Speak and Sing

Collection Editor:
Matt Szalkowski

Speak and Sing

Collection Editor:
Matt Szalkowski

Authors:
Graham Houser
Alysha Jeans
Sam Soundar
Matt Szalkowski

Online:
< http://cnx.org/content/col11151/1.1/ >

CONNEXIONS

Rice University, Houston, Texas

Collection structure revised: December 21, 2009
PDF generated: February 6, 2011
For copyright and attribution information for the modules contained in this collection, see p. 30.

Table of Contents

Chapter 1

Speak and Sing - Introduction[1]

1.1 Speech Scaling and Pitch Correction

The speech scaling and pitch correction program, or “Speak & Sing”, generates a properly-timed and pitch-accurate sample of a known song from recorded spoken words.

A voice modulation application which detects the timing and pitch of the recorded input and automatically performs time scaling and pitch correction to match the speech to a pre-selected song, producing a musical output.

1.1.1 Making a Smarter Autotuner

Pitch correction is used by musicians to improve the sound of song vocals by fixing off-key singing or adding distortion. It can be applied real-time using a synthesizer keyboard or added after recording. However, these “autotuners” can’t fix off-tempo singing, and automatic autotuners depend on the singer to be relatively close to the right pitch.

Goals for the Speak and Sing:

- Proof-of-concept of automated syllable detection, time scaling, and pitch correction in one robust application
- Provides open-ended, customizable options for audio processing
- Demonstrates time and frequency DSP applications using MatLab

1.1.2 Implementation: An Overview

Recording:

Input voice samples are recorded in mono-channel audio at a sampling frequency of 16,000 (although any sampling frequency can be used). It is then imported into MatLab and the following functions are run in sequence:

Song Interpretation and Retrieval:

Contains data for selected songs based on sheet music. It returns a vector of fundamental note frequencies and note lengths depending on the song selected and the desired tempo.

Available Songs:
1. Christina Aguilera - *Genie in a Bottle*
2. *Mary Had a Little Lamb*
3. *Row, Row, Row Your Boat*

Syllable Detection

[1] This content is available online at <http://cnx.org/content/m33229/1.1/>.

Analyzes the input speech data and determines the locations and lengths of each syllable. After dividing the signal up into several short windowed pieces, it detects the periodicity and energy of each window to determine the type of sound (vowel, consonant, or noise). The locations of the syllables is then determined based on the pattern of sounds.

Time scaling

Interprets the detected syllable locations and stretches or shrinks the syllable to match the length of the word in the song. The time scaling is performed using a time-domain Waveform Similarity Overlap Add (WSOLA) algorithm, which breaks up the signal into overlapping windows and copies each window to a new location, either closer together or further apart. This stretches or compresses the length of the speech without losing quality or information.

Pitch correction

Detects the pitch of the signal, compares it to the desired pitch of the song, and makes pitch corrections. Pitch detection is done using FAST-autocorrelation, in which small windows of the signal are offset and autocorrelated to find the period, and thus frequency, of the signal for that interval. Pitch correction is performed with Pitch Synchronous Overlap Add (PSOLA), which moves windowed segments closer or further apart and overlap-added to alter the frequency without loss of sound.

1.1.3 The Result

The resulting audio file sounds like the input speech or singing, but the words will now line up with those of the original song and the pitch will be adjusted. The resulting impact on sound is that the recorded input will now sound more like the song, without compromising the original voice or speech.

Chapter 2

Speak and Sing - Recording Procedure[1]

2.1 Recording

Input voice samples are recorded in mono-channel audio with a sampling frequency of 16,000 Hz. The sampling rate chosen allows for a balance of processing efficiency and sound quality – the computation time of the program generally scales linearly with increase in sampling rate. The selected sampling frequency is also convenient for computation, as the MatLab program's wave audio operations perform best with sampling frequencies in increments of 8,000 Hz. The program allows for the use of any sampling frequency and will perform adequately for sampling frequencies up to and beyond the audio standard 44.1 kHz, but processing time and program durability become an issue.

When recording, the best results are produced for input speech or song which is delivered slowly and clearly, with either brief pauses or strongly-enunciated consonants between syllables and words.

The recorded sound is processed in Audacity, a freeware recording software, to trim out excess electrical and environmental noise and remove existing DC offsets. It is then ready for handling in the MatLab environment.

[1] This content is available online at <http://cnx.org/content/m33233/1.1/>.

Chapter 3

Speak and Sing - Song Interpretation[1]

3.1 Song Interpretation

The songs to be used by the Speak and Sing were predetermined and preprogrammed. For simplicity, the songs feature a one-to-one format. That means every syllable of the lyrics is associated with one duration and one pitch, there are no rests and no slurs. The song interpretation is done in two vectors, one contains all of the durations (in seconds) to be used by the duration matching, and the other contains all of the note frequencies (in hertz) to be used by the pitch matching.

Here is an example of a measure of sheet music that has been turned into a useable vector.

Mary Had a Little Lamb

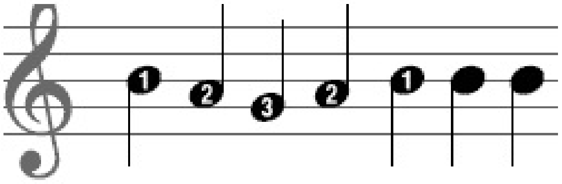


Figure 3.1

```
notes = ([246.94; 220;196; 220; 246.94; 246.94; 246.94;]);
duration = ([.5; .5; .5; .5; .5; .5; .5;]);
```

Three songs were made available for use:

1. Christina Aguilera - *Genie in a Bottle*
2. *Mary Had a Little Lamb*
3. *Row, Row, Row Your Boat*

[1] This content is available online at <http://cnx.org/content/m33237/1.1/>.

Chapter 4

Speak and Sing - Syllable Detection[1]

4.1 Syllable Detection

The syllable detection algorithm takes as its input recorded speech and produces an output matrix denoting the start and end times of each syllable in the recording. There are two main parts to the algorithm. First, each sound in the input file must be classified as a vowel, consonant, or noise. Second, the algorithm must determine which sequences of sounds correspond to valid syllables.

4.1.1 Sound Classification

The sound classification step splits the input signal into many small windows to be analyzed separately. The classification of these windows as vowels, consonants, or noise relies on two core characteristics of the signal: energy and periodicity. Vowels stand out as having the highest energy and periodicity values, noise ideally has extremely low energy, and consonants are everything that falls between these two extremes.

The energy of a window of the input vector W is calculated as

```
E = |W|^2.
```

However it is necessary to set general energy thresholds that are valid for speech samples of varying volume. In order to accomplish this, after the energies of all the windows have been calculated, they are converted into decibels relative to the maximum energy value.

```
E' = 10*log10(E/max(E)).
```

The energy thresholds are then defined in terms of a percent of the total energy range. For example, if an energy threshold was 25 percent and the energies ranged from -100 to 0 dB, then everything from -25 to 0 dB would be above the threshold.

In some cases, energy alone is enough to determine whether a certain sound is a vowel, consonant, or noise. For instance, here is a plot of the energy vs. time of a recording of the spoken word "cat." It is easy to tell which portions of the figure correspond to vowels, consonants, and noise by inspection:

[1] This content is available online at <http://cnx.org/content/m33241/1.1/>.

