Automatic Pitch Detection and Shifting of Musical Tones in Real Time

By Jinho Kim

A&S Undergraduate Honors Thesis 2013

Advised By Professor Sergio Alvarez

Computer Science Department, Boston College

Abstract

Musical notes are acoustic stimuli with specific properties that trigger a psychological perception of pitch. Pitch is directly associated with the fundamental frequency of a sound wave, which is typically the lowest frequency of a periodic waveform. Shifting the perceived pitch of a sound wave is most easily done by changing the playback speed, but this method warps some of the characteristics and changes the time scale. This thesis aims to accurately shift the pitch of musical notes while preserving its other characteristics, and it implements this in real time on an Android device. There are various methods of detecting and shifting pitch, but in the interests of simplicity, accuracy, and speed, a three step process is used. First, the fundamental pitch of a stable periodic section of the signal is found using the Yin pitch detection algorithm. Secondly, pitch marks that represent the local peak of energy are found, each spaced out by roughly one period (inverse of the fundamental frequency). Lastly, these marks are used in the Pitch Synchronous Overlap and Add (PSOLA) algorithm to generate a new signal with the desired fundamental frequency and similar acoustical characteristics to the original signal.

Table of Contents

1. Introduction

1.1 A Brief History of Pitch Shifting

Pitch shifting, the technique of altering the original pitch of a sound, was famously used in the 1950's to create the high pitched voices of Alvin and the Chipmunks. Voice actors would record themselves speaking or singing at half speed at half the normal tape recording rate, and when the tape was played back at the normal speed, the pitch increased by an octave due to the doubled playback rate. This comedic high pitched effect became known as the 'chipmunk effect' and is still used today. Less extreme versions were also used in other productions, such as in the Wizard of Oz (1939) to raise the pitch of the munchkins' voices and to lower the pitch of the witch, and it was also used to create the voices of Daffy Duck and Tweety Bird[1]. This technique had its limitations, however, as it also changes the time length of the signal.

This limitation was overcome by Fairbanks in 1954 with the creation of a modified tape recorder, which had several rotating playback heads. The difference in relative speed of these heads compared to the absolute playback speed of the tape was able to produce continuous pitch shifting of up to +10% and -40%[2] .

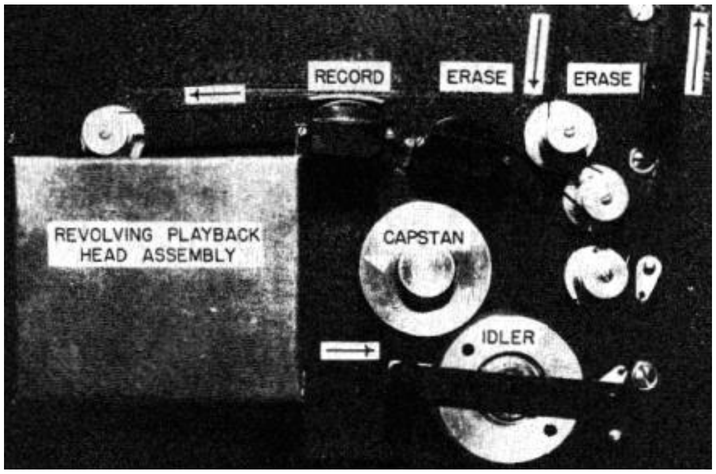


Figure 1 – Fairbanks' multiple revolving heads device[2]

As the capacity of memory improved, digital devices became available to become powerful tools in sound processing. The Eventide H910 was released in 1975 as the first commercial harmonizer[3]. This was the first device that could digitally shift pitch whilst preserving time length, and it was instantly put to use in recording studios. For example, this device was first used to downward shift a sped up version of the sitcom *I Love Lucy* in order to fit in more commercials.

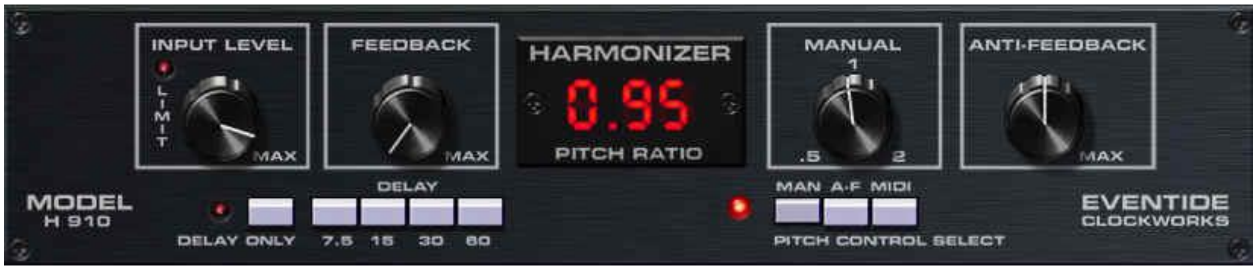


Figure 2 – The front of the Eventide H910, the first commercial harmonizer[3]

Pitch shifting is now a commonly used sound effect that is included in most digital sound processing libraries. As pitch shifting algorithms and their implementations have improved, this effect is now achievable in real time, and it has seen widespread use in the music industry. For example, the popular effect ‘autotune’ is a pitch corrective technique that converts a normal voice into a robotic sounding one by pitch shifting the original signal to a perfect pitch, as well as artificially altering some of the vocal characteristics.

1.2 Main Goals

The purpose of this thesis was to successfully study and implement pitch shifting while meeting specific constraints:

1) Preservation of the time scale – the shifted playback audio should have the same duration as the recorded audio.
2) Preservation of the original acoustical characteristics – the shifted audio should sound the same as the original audio except for the pitch shift.
3) Processing in real time – the processing rate should be faster than the recording rate.

The implementation was done in Android for demonstrative purposes, and a variety of algorithms were explored (see section 2) to find the an appropriate method in order to achieve the above mentioned constraints.

1.3 Main Challenges

As there are so many algorithms available, it was difficult to choose which ones to use for this project. After much research, I decided to pitch shift with PSOLA (section 2.5), pitch detect with YIN (section 2.5.1), and pitch mark with the two-phase algorithm (section 2.5.2).

I first implemented PSOLA using MATLAB for proof of concept. Once pitch shifting was achieved and the sound quality was deemed acceptable, I shifted my focus towards Android. However, I had no prior experience with Android development, so programming an Android application from scratch proved to be quite a challenge. Also, creating an efficient class hierarchy and optimizing the application speed was fairly challenging (more in section 3).

1.4 Basics of Sound

The manipulation of sound first requires an understanding of it. In the most basic sense, sound waves are vibrations that propagate through a physical medium, the most common of which is air[5]. Vibrations cause a rapid alternation between compression and expansion of the medium, which is propagated outwards from the source in a wavelike fashion by the transference of energy from molecule to molecule. These vibrations are generally sinuisoidal and thus periodic in nature.

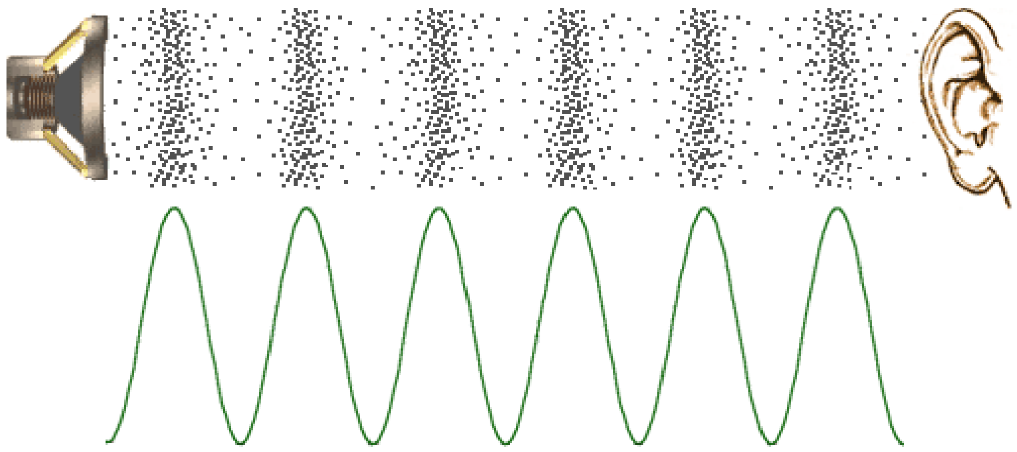


Figure 3 – A graphical representation of the propagation of a sound wave[4]

1.4.1 Amplitude – Loudness of a Sound

The oscillating pressure caused by sound waves can be measured and represented in the time domain, with the x axis denoting time and the y axis representing the change in pressure. As shown above in **Figure 3**, the compressions correspond to the high points (peaks), while expansions correspond to the low points (valleys). This pressure variation is called the amplitude and is typically measured in decibils (dB). The magnitude of a sound wave's amplitude is represents the intensity of the vibration (pressure variations) which determines the loudness of a sound.

1.4.2 Frequency and Period – Pitch of a Sound

The rate at which the vibrations oscillate is known as the frequency and is typically measured in Hertz (Hz), which is the number of cycles completed per second. The inverse of frequency is the period (wavelength), which is the length of time it takes to finish one cycle, as shown in **Figure 4** below.

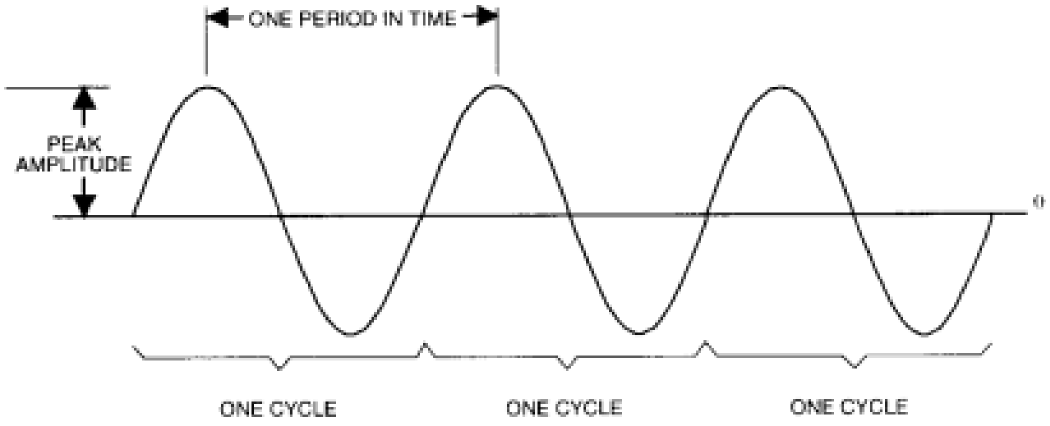


Figure 4 – Labeled terminology of a sound wave[5]

Pitch, which is the 'highness' or 'lowness' of a musical note, is directly related to the frequency of a sound wave, so a high pitched squeak of a mouse would be characterized by rapid vibrations in the air, while a low pitched call of a whale would be characterized by slower vibrations in water. Different animals have varying ranges of sensitivity to pitch. For example, humans' ears are able to detect sounds from a frequency range of roughly 64Hz to 23kHz, while dogs can hear from a frequency range of 67Hz to 45kHz – precisely why dog whistles cannot be heard by humans[6].

