effect of the phase discontinuities.
- Write a library that dynamically adjusts the pitch to the "best" frequency, as opposed to its current operation, which is to adjust the pitch to a set piano key scale.
- Include a efficient fixed-point coefficient generator for an arbitrary length Hann window, which will replace the fixed-point coefficient generator for an arbitrary length bartlett window.
- Implement a time-expansion and compression mode, which is simply the TD-PSOLA repurposed.

Acknowledgment

I'd like to thank Professor Ronald Schaffer and Professor Fernando Mujica for their extremely supportive guidance throughout this project. Thanks to them this project became more than just a small dysfunctional MATLAB program. I'd also like to thank the TAs Reggie Wilcox and Maisy Wieman for their support and instruction for proper documentation of the software.

References

[1] E. Larson, R. Maddox, Real-time Time Domain Pitch Tracking Using Wavelets, *In Proceedings of University of Illinois at Urbana Champaign Research Experience for Undergraduates Program*, 2005.

[2] Lawrence Rabiner and Ronald Schafer. 2010. *Theory and Applications of Digital Speech Processing (1st ed.)*. Prentice Hall Press, Upper Saddle River, NJ, USA. Chapter 13.5

Appendix

Project Timeline

1) Week 1 (2/16-2/22): Implemented Pitch Detection in MATLAB

2) Week 2 (2/23-3/01): Implemented Pitch Detection in C++

3) Week 3 (3/02-3/08): Implemented Pitch Correction in MATLAB

4) Week 4 (3/09-3/13): Implemented Pitch Correction in C++ and Wrote Supporting Libraries for Final Demo