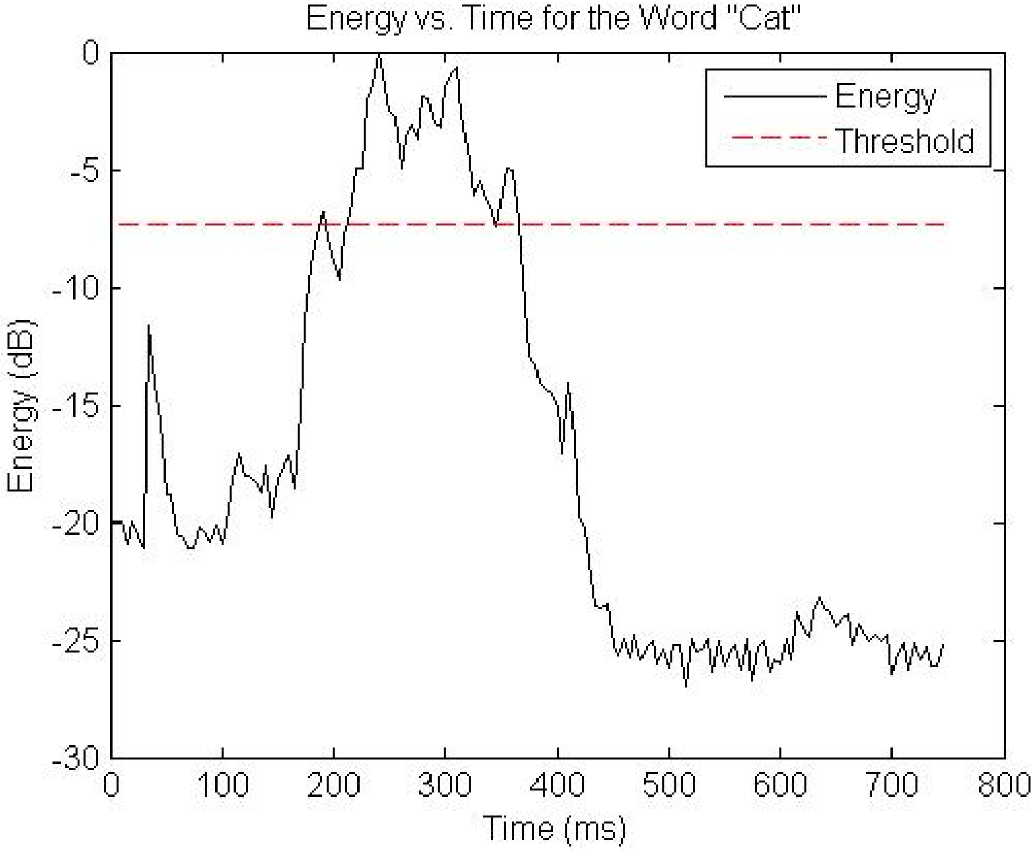


Figure 4.1: The energy threshold clearly divides the high-energy vowel portion of the signal from the consonants.

However, energy cannot always separate vowels and consonants so dramatically. For example, the word "zoo."

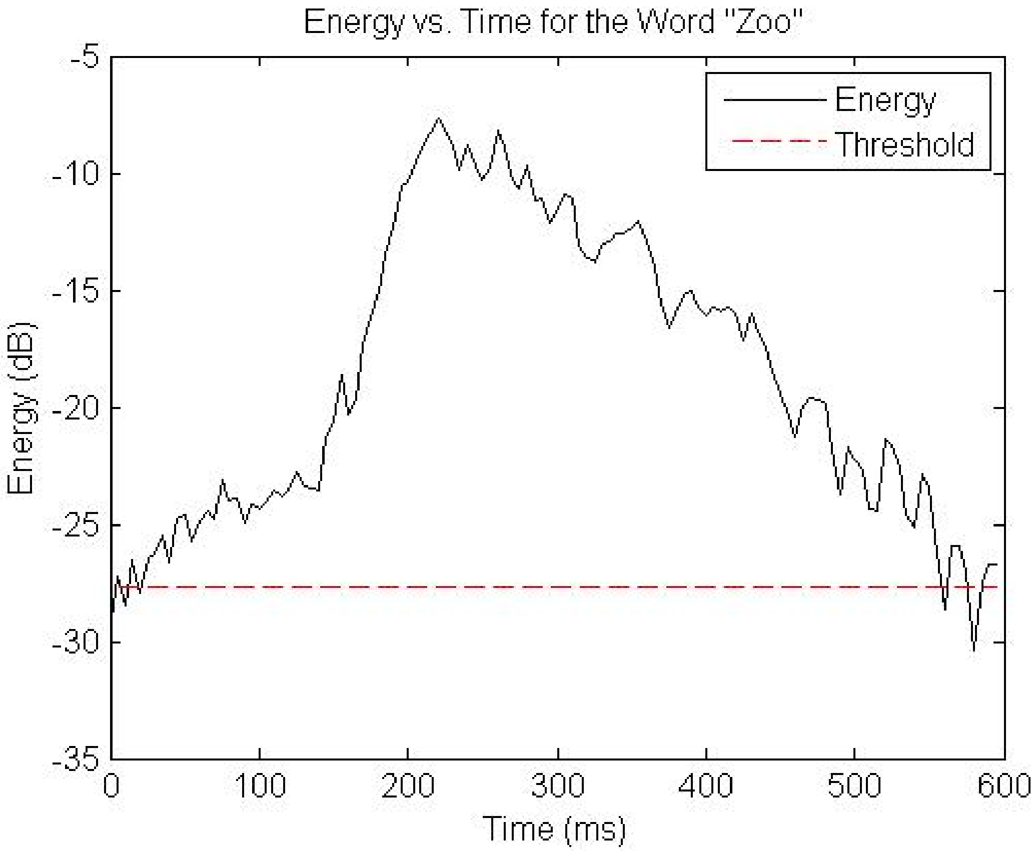


Figure 4.2: The ending vowel sound drops too close to the threshold.

Although a portion of the vowel still has significantly higher energy than the consonant, the ending portion of the vowel drops in energy to the point where it is dangerously close to the threshold. Raising the threshold so that the "z" sound is certain not to be counted as a vowel only makes it more likely that portions of the "oo" sound will be mistakenly classified as consonants. Clearly, additional steps are necessary to more accurately differentiate between consonants and vowels.

4.1.2 Periodicity Analysis

The algorithm uses the periodicity of the signal to accomplish this task. The periodicity is obtained using the autocovariance of the window being analyzed. This is calculated as:

```
C(m) = E[(W(n+m)-mu)*conj(W(n)-mu)]
```

Mu is the mean of the window W. It measures how similar the signal is to itself at shifts of m samples and can therefore distinguish periodic signals from aperiodic ones due to their repetitive nature. The autocovariance vector is most stable and therefore most meaningful for values of m relatively close to 0, since for larger m, fewer samples are considered, causing the results to become more random and unreliable. Therefore, the sound classification algorithm only considers autocovariance values with m less than 1/5 the total window size. These autocovariance values are normalized so that the value at m = 0 is 1, the largest possible value. The maximum autocovariance in this stable region is considered the periodicity of the window.

The periodicity values for vowels are extremely high, while most unvoiced, and some voiced, consonants exhibit very low periodicity. Periodicity is especially useful in detecting fricative or affricate consonants which are both characterized by a great deal of random, possibly high-energy, noise due to their method of articulation. Examples of these consonants include "s," "z," "j," and "ch." The contrast between the periodicity of a fricative consonant and a vowel can be clearly seen in this plot.