1.4.3 Harmonics and Complex Waves – Tone Quality of a Sound

The previously shown sound waves were perfect sine waves for simplicity's sake. A perfect sine wave is a pure tone of a single frequency and sound uncharacteristically dull[5]. Most musical tones are complex and have multiple sine waves of different frequencies in them. The lowest frequency in the tone is known as the *fundamental frequency*, which is the most important component because it determines the pitch of the overall sound. Musical tones typically have additional frequencies embedded in the signal that are integral multiples of the fundamental frequency. These multiples are known as harmonics, and the presence and strength of these harmonics determine the tone quality (timbre) of a sound. Different instruments and organisms have varying patterns of harmonics and thus produce a diverse array of sounds even when playing a note of the same pitch. As shown in **Figure** 5 below, the fundamental frequency and its accompanying harmonics are algebraically summed together to produce a complex waveform. Although the fundamental frequency is usually the dominant signal in a sound, sometimes it may be overriden by another harmonic or may be missing completely[7].

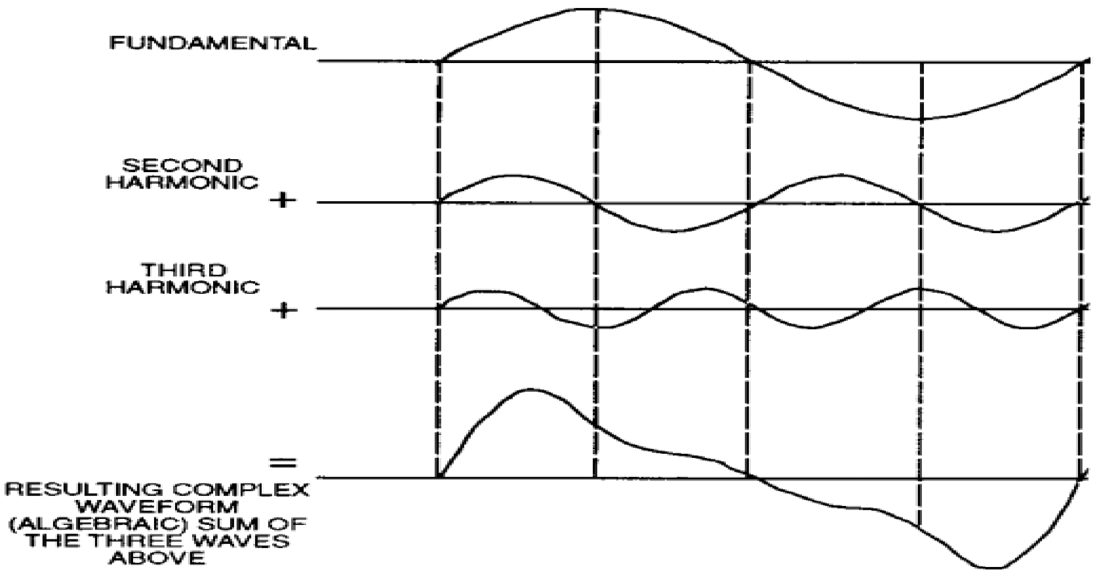

FUNDAMENTAL
SECOND HARMONIC
+
THIRD HARMONIC
+
=
RESULTING COMPLEX WAVEFORM (ALGEBRAIC) SUM OF THE THREE WAVES ABOVE

Figure 5 – The summation of the fundamental frequency and its harmonics to create a complex waveform[5]

2. Pitch Shifting Algorithms

2.1 Overview

The Phase Vocoder was one of the earliest methods for pitch shifting with time preservation, and it was proposed in 1966 by Flanagan and Golden[8]. This algorithm was first implemented 10 years later by Portnoff in 1976 using the Fast Fourier Transform[9]. Digital Sound Processing has exploded in growth since then, and a number of pitch shifting algorithms have been invented and improved.

2.2 Time Domain versus Frequency Domain

Pitch shifting algorithms are typically separated into two groups – those that operate in time domain, and those that operate in the frequency domain. Sound processing in the time domain is done using the familiar representation of the change of amplitude over time. The frequency domain is a different representation, where the x axis is frequency and the y axis is the amplitude. This representation can show the strengths of the individual frequency components present in the signal, which would be difficult to see or detect in the time domain. **Figure 6** shown below depicts a single complex signal in the time domain. This signal is composed of the union of 2 different frequencies, which are easily separated in the frequency domain, as shown by the 2 distinct lines in the right graph[10].

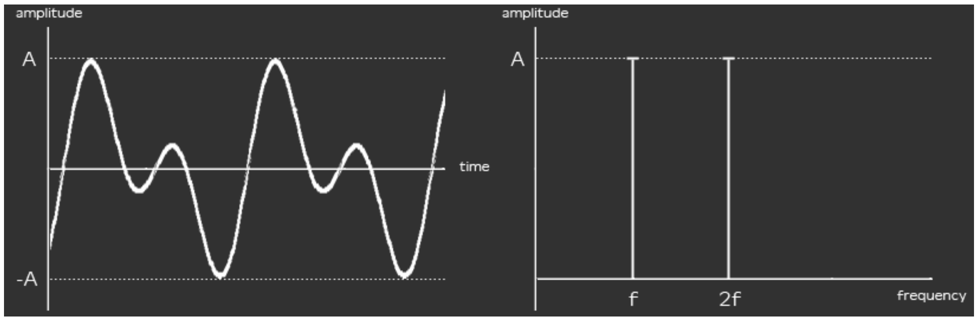


Figure 6 – Time Domain on the left, Frequency Domain on the right[10]

Pitch shifting algorithms utilize the different information available in the time and frequency domains and have varying degrees of performance. The general pros and cons of each are summarized below in **Table 1**.

	Pros	**Cons**
Time Domain	-Fast -Easier to implement	-Performs poorly with large shift factors -Unable to process polyphonic signals well -Significantly less accurate in the presence of noise
Frequency Domain	-Works well with large shift factors -Able to process polyphonic signals well -Resistant to noise	-Computationally expensive -Difficult to implement properly (requires post-processing to preserve original acoustical characteristics)

Table 1 – Pros and Cons of Pitch Shifting in the Time and Frequency Domains

2.3 Pitch Shifting in the Frequency Domain

Sound manipulation in the frequency domain is usually done by breaking up the signal into small overlapping segments (windows) and then processing each segment separately. This is necessary because the frequency of a signal can change over time, which would result in a large number of detected frequencies if the whole signal were to be converted into the frequency domain at once. By processing each small windowed segment separately, we can isolate the frequencies present in that period of time to achieve greater accuracy.

In order to process a windowed segment in the frequency domain, the window must first be converted from the time domain. Typically, this is done with the Fast Fourier Transform (FFT). The FFT outputs an array of complex numbers that can be converted into magnitudes that represent the average power of certain frequency ranges. These frequency ranges are called bins, and the number of bins determines the frequency resolution of the FFT given by the following equation[11]:

Resolution (Hz) = (sampling rate) / (window size)

Higher resolution can be achieved with a larger window, but the downsides are slower processing and the increased likelihood of instability in frequency.

The resulting spectrum of the windowed segment then undergoes spectral manipulation to change the frequencies within the signal, which is done differently by various pitch shifting algorithms.

Lastly, the processed windowed segments are then combined and resynthesized to recreate the pitch shifted signal. These steps are visually outlined in **Figure 7** below.

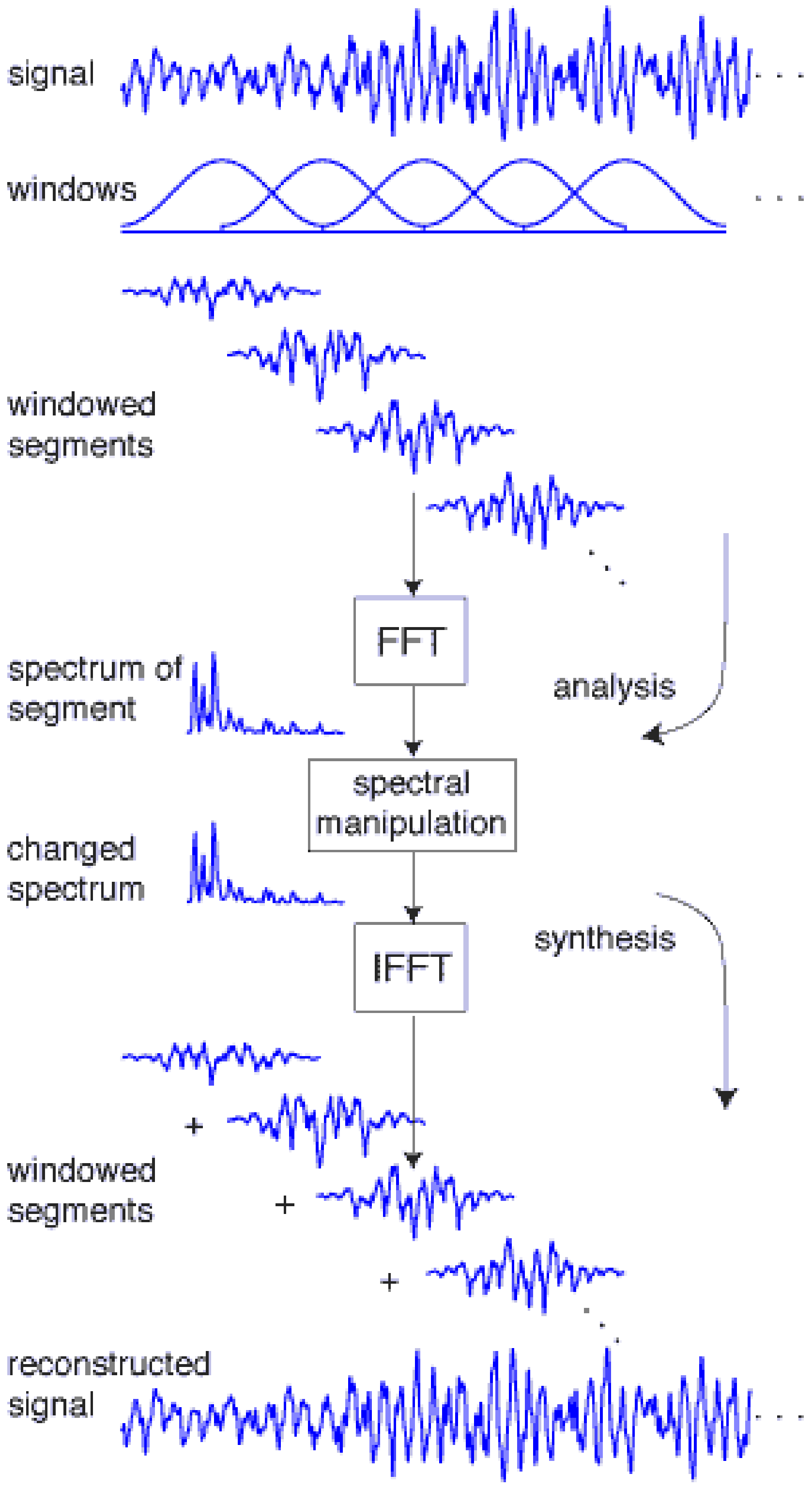


Figure 7 – The steps behind a basic frequency domain based pitch shifting algorithm[11]