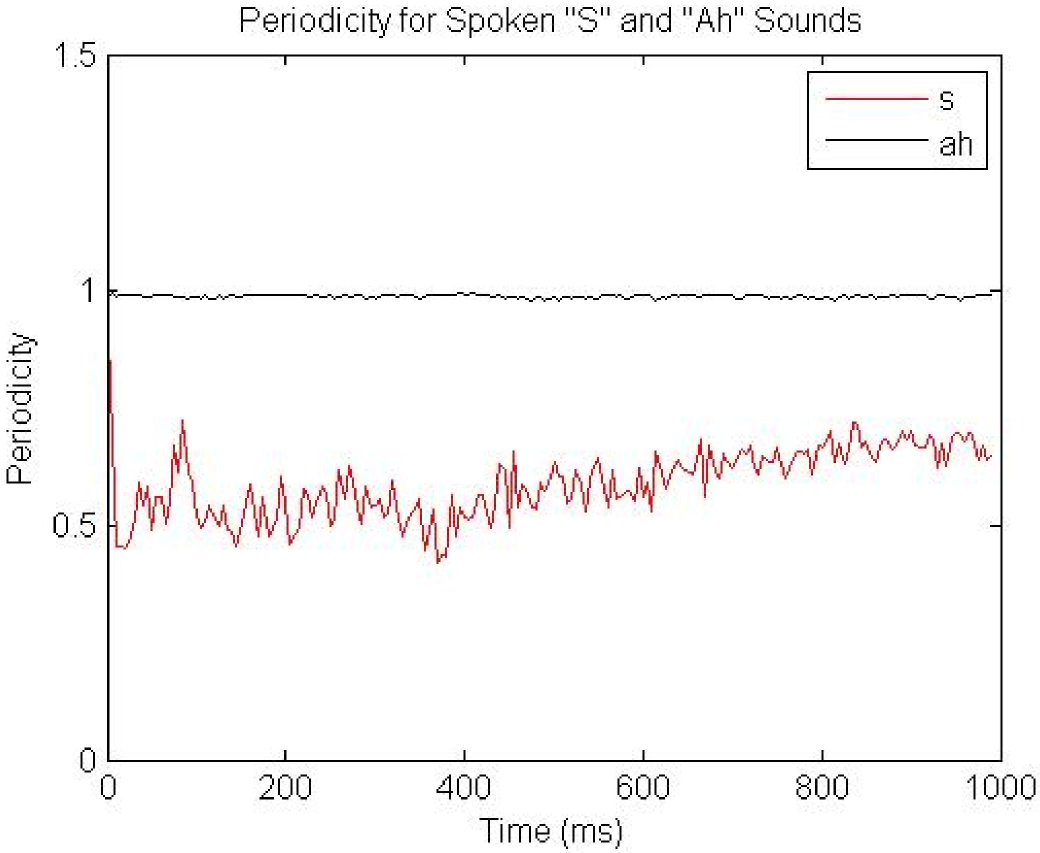


Figure 4.3

Putting it all together, the sound classification portion of the algorithm first calculates the energy and periodicity of each window of the input signal. If both the energy and periodicity are higher than certain thresholds, the window is classified as a vowel. If the energy is smaller than a very low threshold, the window is counted as noise, and everything in between is considered a consonant. Let's take another look at the energy characteristics of the word "zoo" (refer to figure 2). Using this alone, we could not easily distinguish the high-energy "z" from the lower-energy portion of the "oo." However, here is a plot of the periodicity vs. time for the same recording.

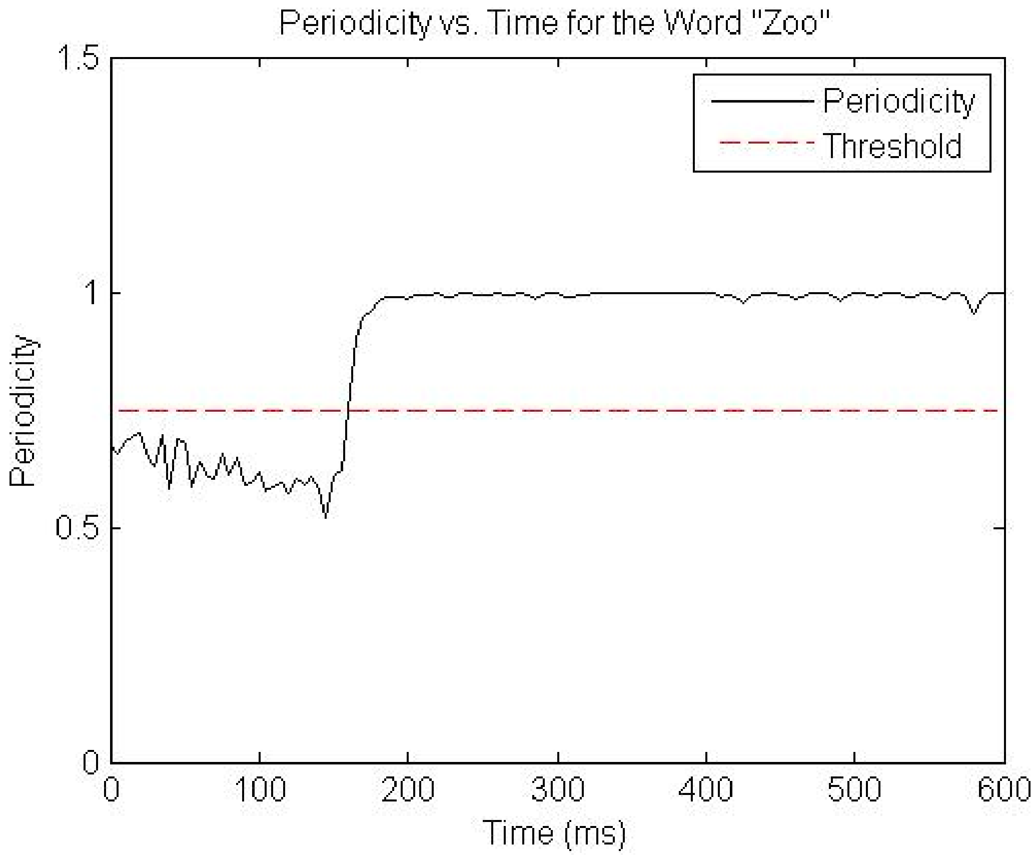


Figure 4.4: The difference between the "z" and the "oo" is now much more pronounced.

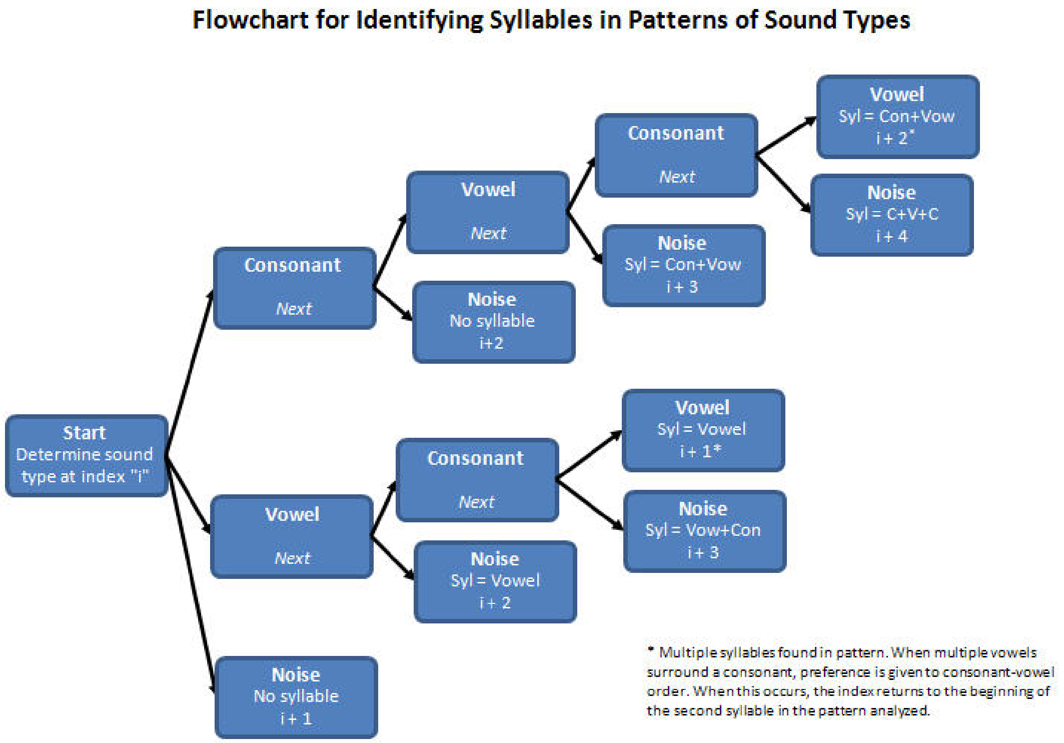
This plot shows a clear contrast between the aperiodic fricative "z" and the periodic vowel. Taken together, these data now provide sufficient information for the sound classification algorithm to correctly identify each sound in this recording.

This method works with a reasonable degree of accuracy, but there are a few challenges that must be considered. The greatest among these is the handling of liquid consonants like "l," "y," or "m." In certain cases, these sounds are used as consonants at syllable boundaries, while in other circumstances, they act as a vowel usually would in making up the majority of the syllable. For example, in the word "little," the first "l" is acting as a consonant, but the "l" sound is also used as the central portion of the second syllable. Therefore, these sounds are not always accurately classified, and they must be annunciated strongly in the input recording if they are acting as syllable boundaries.

Another issue with this method is that sometimes it detects short bursts of one sound type in the middle of another. For instance, there may be 1 or 2 consonant windows surrounded by a large number of noise windows or a small number of vowel windows in the middle of a large section of consonant windows. Several situations can lead to errors like this. For example, the background noise in a recording might boost the energy of a window high enough to be classified as a consonant, or random spikes in the periodicity of an otherwise aperiodic signal could cause part of a consonant to be classified as a vowel. These errors can be minimized by imposing a length constraint on sounds. In order for a group of windows to be classified as a particular sound, they must represent a long enough chunk of time to be considered meaningful. If the group of windows is too small, they are reclassified to match the sound immediately preceding them.

4.1.3 Syllable Interpretation

After each sound in the input has been classified, it is necessary to determine which sound sequences should be interpreted as syllables. This is accomplished using a tree-like decision structure which examines consecutive elements of the sound classification vector, comparing them to all possible sequences. Once a known sequence is identified, it is added to the list of syllables, and the algorithm moves on to the next ungrouped sounds. The decision structure is depicted in the following figure.



Flowchart for Identifying Syllables in Patterns of Sound Types

* Multiple syllables found in pattern. When multiple vowels surround a consonant, preference is given to consonant-vowel order. When this occurs, the index returns to the beginning of the second syllable in the pattern analyzed.

Figure 4.5

After this step, some syllables were occasionally much too short. For instance, the word "good" had a small probability of being split up into two syllables ("goo"" and "d") depending on how much the speaker emphasizes the voicing of the d. Further increasing the minimum allowable sound duration caused too much information to be lost or misinterpreted, so a minimum syllable duration parameter was also added. If a syllable is too short, it is combined with an adjacent syllable based on its surrounding sounds. If one of the sounds adjacent to the short syllable is noise and the other is not, the short syllable is added to the side without noise to preserve continuity of the signal. If neither sound adjacent to the syllable is noise, the duration of each adjacent sound is calculated, and the syllable is tacked onto the side with the shortest neighboring sound as this one is more likely to have been cut off in error.

The following table lists the values for the various thresholds and parameters we found worked best for relatively clean, noise-free, input signals. These parameters must be adjusted if a great deal of periodic or energetic background noise, such as might be caused by a microphone picking up the sound of a computer fan, is expected to corrupt the input recording.

Parameter	Value
Window length	5 ms
Vowel periodicity threshold	.75
Vowel energy threshold	27% of total energy range
Noise energy threshold	55% of total energy range
Minimum sound duration	40 ms
Minimum syllable length	80 ms

Table 4.1

Chapter 5

Speak and Sing - Time Scaling with WSOLA[1]

5.1 Introduction

There are many applications for time-scale modification ranging from post production of audio video synchronization in film to voicemail playback. Time-scale modification essentially is the process of either speeding up or slowing down the apparent rate of speech without corrupting other characteristics of the signal such as pitch and voice quality. Resampling is out of the question because is directly modifies pitch and very often voice quality loss is significant. To maintain these characteristics, the short-time Fourier transform of corresponding regions of the original (input) and scaled (output) signals should be very similar. Overlap add algorithms achieve this by simply cutting out smoothly windowed chunks of the input signal, repositioning them to corresponding time indexes in the output signal, overlapping the windows to achieve continuity, and adding. WSOLA is unique among overlap add algorithms in that it maintains local Fourier similarity in a time-scaled fashion but more importantly, the excised segment is similar to the segment adjacent to the previously excised segment. This makes WSOLA a very robust time-scaling algorithm being able to time scale even in the presence of noise and even competing voices in the input speech signal.

5.2 The Algorithm

The first step is to window the input signal with a smooth window such as a hanning window. Let w(n) be the window. Then establish a time warp function τ(n) such that for n an index in the input signal τ(n) equals the time scaled index in the output signal.

The input signal should then be windowed such that each segment overlaps with half of the previous segment. Then copy the first windowed segment to the output signal. The first segment of the input should be copied to the first segment in the output without consideration for the time warp function. Call the location of the last copied segment in the input S_1. Now the algorithm needs to find the next segment which it will copy, overlap and add with the current output signal.

There are quite a few ways to find this next segment. The most obvious method is to simply copy the segment at $\tau^{-1}(S_2)$ to the segment at S_2 in the output. However, this would wreak havoc on the phase synchronicity of the signal. The second method is to copy phase synchronous segment such that overlapping and adding will not cause huge phase differences between the two segments. This would maintain Fourier characteristics but would sound very choppy at syllable change edges in the speech. The WSOLA algorithm looks for a segment near $\tau^{-1}(S_2)$ that is most similar to the segment at $\tau^{-1}(S_2)$ + length(w). The next signal must be near (within a threshold) of the index given by the time warp function but also similar in Fourier

[1] This content is available online at <http://cnx.org/content/m33240/1.1/>.

characteristics to the next segment in the input signal. In other words, it finds a segment near the time scaled index such that it is very similar to the next naturally occurring segment in the input signal.

There are many ways to define most similar. The easiest way is Euclidian distance between the two signals. But computing the distance between the segments is computationally very expensive $O(N^2)$. The cross-correlation between the next adjacent segment and the region of interest seems to be the next alternative. The normalized cross-correlation if computed in the time domain is equally expensive. However, in the frequency domain, the computation is rather fast $O(N\log N)$. The peak of the normalized cross-correlation occurs at the point of highest similarity. The segment corresponding to this point is then taken as the next segment and copied over to the output signal, overlapped with the existing signal and added. This is done iteratively until the entire output signal is created.

This algorithm can take arbitrary time-warp functions and can time-scale a signal while reliably maintaining Fourier characteristics. It is also less computationally expensive than simple up sampling or down sampling for non-integer scaling factors. In fact, sampling rate modification cannot be easily done for irrational scaling factors but this algorithm will even handle that.

5.3 Implementation

This algorithm was implemented in matlab and achieved good results even with arbitrary constant time warp functions. Constants ranging from .1 to 10 were tested with satisfactory results.