2.3.1 Phase Vocoder[12]

As mentioned previously, the Phase Vocoder algorithm is a classic algorithm in pitch shifting, as it was one of the first methods to be able to pitch shift while preserving the time length of a signal.

As phase modification is a major part of this algorithm, it is important to know what phase is. The phase of a wave at any given point is the measure of the progression of the cycle. This progression is measured in degrees, with 360 degrees being a complete cycle. The cycle starts from the baseline (0^o), increases to the peak (90^o), goes back down to the baseline (180^o), sinks to the valley (270^o), and then finally returns back to the baseline (360^o or 0^o). **Figure 8** below shows the various stages of phase, which is also known as the phase angle, and it also shows what a phase shift looks like[5].

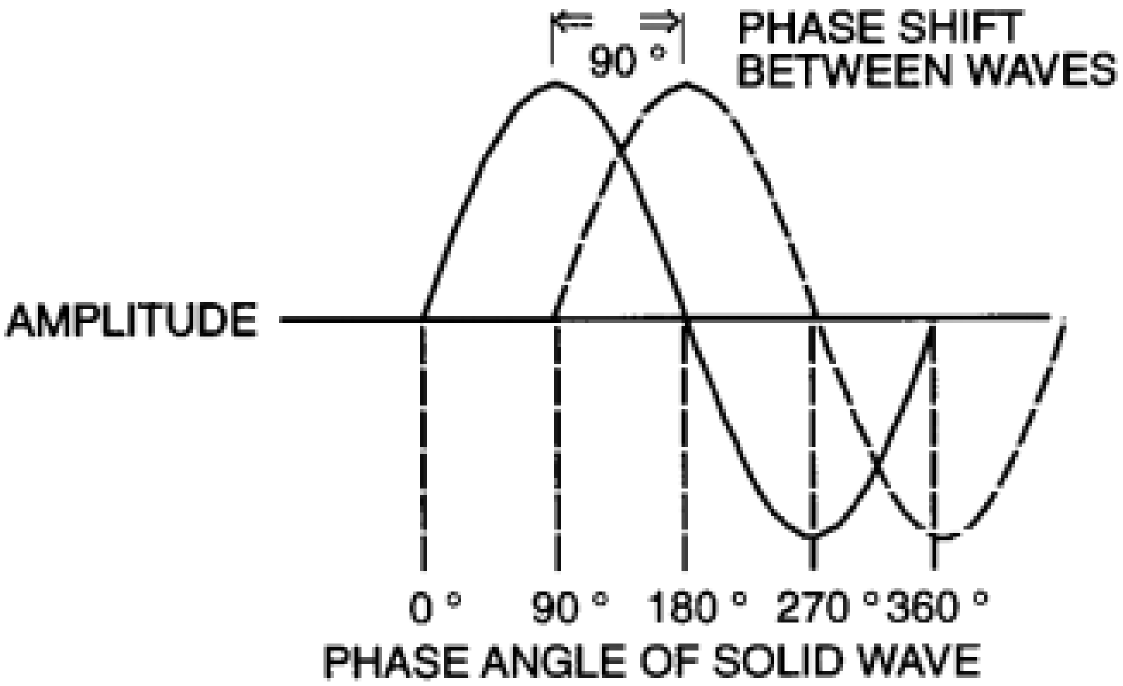


Figure 8 – Phase and Phase Shift[5]

The basic steps of the algorithm are as follows:

1) Time-shift overlapping blocks (windows) to alter the length of the signal while preserving the pitch. As shown in **Figure 8**, shifting the blocks forward in time elongates the length, and shifting them backwards in time shortens the length.
2) Take the FFT of each overlapping part of the blocks and modify the phase of each bin to ensure phase continuity.
3) Resynthesize blocks using the inverse FFT
4) Resample blocks to restore the original time length while changing the frequency.

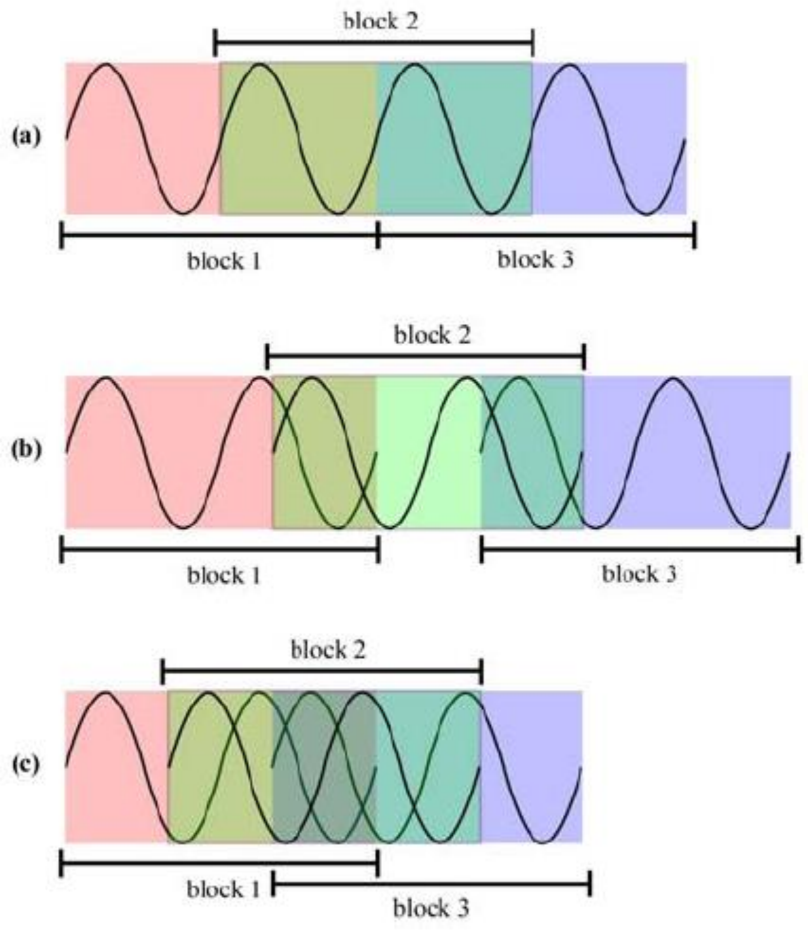


Figure 9 – Time-shifting of overlapping blocks. (a) is the original signal. (b) shifts the blocks forward in time and extends the length. (c) shifts the blocks backwards in time and shortens the length[12].

2.4 Pitch Shifting in the Time Domain

The speed and simplicity of pitch shifting algorithms in the time domain make them good candidates for real time processing. Most of these algorithms are variants of Overlap and Add (OLA) methods. In particular, the Pitch Synchronous Overlap and Add (PSOLA) method will be discussed as it was used in the implementation because of its relative robustness and efficiency.

2.5 Pitch Synchronous Overlap and Add (PSOLA)[17]

As with most pitch shifting algorithms, PSOLA is divided into two main parts – Analysis and Synthesis. These two parts can then be further divided into two steps each.

1) Analysis
    - a) Pitch Detection
    - b) Pitch Marking
2) Synthesis
    - a) Synthesis Pitch Marking
    - b) Overlap and Add

2.5.1 Pitch Detection – AutoCorrelation Function (AC)

Pitch detection is the process of finding the dominant pitch in a sound signal by finding the fundamental frequency. Autocorrelation is a basic time domain pitch detection algorithm that takes the cross-correlation of a signal with itself. It takes advantage of the fact that periodic signals are similar from one period to the next and computes the similarity between the signal and time-lagged versions of itself. The correlation signal should have the greatest peak in magnitude at the time lag that corresponds to the pitch period of the fundamental frequency of the signal. The following equation can be used to compute the cross-correlation of a signal s[m][13]:

$$R[k] = \sum_{m=0}^{N-k-1} s[m]s[m+k] \qquad (1)$$

N is the total number of samples in a window and k is the lag index. This function will have the greatest values at time lag 0 because the similarity of a signal with itself is 100%. If the signal is periodic, the next greatest peak after lag 0 should correspond to the fundamental period of the signal. **Figures 10, 11, and 12** shown below depict the correlation values of different time lags of a signal.