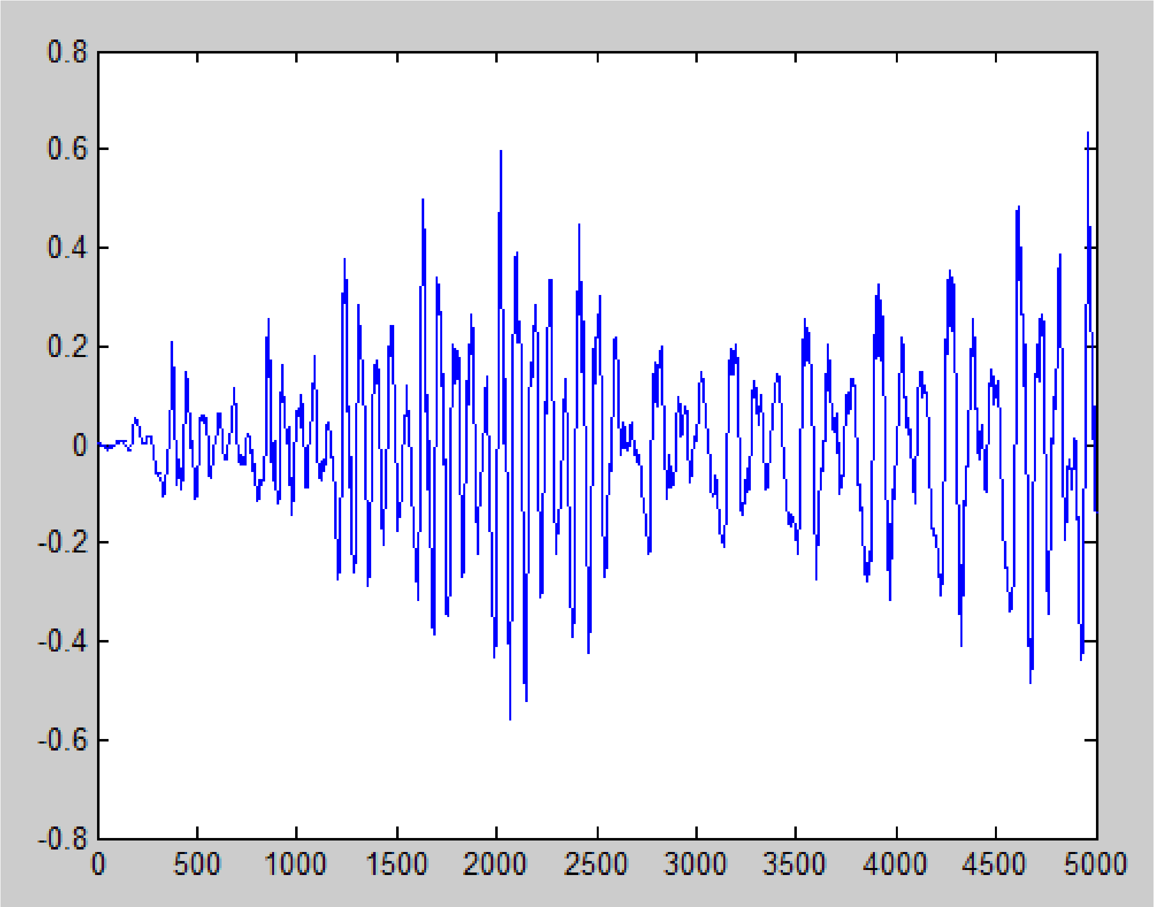


Figure 5.1: Short sample of a speech signal

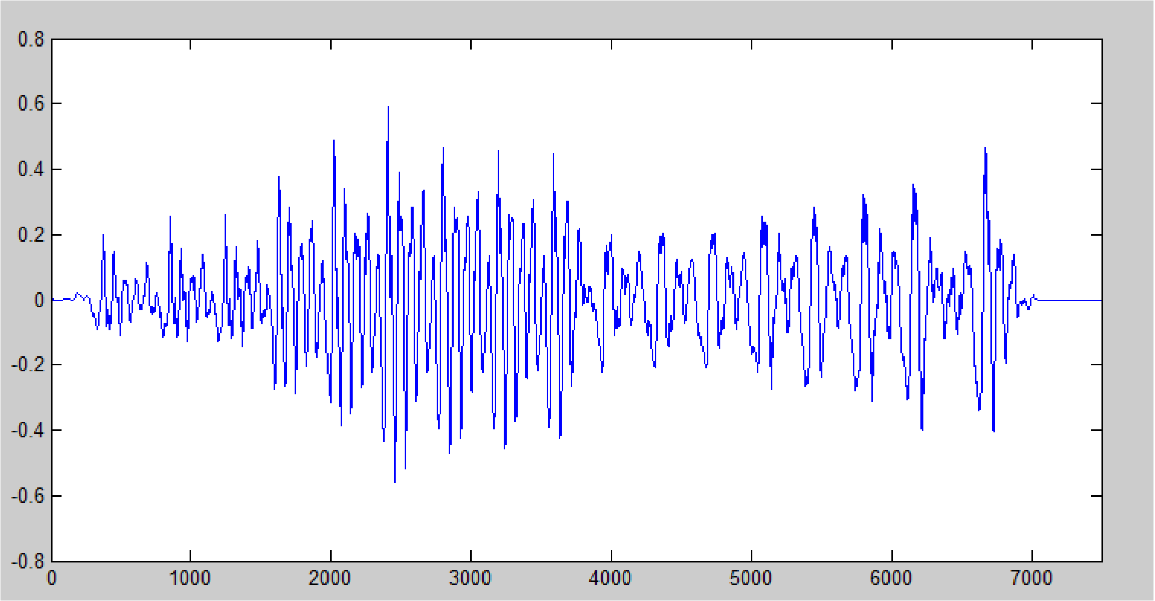


Figure 5.2: Time scaled version of previous speech signal

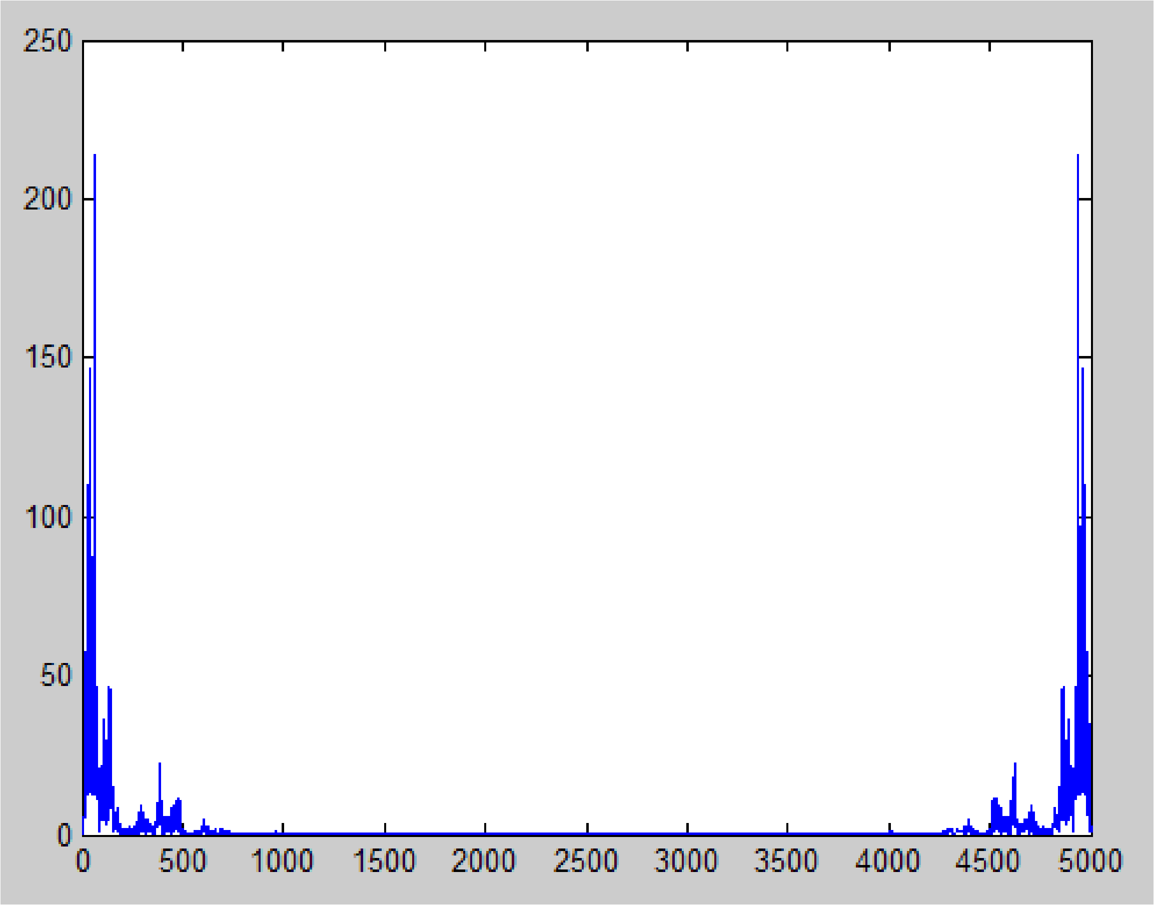


Figure 5.3: Fourier Transform of speech signal

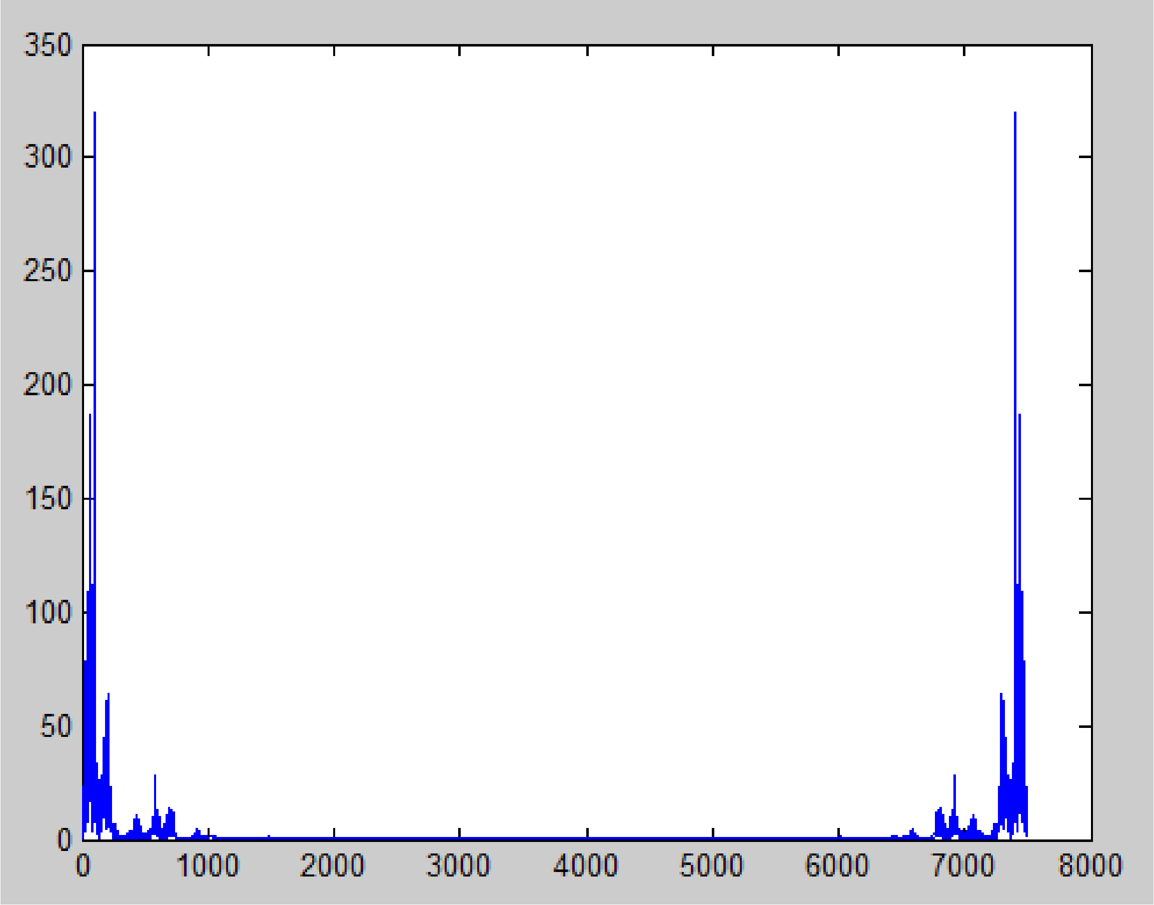


Figure 5.4: Fourier Transform of time scaled speech signal

Chapter 6

Speak and Sing - Pitch Correction with PSOLA[1]

6.1 Introduction

Pitch correction of the human voice is a common activity, with applications in music, entertainment, and law. It can be used to alter pitch to produce a more accurate or more pleasing tone in music, as well as add distortion effects. Several programs for entertainment use a form of pitch correction to modulate and distort a user's voice, allowing one to sound like a different gender or emulate a celebrity or other well-known voice. Voice distortion is also often required to protect the anonymity of individuals in the criminal justice system. However, it is the first of these applications that we are most interested in - producing a pleasing, tone-accurate song from a human voice.

Implementation

Pitch adjustment of a digitally-sampled audio file can be implemented simply using resampling. However, this completely alters the time scaling and cannot account for changes in the pitch and inflection of a voice over time, and thus cannot be considered. Instead, we shall use the more sophisticated Pitch-Synchronous Overlap Add algorithm, which allows us to modify pitch without compromised information or modifying the time scaling.

The pitch correction method involves the following basic steps:

- Detection of original pitch
- Parsing of desired pitch frequencies
- Correction of pitch

6.2 Pitch Detection

First, the pitch of the original signal is determined. This is done using the FAST-Autocorrelation algorithm. This algorithm makes use of the fact that for a signal to have pitch, it must have a somewhat periodic nature, even if it is not a strictly periodic wave. The signal is divided into several small windows, each only a few milliseconds long and containing thousands of samples - enough to detect at least two periods and thus to determine the window's frequency.

Finding periods

Each windowed segment is autocorrelated with itself to identify the length of the period. This is done by convolving the signal with itself with an increasing offset τ to obtain the autocorrelation function:

R(τ) = f(-τ) * f(τ)

[1] This content is available online at <http://cnx.org/content/m33242/1.1/>.

For discrete, finite-length signals, it can be found as a sum of the product of the signal and its offset, in this form:

R(s) = Sum(x(n)x(n-s))

This autocorrelation acts as a match filter: the signal and its offset form will be the most alike when offset s is equal to one period. Thus, the autocorrelation function is at a minimum when the offset corresponds to the length of one period, in samples.

Making it FAST

Autocorrelation in this fashion is very computationally expensive - one can expect that the algorithm will have to convolve two length-1000 signals several hundred times for each window to obtain the frequency from within the full possible range of frequencies for a human voice. To speed this up, we can make two assumptions:

1. The frequency of a window should be relatively close to that of the window before it
2. The first minimum corresponds to the period, so no further minima are needed

By starting at an offset relatively close to the previously found period length (perhaps 20 samples before where the period was found), we can eliminate a few hundred calculations per window. If a minimum is not found in this area, we simply broaden our range and try again. To reduce the computation time further, we also calculate the derivative dR(s)/ds to determine where the minimum occurs. Once we find the first minimum, we are finished with obtaining the frequency for this window, having shaved off up to 70% of our computation time.

When we're done...

Once a frequency has been found for every window, a vector of frequencies (one for each window) is compiled and returned to the pitch correction handler function.

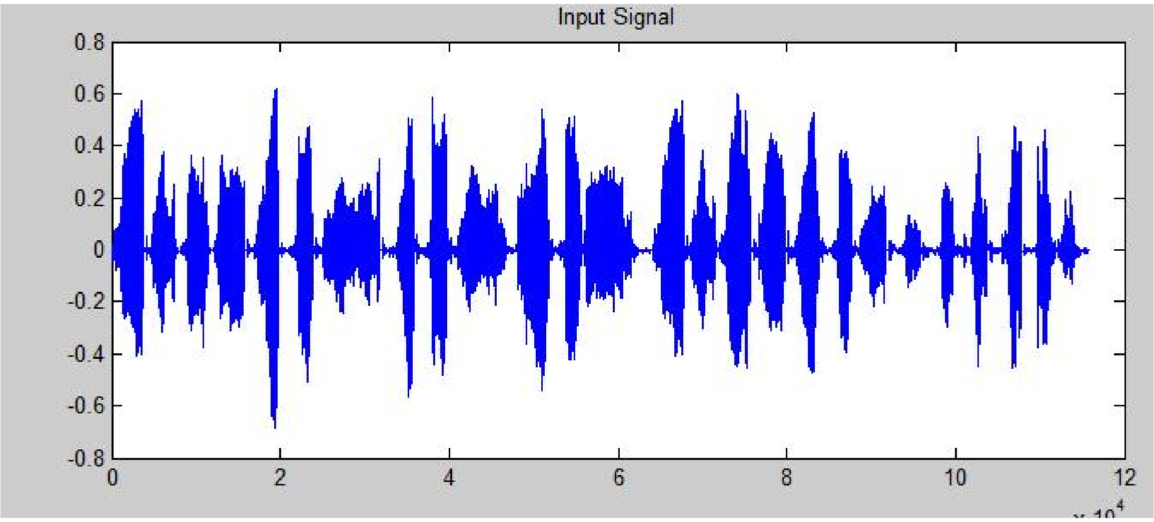


Figure 6.1: Waveform of an input audio signal (speech: "Mary had a little lamb...")