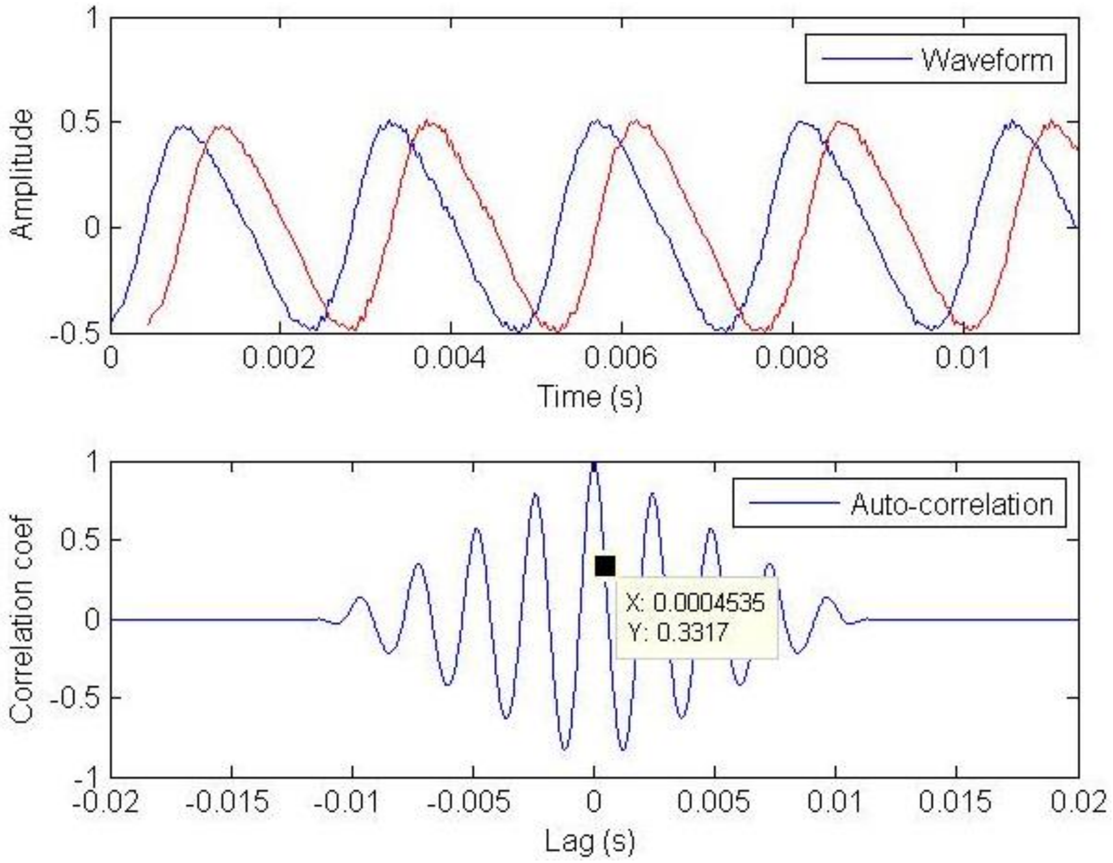


Figure 10 – Slight time lag between the original (blue) and lagged (red) signal. The correlation value is 0.3317, which is fairly low.

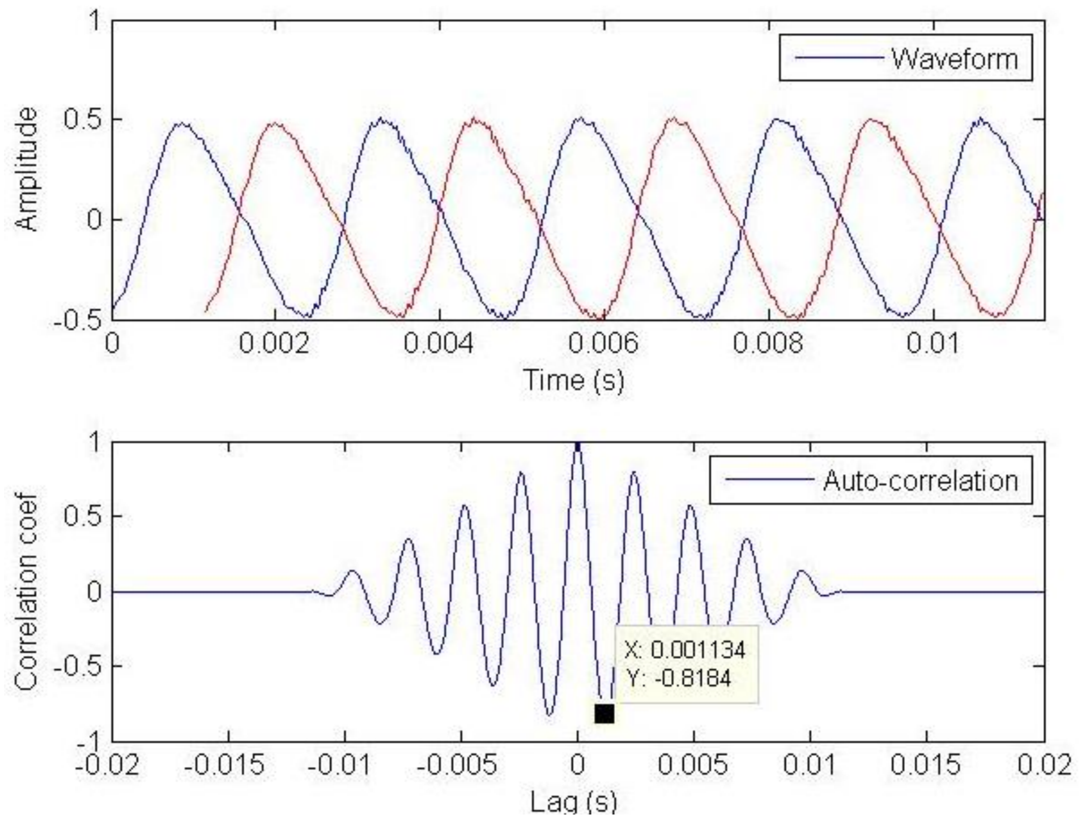


Figure 11 – Medium time lag between the original (blue) and lagged (red) signal. The correlation value is -0.8184, which is extremely low.

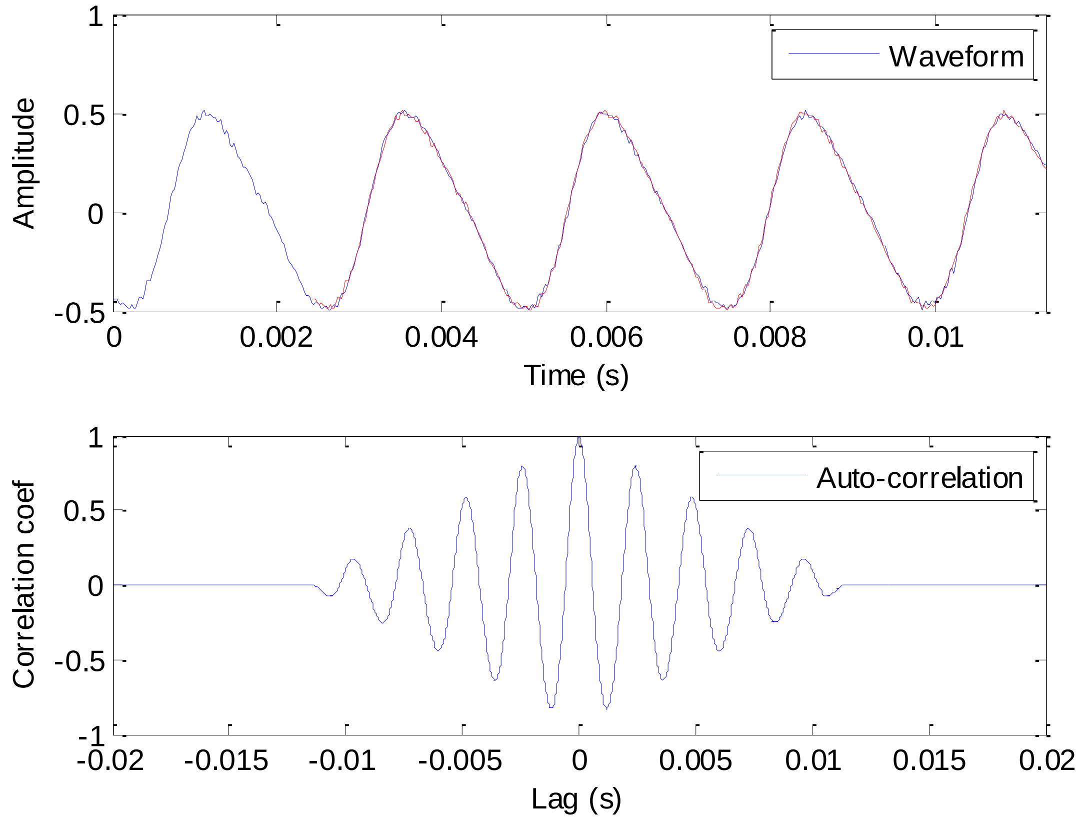


Figure 12 – Large time lag between the original (blue) and lagged (red) signal. The lag amount is almost exactly one pitch period, resulting in a peak correlation value of 0.7744.

Basic autocorrelation by itself is not reliable or accurate enough, however, as it is reported to have a gross error rate of 10% in tests, where gross errors are defined to be a deviation of at least 20% from the correct fundamental frequency[14]. Most of these errors are a result of pitch doubling or pitch halving, where the algorithm chooses the wrong peak in the autocorrelation values.

Instead, I chose to use the YIN algorithm, which boasts a gross error rate of 0.5% in tests. The YIN algorithm is based on autocorrelation, and it uses 5 additional steps to improve the results by reducing the common errors of autocorrelation and increasing accuracy through parabolic interpolation[14].

2.5.2 Pitch Marking

The quality of the synthesis of PSOLA relies on the proper synchronization of its analysis, so the accuracy of pitch marks are an essential part of the algorithm. In the interests of accuracy, speed, and the lack of additional equipment such as a laryngograph, I decided to use a two-phase pitch marking algorithm utilizing dynamic programming to pick optimal pitch mark candidates[15].

In general, pitch marks should simultaneously address two optimization criteria:

1) Magnitude of amplitude should be as large as possible
2) Distance between adjacent pitch marks should be as close to the fundamental period as possible

The first phase of the algorithm consists of finding the pitch marks at the peaks and valleys separately, and then choosing one of the two to use. In order to avoid phase discontinuities, pitch marks must remain consistent at either the peaks or the valleys. For the sake of simplicity, only peaks will be discussed, as the same algorithm can be used on a negated signal to find the valleys. The first step of this phase of the algorithm is to find the global maximum of the waveform and make this the first pitch mark. This mark is circled below in **Figure 13**.