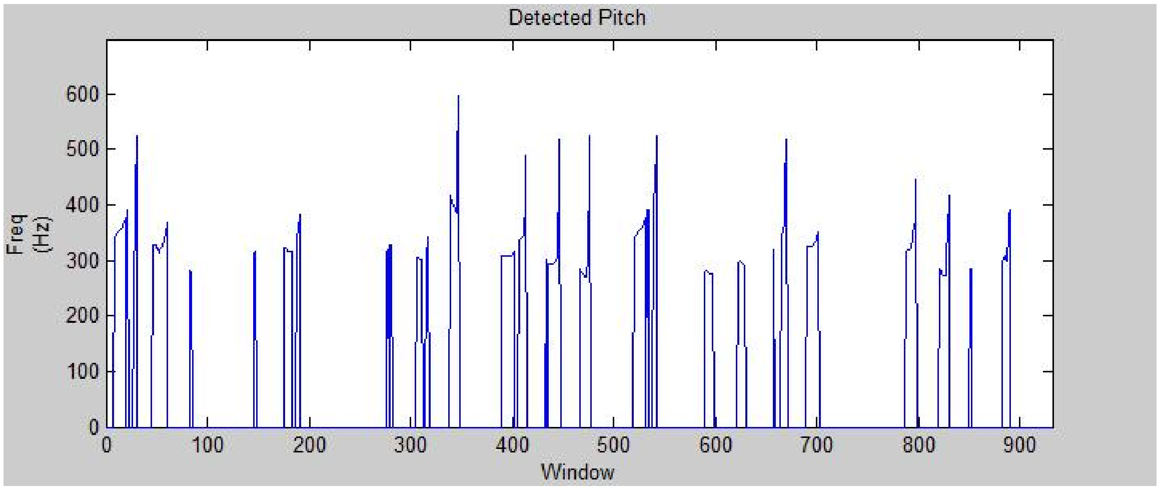


Figure 6.2: Detected frequencies of the signal above, one per window. Here it is easier to observe the spikes in frequency for parts of speech that may be spoken higher in pitch. If this input was sung rather than spoken, this plot would be much smoother and look closer to the desired frequency.

6.3 Desired pitch

The PSOLA pitch correction algorithm requires both an original pitch and a "target" pitch to achieve. If this were a fully-automated pitch-smoothing autotuner, the target pitch would be whatever "note" frequency was closest to the one observed. We on the other hand would like to bend the pitch to the specific frequency of the song, regardless of our starting point. To this end, we must generate a vector of desired frequencies.

Fortunately, thanks to our song interpretation earlier, we already have vectors of the pitch and length of each note in the song at hand. These vectors assume the following format:

- Frequencies: fundamental frequency in Hz (one per note)
- Durations: length in seconds independent of sampling frequency (one per note)

First, we generate a vector of frequencies for each sample at our defined sampling rate. This is as simple as producing a vector with a length equal to the total length of the song in seconds times the sampling frequency (thus, lengthN = sum(durations)*Fs). Then, for each note, we copy the frequency of that note over every sample in the vector for a range of the note's duration. This is most easily done using MatLab's "cumsum" function on the durations vector to make each note indexed by the cumulative time passed, and then multiply these by the sampling frequency to produce the index of each note in samples.

Now that we have the frequency for every sample, we can chop up this full-length signal into windows just as we did to the input signal. For each window's range, we simply take the mode of the frequencies in that range (given their short length, a window will never span more than two notes) and let that be the desired frequency for that window.

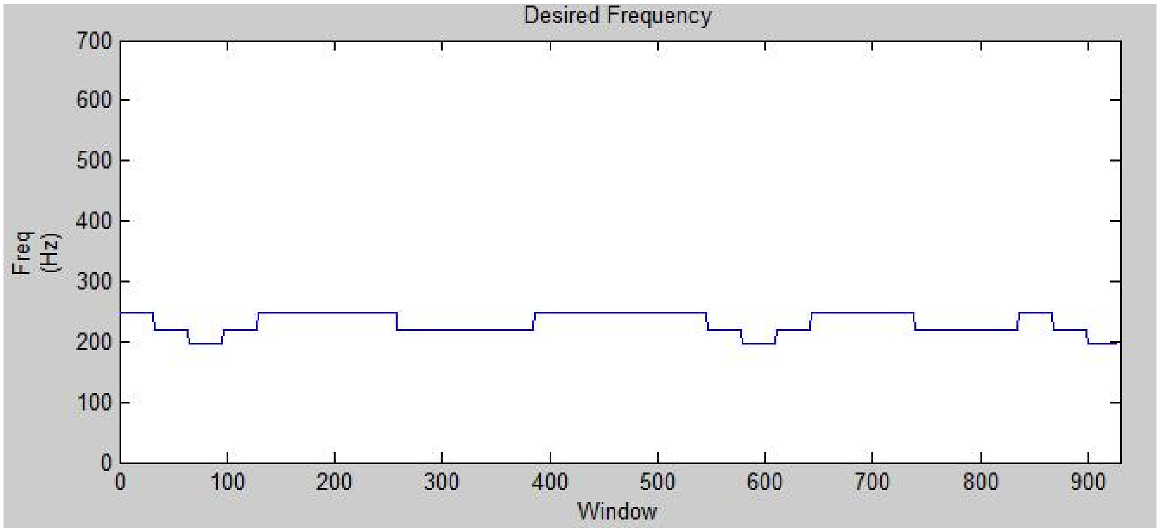


Figure 6.3: A plot of the desired frequency-per-window of "Mary Had a Little Lamb". The high and low notes are very clearly distinguishable.

6.4 PSOLA

Now that we have our original and target frequencies, we can exercise the Pitch-Synchronous Overlap Add algorithm to attempt to correct the frequencies. Like autocorrelation, the PSOLA begins with a windowed, segmented signal. Because we have already determined pitches for a specific number of segments, the PSOLA computations will use the same segment length. This is easy to remember, but introduces some issues. For example, the PSOLA algorithm can make the finest pitch corrections with a greater number of smaller segments, allowing for smoother correction across the signal. But what would happen to the autocorrelation pitch detector if the segment was so small that a full period could not be obtained? A compromise must be made on a segment length which allows for optimal pitch detection and pitch correction, with guesswork as the only means of finding the "happy medium".

Modifying pitch with Hanning windows

The signal we input to the PSOLA algorithm is already "windowed" into several overlapping segments. For each segment, the PSOLA creates Hanning windows (windows with a centralized hump-shaped distribution) centralized around the pitch-marks, or spikes in the amplitude. Once the segment is divided into overlapping windows, these windowed areas can be artificially pushed closer together for a shorter, higher-pitched signal, or farther apart for a longer, lower-pitched signal. The jumps between the beginning of each window is shortened or lengthened, and segments are duplicated or omitted where necessary. Unlike resampling, this change of pitch and duration does not compromise the underlying information.

Smoothing it out with Overlap and Add

Once the pitch and duration of the signal have been adjusted, the segments are then recombined by overlapping and adding. This Overlap-Add method exploits the knowledge that a long discrete convolution can be simplified as the sum of several short convolutions, which is convenient for us since we already have a number of short segments. The Overlap-Add produces a signal which is the same duration as its input and has roughly the same spectrum as the input, but now contains bands of frequency close to our desired frequency and, when played back, shows the result of our desired pitch correction effect.

The PSOLA algorithm described here is the Time-Domain PSOLA. Alternative PSOLA methods exist which depend on linear predictor coefficients rather than segmented waves. The TD-PSOLA is used for its *simplicity in programming versus marginal increase in computational cost.*

6.5 References

Gareth Middleton, "Pitch Detection Algorithms," Connexions, December 17, 2003, http://cnx.org/content/m11714/1.2/[2]

Lemmetty, Sami. Review of Speech Synthesis Technology. (Master's Thesis: Helsinki University of Technology) March 1999. http://www.acoustics.hut.fi/~slemmett/dippa/thesis.pdf[3]

Upperman, Gina. "Changing Pitch with PSOLA for Voice Conversion." Connexions. December 17, 2004. http://cnx.org/content/m12474/1.3/[4]

[2] "Pitch Detection Algorithms" <http://cnx.org/content/m11714/1.2/>

[3] http://www.acoustics.hut.fi/~slemmett/dippa/thesis.pdf

[4] "Changing Pitch with PSOLA for Voice Conversion" <http://cnx.org/content/m12474/1.3/>

Chapter 7

Speak and Sing - Conclusion[1]

7.1 Results

After the signal has been processed by the various functions, we obtain a resultant signal (in the form of a data vector) which has been time-scaled and pitch-corrected. The resulting audio playback is (presumably) on-beat, in time with the song, and of the correct pitch. The individual results of each step's implementation are described individually below.

Syllable Detection

The method of detecting sound types by energy and periodicity proved highly effective, with a decent rate of accuracy. The syllable identification by patterns seems to cover all cases effectively (assuming sound type detection worked). The input signal may need to be "doctored" a bit to remove the DC offset, amplify the signal, and remove excessive noise caused by noisy environments or electronic interference.

Time Scaling

The WSOLA algorithm works very well for duration scaling. It is able to shorten or expand syllables dramatically without any discernible loss of information. Tests indicate that the signal could be stretched to ten times its original length without audio artifacting occurring. Assuming the syllable detection function was accurate, the time scaling function produces a signal timed exactly to the song.

Pitch Correction

The PSOLA algorithm works as designed and introduces pitch correction. Given a relatively pitch-accurate input signal (such as a song or a sine wave), it will correct the input to the desired frequency. However, attempting to correct a dramatically different pitch (such as correcting the low timbre of a male voice speaking to a middle true C-note) causes a discernible gap between frequencies when listening to the signal. The result does not truly bend the pitch of the signal, but rather introduces harmonized distortion (hereafter dubbed "the T-Pain effect[2] ").

7.2 Limitations and areas for improvement

Song Interpretation

- No allowance is made for note slurs or variations within a syllable. Sustained words which vary in pitch are not currently supported. This would be relatively simple to implement by modifying the song vectors so that each syllable can contain multiple notes and durations.
- Songs must currently be coded by hand by examining sheet music or "playing by ear". For this reason, song selection is limited to how many man-hours are put into song coding. In the future, a MIDI file

[1] This content is available online at <http://cnx.org/content/m33243/1.1/>.

[2] http://www.youtube.com/watch?v=R7yfISlGLNU

decoder could automate this task. A more advanced approach would be to develop a pitch detector which identifies the vocal component of the song and then detects pitch.

Syllable Detection

- A relatively "clean" pre-processed signal produces best results. It would be possible to include DC offset removal, amplification, and finer noise detection in the MatLab function itself rather than rely on an outside program.
- Soft consonants (L, R, Y, and others) do not produce the same contrasting energy and periodicity as hard consonants. Multiple syllables which are separated by a soft consonant which is not clearly enunciated or emphasized may be grouped together as one. Further research and a more robust understanding of this sound type should allow for changes which will improve detection.

Pitch Correction

- The PSOLA algorithm is well-suited for minor pitch corrections but cannot produce major pitch bends; attempts to do so result in the T-Pain effect. A more aggressive pitch correction method could produce tonal sound from any input but would compromise the sound information, leaving very little of the original speech intact.
- The FAST-autocorrelation and PSOLA algorithms must use the same length and number of windowed segments. This creates a trade-off: autocorrelation can catch higher pitches and detect more accurately when using a longer window length, while PSOLA is able to make finer adjustments when given a larger number of smaller windows. If the window length is too small, autocorrelation may not detect any periods and would return zero frequency for that window. If the window length is too large, shorter sounds would not be pitch-corrected.
- The repeated convolutions of the autocorrelation and PSOLA algorithms make this the most computationally expensive step in the process. Methods such as the FAST method of reducing the number of test cases have dramatically improved this time, but there is still room for improvement.

7.3 Potential Applications

The Speak and Sing is a robust package which offers functionality and techniques not found in conventional autotuners. This all-in-one program and derivatives thereof show potential for applications in:

- **Music:** the Speak and Sing could provide a multi-functional alternative in situations where pitch correction and autotuner distortion are desired. It can also provide tempo and timing corrections on a dynamic scale.
- **Entertainment:** the Speak and Sing would, at the very least, make for an interesting iPhone app of the same name.
- **Communication:** pitch correction and timescaling are important facets of voice synthesis and could be used to augment human-interface and accessibility programs.
- **Speech analysis:** the syllable detection algorithm can be used to parse recordings and perhaps find use in speech-to-text applications.

7.4 In conclusion...

The Speak and Sing has served as an excellent demonstration of core digital signal processing techniques. Its development has served as a great learning experience for the team and has allowed each of us to flex our creative muscle.

While the "spoken words to full song" concept was not fully realized within this limited framework, the Speak and Sing is nonethelesss a functional, robust, and impressive program. It executes its syllable detection, time scaling, and pitch correction components correctly and produces an audible, tangible result from the input speech.

Index of Keywords and Terms

Keywords are listed by the section with that keyword (page numbers are in parentheses). Keywords do not necessarily appear in the text of the page. They are merely associated with that section. *Ex.* apples, § 1.1 (1) **Terms** are referenced by the page they appear on. *Ex.* apples, 1

Attributions

Collection: *Speak and Sing*
Edited by: Matt Szalkowski
URL: http://cnx.org/content/col11151/1.1/
License: http://creativecommons.org/licenses/by/3.0/

Module: "Speak and Sing - Introduction"
By: Graham Houser, Alysha Jeans, Sam Soundar, Matt Szalkowski
URL: http://cnx.org/content/m33229/1.1/
Pages: 1-2
Copyright: Graham Houser, Alysha Jeans, Sam Soundar, Matt Szalkowski
License: http://creativecommons.org/licenses/by/3.0/

Module: "Speak and Sing - Recording Procedure"
By: Graham Houser, Alysha Jeans, Sam Soundar, Matt Szalkowski
URL: http://cnx.org/content/m33233/1.1/
Page: 3
Copyright: Graham Houser, Alysha Jeans, Sam Soundar, Matt Szalkowski
License: http://creativecommons.org/licenses/by/3.0/

Module: "Speak and Sing - Song Interpretation"
By: Graham Houser, Matt Szalkowski, Alysha Jeans, Sam Soundar
URL: http://cnx.org/content/m33237/1.1/
Page: 5
Copyright: Graham Houser, Matt Szalkowski, Alysha Jeans, Sam Soundar
License: http://creativecommons.org/licenses/by/3.0/

Module: "Speak and Sing - Syllable Detection"
By: Graham Houser, Alysha Jeans, Sam Soundar, Matt Szalkowski
URL: http://cnx.org/content/m33241/1.1/
Pages: 7-13
Copyright: Graham Houser, Alysha Jeans, Sam Soundar, Matt Szalkowski
License: http://creativecommons.org/licenses/by/3.0/

Module: "Speak and Sing - Time Scaling with WSOLA"
By: Sam Soundar, Alysha Jeans, Graham Houser, Matt Szalkowski
URL: http://cnx.org/content/m33240/1.1/
Pages: 15-20
Copyright: Sam Soundar, Alysha Jeans, Graham Houser, Matt Szalkowski
License: http://creativecommons.org/licenses/by/3.0/

Module: "Speak and Sing - Pitch Correction with PSOLA"
By: Graham Houser, Alysha Jeans, Sam Soundar, Matt Szalkowski
URL: http://cnx.org/content/m33242/1.1/
Pages: 21-25
Copyright: Graham Houser, Alysha Jeans, Sam Soundar, Matt Szalkowski
License: http://creativecommons.org/licenses/by/3.0/

Module: "Speak and Sing - Conclusion"
By: Graham Houser, Alysha Jeans, Sam Soundar, Matt Szalkowski
URL: http://cnx.org/content/m33243/1.1/
Pages: 27-28
Copyright: Graham Houser, Alysha Jeans, Sam Soundar, Matt Szalkowski
License: http://creativecommons.org/licenses/by/3.0/

Speak and Sing
Documentation for the Speak and Sing, the smarter autotuner (an ELEC 301 semester project).