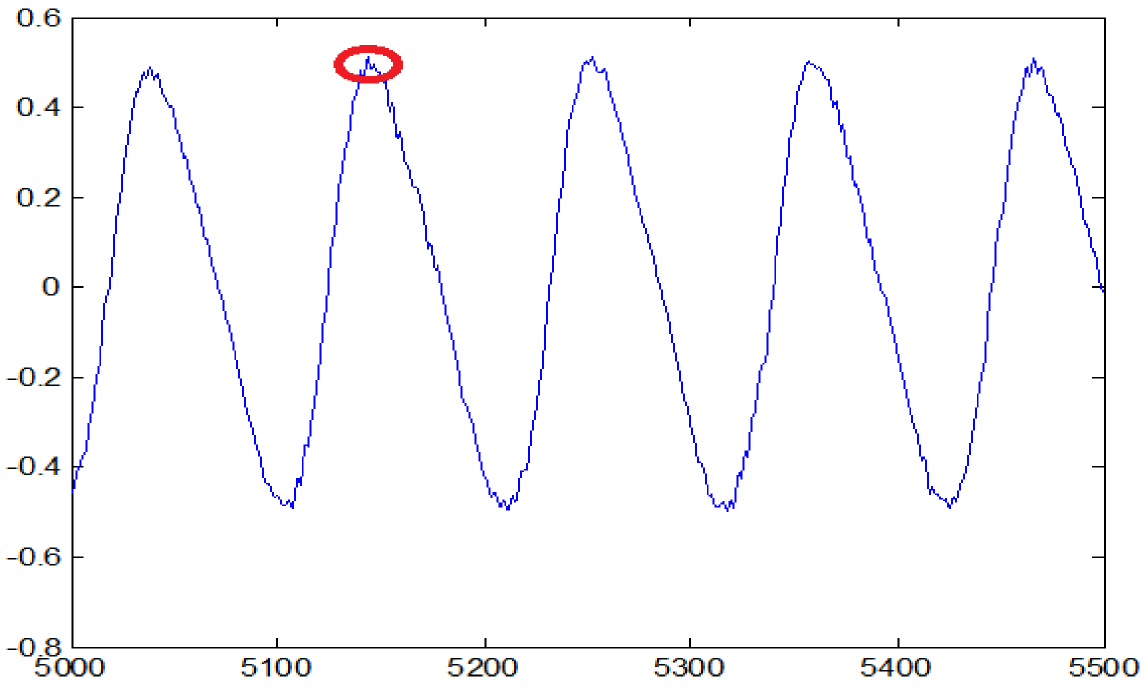


Figure 13 – The global maximum of a signal circled in red

Looking to the right of the current pitch mark, the local maximum is found within a specific search region. The next pitch mark should be roughly one fundamental period away, so the search region is limited to the region close to that. The search region is given by [t_m + f*T_0, t_m + (2 – f)*T_0], where t_m is the current pitch mark, f is a constant whose range is 0.5-0.9 (which is usually set to 0.7), and T_0 is the current fundamental period, which should have been found in the previous pitch detection step. The green lines in **Figure 14** show the search range, and the smaller red circle represents the local maximum within that region.

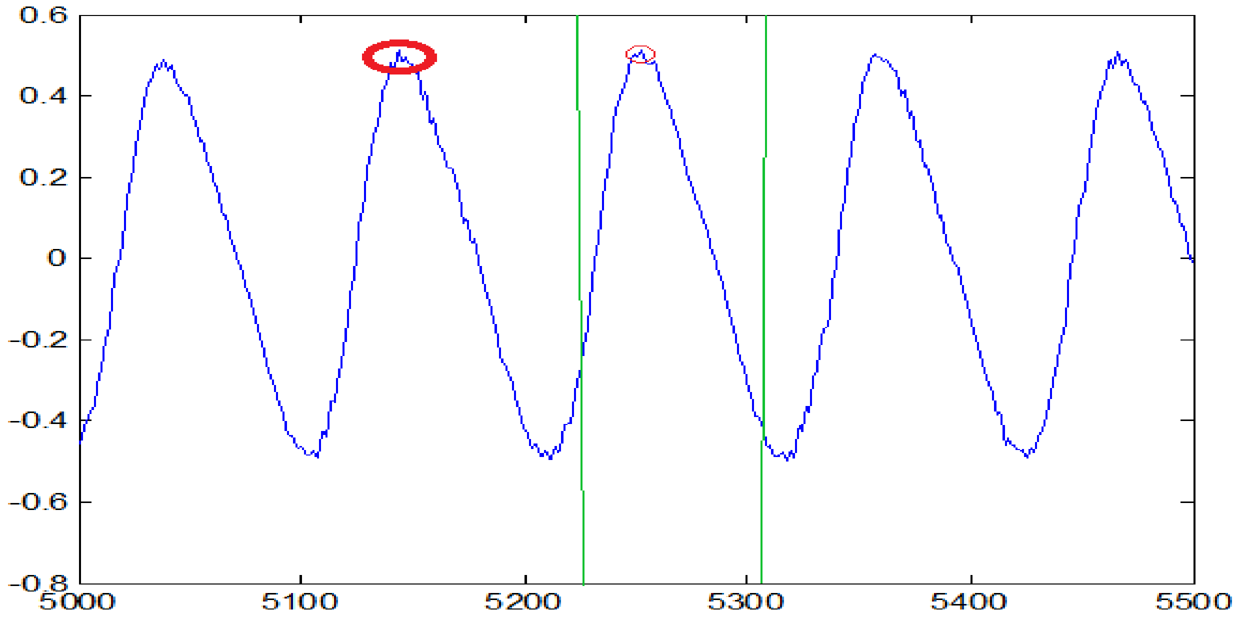


Figure 14 – The search region is marked by the green lines, and the local maximum is marked by the small red circle

This process is repeated until it reaches the end of a signal or an unperiodic section of the signal, where the pitch detection was unable to find a fundamental frequency. This process can also be repeated towards the left of the global maximum to fully mark the signal.

This simple algorithm is unreliable, however, as it only maximizes magnitude and not pitch mark distances. As shown in **Figure 15**, the largest peak does not always correspond to the largest peak in the next period[16].

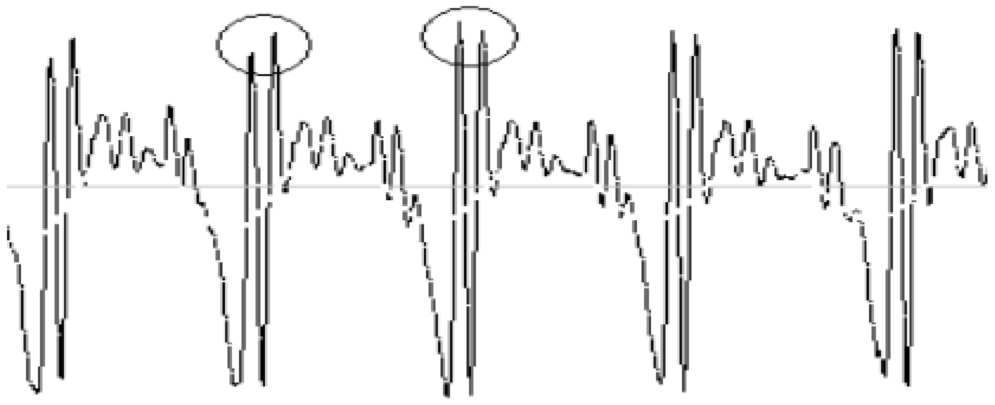


Figure 15 – The two circled peaks are the local maxima in their respective search regions, leading to an incorrect pitch mark[16]

The second phase of this algorithm attempts to fix this error by first finding multiple candidates within each search region and then using dynamic programming to find the best sequence of pitch mark candidates.

Each pitch mark candidate is assigned a state probability that is associated with its relative magnitude and is given by the following equation:

$$\bar{s}_i(j) = \frac{h_i(j) - h_{\min}}{h_{\max} - h_{\min}}$$

Where j ranges from 1 to n (the number of pitch mark candidates), $h_i(j)$ is the height of candidate j in region i, and h_{max} and h_{min} are the max and min of the signal.

Each search region (except the last one) is also assigned an nxn matrix to represent the transition probabilities of the n candidates in a given search region to the n candidates in the next. The transition probability is associated with the similarity of the distance between pitch marks and the detected fundamental period found in the previous pitch detection process and is given by the following equation:

$$\bar{t}_i(j_1, j_2) = \frac{1}{1 + \beta\left|f - \frac{f_s}{d}\right|}$$

Where f is the detected fundamental frequency, f_s is the sampling rate, d is the distance (in sample points) between candidates j_1 and j_2, and β is a fine tuning parameter (I left this at 1).

These probabilities could then be used to assign accumulated probabilities from the first pitch mark (typically the global maximum). The first pitch mark would have an accumulated probability of 1. The rest of the accumulated probabilities of pitch mark candidates are calculated dynamically; the candidate in the previous search region that gives the highest sum of accumulated probability and transition probability to that particular candidate in the current search region is found, and the accumulated probability is calculated by adding that sum to the state probability of the current candidate. The previous candidate used is saved for each new candidate calculated, so that the optimal path can be backtracked at the end. In **Figure 16** below, the number of candidates per region is limited to two, and the optimal path of pitch marks can include any combination of white and black circles (representing different peaks)[16].

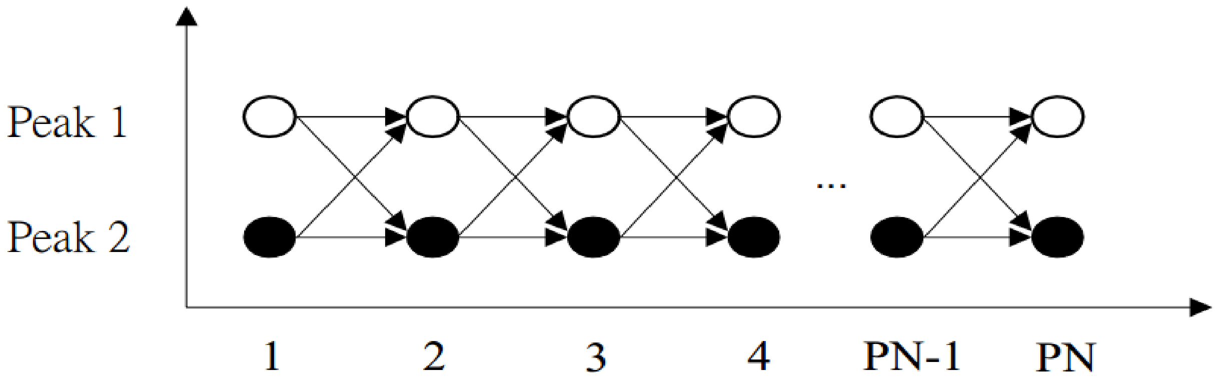


Figure 16 – Each candidate is dynamically assigned an accumulated probability, and the path with the highest total accumulated probability is chosen at the end[16]

The effectiveness of this two-phase algorithm was tested by comparing it (using 3 candidates per search region) to the simple one-phase algorithm that only looked for one max peak. I tested these two algorithms with 4 different types of frequencies; low (100Hz), medium (200Hz), high (400Hz), and moving (ranging from 100-400Hz). The accuracy of these algorithms was measured by the deviation of consecutive pitch mark differences from expected periods. The variance of the differences between consecutive pitch marks was found by calculating the average squared difference between the expected period (the detected fundamental period in the previous step) and the detected period (the difference between consecutive pitch marks). The standard deviation was then calculated by finding the square root of the variance. This standard deviation was then divided by the average period to get a relative idea of the magnitude of the standard deviation compared to the average period. The comparison between the performances of the two algorithms is shown in **Figure 17** and **Figure 18** below.

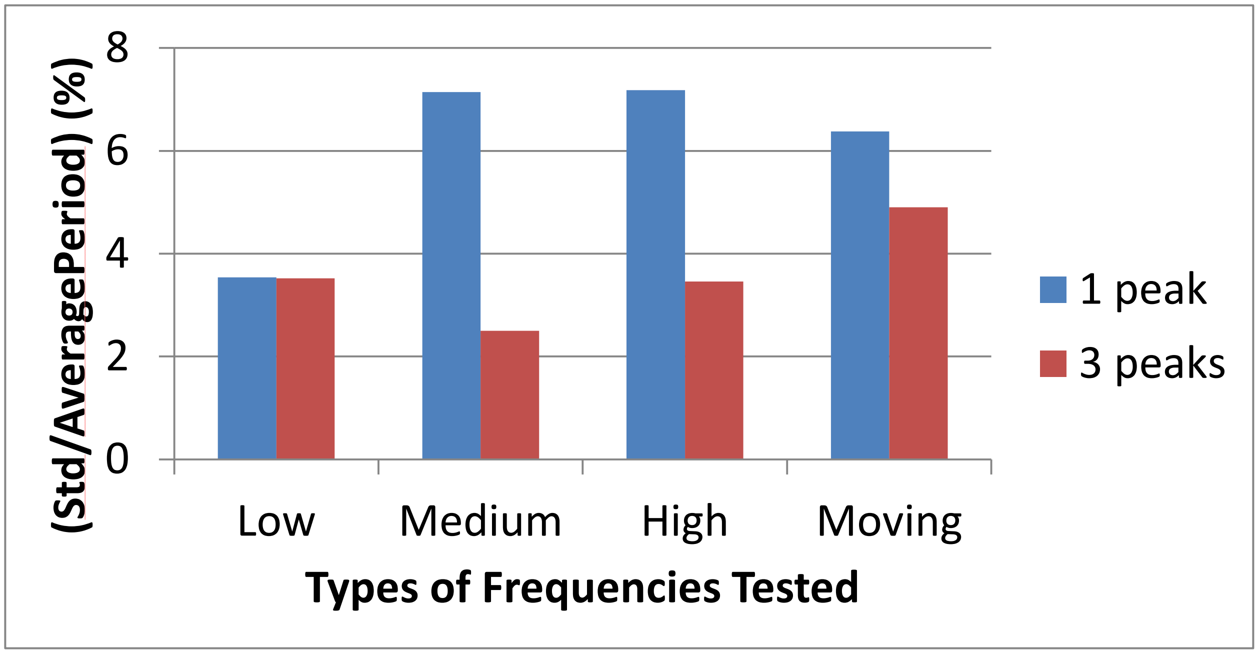


Figure 17 – Using 3 candidate marks per search region significantly reduced the error rate and gave smoother results

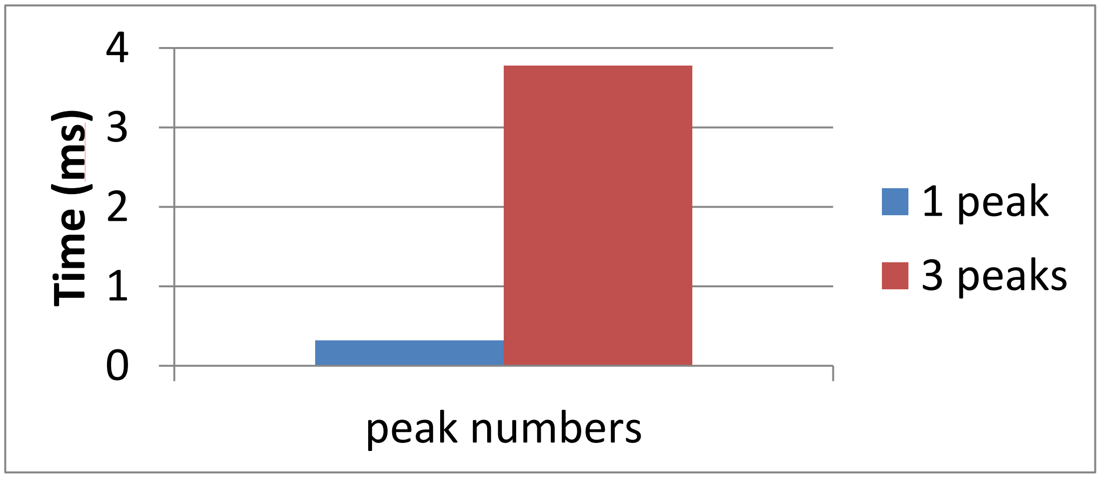


Figure 18 – Using 3 candidate marks per search region required a significant increase in computation, but there was more than enough leeway to disregard this time constraint

2.5.3 Synthesis Pitch Marks

Once the pitch marks have been found, the synthesis pitch marks need to be placed. These synthesis pitch marks are spaced relative to the distances between the analysis pitch marks, but synthesis marks can be affected by a shift factor. If the shift factor *B* is given by (desired frequency) / (original frequency), the distance between consecutive synthesis marks should be the (current period) / *B*. If the desired frequency is higher than the original, then the period will be smaller, and thus the synthesis marks will be closer together than the analysis marks. This would eventually lead to a reconstructed signal with a higher frequency in the next step, with the opposite occurring if the desired frequency is lower. The first synthesis pitch mark starts at the first analysis pitch mark, but each synthesis mark thereafter is (current period) / *B* samples away. In **Figure 19** below, the analysis step finds 4 pitch marks, but the synthesis step results in 5 synthesis marks, which leads to a higher pitch[17].

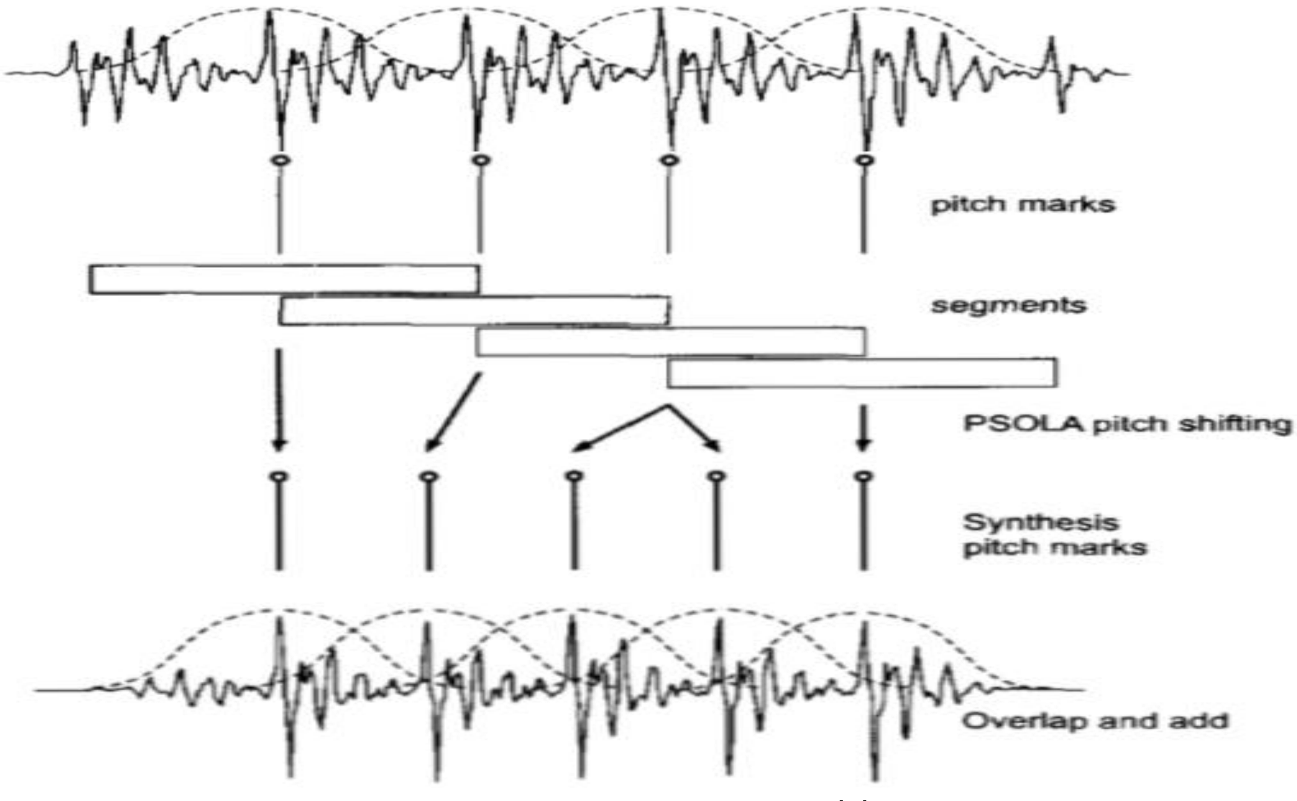


Figure 19 – A full diagram of the steps in PSOLA[17]

2.5.4 Overlap and Add

Once the synthesis pitch marks are placed, the algorithm continues with the final step – the reconstruction of the shifted signal. The shifted signal starts off empty, and the algorithm starts on the initial synthesis mark and then does the following:

1) Find the closest pitch mark p to current synthesis mark s_i
2) Obtain a windowed segment centered on p with the width being 2 * pitch period
3) Multiply this window by a hanning window of the same size to taper off ends
4) Overlap and Add this window to the shifted signal, centering it at current synthesis mark s_i.
5) Repeat for the next synthesis mark s_{i+1}.

The number of periods in the resulting shifted signal will correspond to the number of synthesis marks, as opposed to the number of analysis marks. If these numbers differ, the pitch will have successfully shifted.

3. Android Implementation

This pitch shifting project was implemented in Java on my personal phone, a 2-year-old Android device (HTC Droid Incredible) with a single core 1GHz processer. Although it has fairly weak performance compared to the quad-core smartphones of today, my phone was able to handle the sound processing in real time well enough.

3.1 Android Data Flow

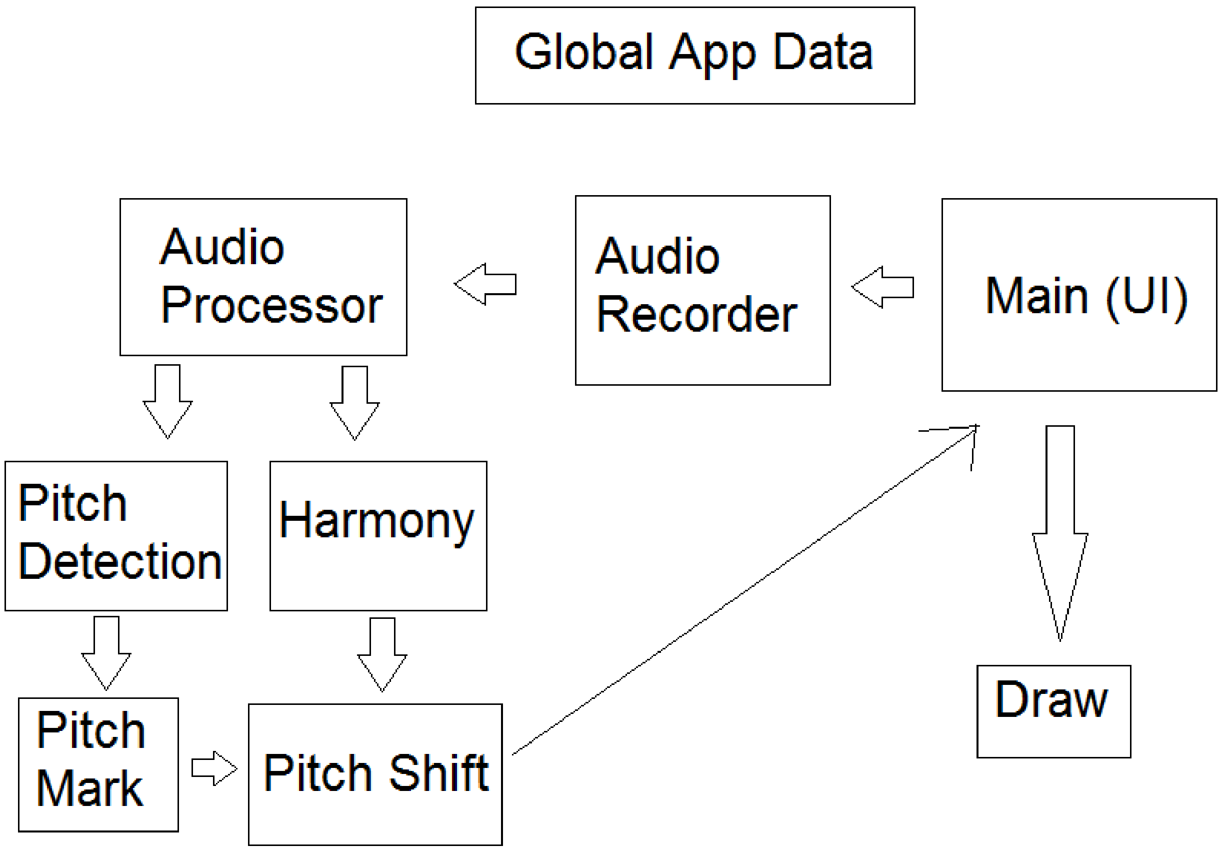
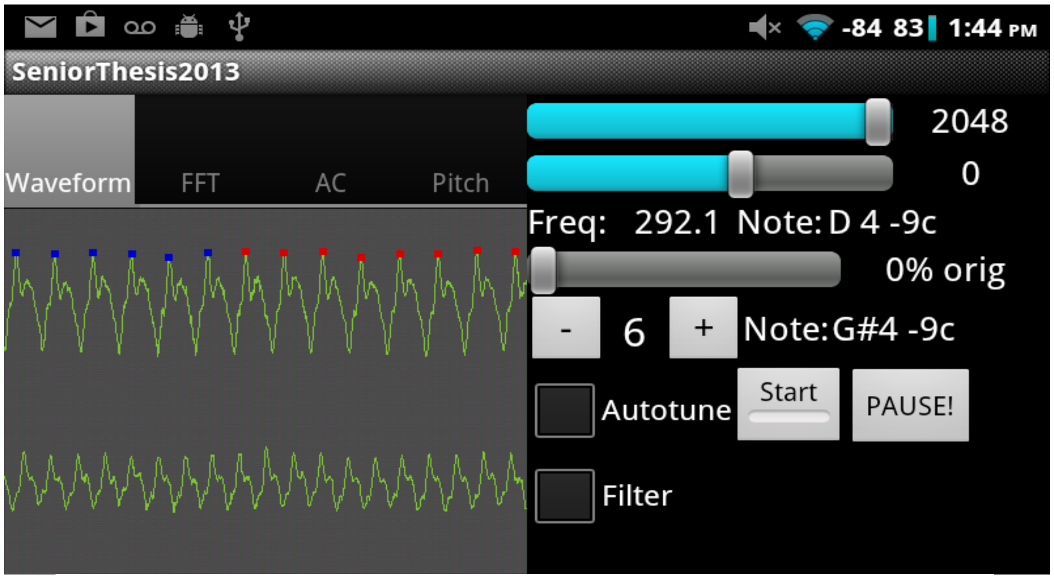


Figure 20 – A simplified diagram of the Android project design

The flow mostly revolved around the Main, Audio Recorder, Audio Processor, and Draw threads. The Audio Recorder thread obtains data from the microphone and passes it over to the Audio Processor thread using a double buffer system to avoid concurrency issues. The Audio Processor thread then shifts the microphone data if pitch is detected by using the

PSOLA algorithm. The amount to shift is determined by either the user or the Harmonizer class (experimental). This shifted signal data is then sent back to the Main thread, which then passes it to the Draw thread to display 1 of 4 options:

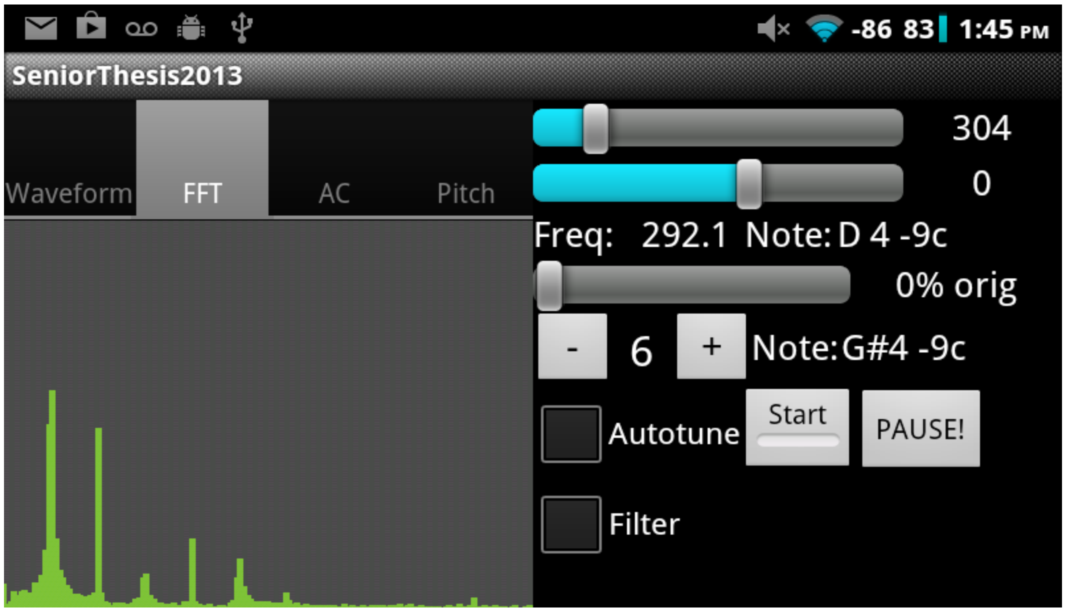
1) Waveform



2048
0
Freq: 292.1 Note: D 4 -9c
0% orig
6 Note: G#4 -9c
Autotune
Filter

Figure 21 – The top plot shows the waveform of my voice singing the tone D4. The blue and red boxes are the pitch marks found on this signal. The bottom plot shows the waveform of the shifted tone (G#4).

2) FFT



304
0
Freq: 292.1 Note: D 4 -9c
0% orig
6 Note: G#4 -9c
Autotune
Filter

Figure 22 – Diagram of the FFT plot of the same signal

3) YIN

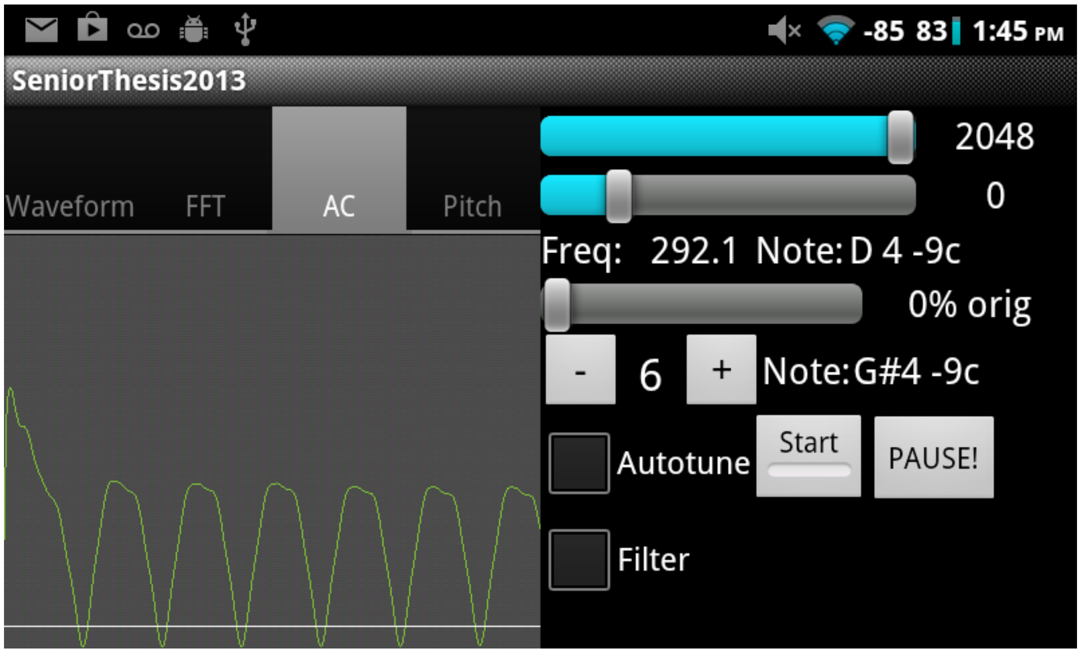


Figure 23 – Diagram of the YIN coefficient values. The white line represents the threshold value (the algorithm chooses the first valley that is less than the threshold

4) Pitch

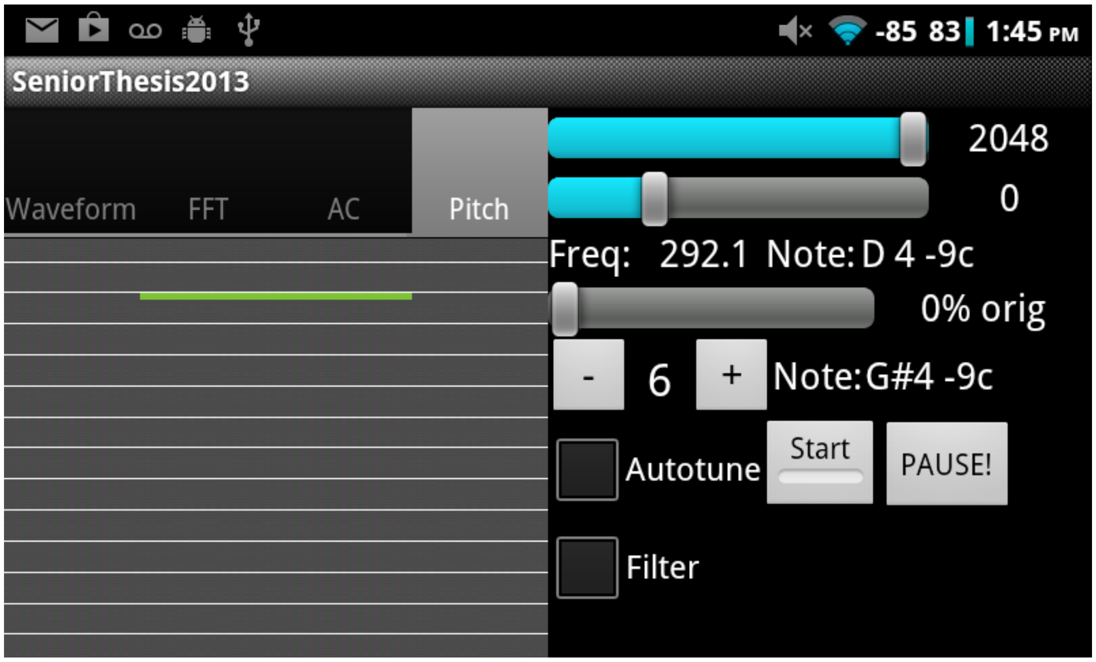


Figure 24 – Diagram of the relative pitch. The green line represents the tone detected from the microphone, and the white lines represent perfect semitones

3.2 Real-Time Processing Buffer System

In the class setup, new microphone data is recorded in the Audio Recorder Thread, and when a buffer of 2048 samples gets full, it sends that block to the Audio Processor Thread. **Figure 25** below depicts how this data is then processed.

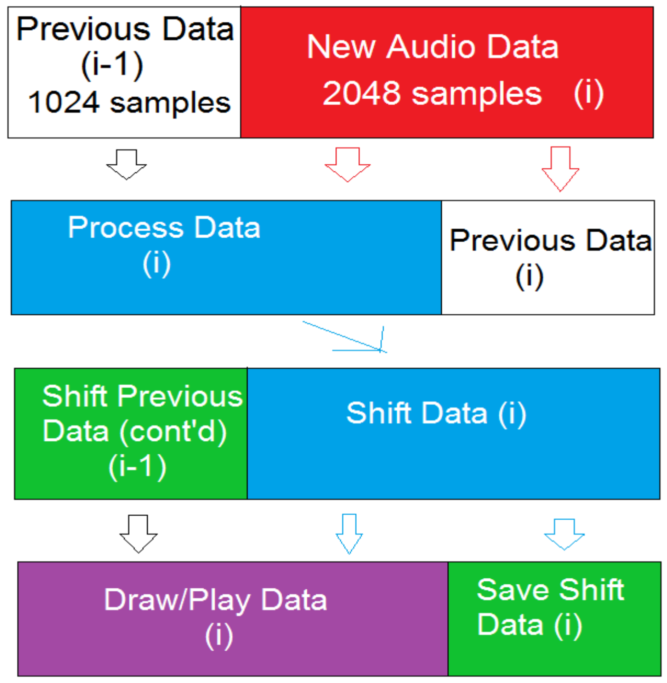

Previous Data (i-1) 1024 samples
New Audio Data 2048 samples (i)
Process Data (i)
Previous Data (i)
Shift Previous Data (cont'd) (i-1)
Shift Data (i)
Draw/Play Data (i)
Save Shift Data (i)

Figure 25 – Data flow of buffers for an arbitrary frame i

For every new block of audio data, the latter half (1024 samples) are saved to use in the next frame. The latter half of the shifted (post processed) data is also saved for the next frame. These half buffers will be referred to as 'previous' data, and when the program is initialized, these buffers are zeroed. When new audio data comes in, the following steps are executed:

1) Previous Data (1024 samples) and the new audio data are pitch marked, starting from a previously saved location in the Previous Data section (if available). The latter half of the new audio data is saved to use in the next frame, and the last pitch mark location is saved.
2) Starting from a previous synthesis pitch mark in the Shift Previous Data buffer if possible, the new audio data is pitch shifted (1024 samples from previous data + 2048 samples from new audio data = 3072 samples). The latter 1048 of the shifted data is saved to use in the next frame, and the last synthesis pitch mark location is saved.
3) The first 2048 samples of the 3072 sample shifted block is used as output in the draw class and as output sound for the speakers.

4. Conclusion

Overall, the implementation of this project was very successful. My outdated phone was able to meet all the goals that were outlined; It was able to pitch shift in real time while preserving the original acoustical characteristics and time length. Additionally, I was able to explore and experiment with things like Autotune, automatic harmonization, and the use of filters. There is a lot of potential for the exploration of these 3 subtopics alone. For example, the addition of a zero-phase filter could help with the pre or post processing steps. The harmonization could also be vastly improved, by implementing more rules, such as those regarding the progression of the harmony which would improve the sound. Furthermore, my implementation only used one algorithm, and thus I was not really able to compare the sound quality or computational speed of competing algorithms. It would be interesting to try out other algorithms and implement those properly instead of just reading about them.

5. Acknowledgements

I would like to thank Professor Alvarez for all the advice and encouragement given throughout the past few years pertaining to my thesis, my education, and my personal growth. This thesis would lack both breadth and depth if it were not for him. Professor Alvarez is not someone you would want to disappoint.

I would also like to thank Professor Signorile for answering all my random questions about Android design. The application flow improved significantly with his help, and real time processing became possible as a result.

6. References

[1] "Audio time-scale/pitch modification - Wikipedia, the free encyclopedia." Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Audio_time-scale/pitch_modification (accessed May 16, 2013).

[2] G. Fairbanks, W. Everitt, and R. Jaeger, "Method for time of frequency compression-expansion of speech," *Transactions of the IRE Professional Group on Audio, vol.2, no.1, pp. 7-12,* Jan 1954

[3] "1975 Eventide H910 Harmonizer-Mix Inducts Eventide H910 Harmonizer Into 2007 TECnology Hall of Fame.*" Mix Magazine | Pro Audio, Live Sound, Music Recording and Live Post for Audio Pros.* http://mixonline.com/TECnology-Hall-of-Fame/1975-eventide-harmonizer (accessed May 16, 2013).

[4] PRASHANT. "Acoustics." Padante. padante.com/acoustics/ (accessed May 16, 2013).

[5] Goldline Inc.. "Basics of Sound." Audio Engineering Society. www.aes.org/sections/pnw/reference/basics_of_sound.pdf (accessed May 16, 2013).

[6] "Frequency Hearing Ranges in Dogs and Other Species." Louisiana State University. http://www.lsu.edu/deafness/HearingRange.html (accessed May 16, 2013).

[7] "Missing Fundamentals. Periodicity and Pitch | Auditory Neuroscience." Auditory Neuroscience. http://auditoryneuroscience.com/topics/missing-fundamental (accessed May 16, 2013).

[8] J. L. Flanagan, R. M. Golden, "Phase Vocoder," *Bell System Technical Journal, pp. 1493-1509,* November 1966.

[9] M. R. Portnoff, "Implementation of the Digital Phase Vocoder Using the Fast Fourier Transform," *IEEE Trans. Acous. Speech, and Signal Proc., vol. ASSP-24, pp. 243-248,* June 1976

[10] "Time and Frequency Domain." Audiophile Headphone Amplification - Erzetich Audio. http://www.erzetich-audio.com/knowledgebase-05-time-vs-frequency (accessed May 16, 2013).

[11] "A Phase Vocoder in Matlab." University of Wisconsin-Madison College of Engineering. http://sethares.engr.wisc.edu/vocoders/phasevocoder.html (accessed May 16, 2013).

[12] Sawyer, Scott, Habib Estephan, and Daniel Wanninger. "Real-Time Pitch Shifting on an FPGA - II. Design Overview - Frequency-Domain Pitch Shifting." Villanova University. www56.homepage.villanova.edu/scott.sawyer/fpga/II_freq_domain.htm (accessed May 16, 2013).

[13] M. Ghulam, "Noise-Robust Pitch Detection using Auto-correlation Function with Enhancements," *J. King Saud University, Vol. 22, Comp. & Info. Sci., pp. 13-28,*Riyadh (1431H./2010)

[14] A. de Cheveigne and H. Kawahara, "YIN, a fundamental frequency estimator for speech and music," *Journal of the Acoustical Society of America, vol. 111, pp. 1917–1930,* 2002.

[15] C.-Y. Lin and J.-S. R. Jang, "A two-phase pitch marking method for TD-PSOLA synthesis," in *Proceedings of Interspeech/ICSLP pp. 1189–1192*, Jeju Island (Korea), October 2004.

[16] Chen, J.-H. and Kao, Y.-A., "Pitch marking based on an adaptable filter and a peak-valley estimation method", *Computational Linguistics and Chinese Language Processing, Vol. 6, No. 5, pp. 1-12,* February 2001.

[17] Zölzer, U. e. a., *DAFX Digital Audio Effects,* John Wiley & Sons, eds., 2002