Speech Morphing for Space Marines

18–551 Spring 2001
Group 1 Final Report
Kirstin Connors
Dan Hook
Catherine Lyons

In this age of Hollywood invading every aspect of the American daily life, the consumer population is becoming more and more vain about how they look, who they know, and what people sound like. Actors are paid millions of dollars to do voiceovers for advertisements, dolls talk to our kids, and the phone companies still call to ask us if we want to change our long distance carrier. Much of these commercial and "toy" technologies are making their way online, leading to speech encoding becoming more and more important in industry. Speech Transformation or "Morphing" is a part of this increase in popularity allowing average people to sound like the Hollywood ideal. With this technology, people can make themselves sound like a tough Space Marine, a vengeful Darth Vader, or a sultry temptress, to name a few examples. Morphing has the potential to make a lot of money for a lot of companies.

"Speech morphing is the process of transforming one voice into another." If you have done any research into the topic of speech morphing, you've probably read this same statement, give or take an article or two, about ten times by now. What exactly does this mean within the context of this project? Speech can be characterized by two factors, linear prediction coefficients and pitch. When these two characteristics are calculated for each voice, we can then alter these factors in a given speech sample to resemble those for the other voice.

The basic idea is as follows. Two people give training samples and following this the computer will be able to transform the voice and whatever the first person says into the voice of the second. This undertaking is not dependant on the words that are said or the language that is spoken. It is dependant on the qualities of each individual's unique voice.

LPCs

Linear prediction coding is a way to represent speech that is often used for compression, but the way it represents speech is also useful for voice recognition and

transformation. As the name suggests, it involves predicting the next speech sample from previous samples. Linear prediction coefficients (LPCs) found by this process can then used to reconstruct the speech signal. The speech signal, s, is resynthesized using the following equation where a(i) are the LPCs, e is the error in the resynthesis process, and the summation goes from 0 to the number of coefficients.

$$s(n) = -\Sigma a(i)s(n-i) + e(n)$$

The equation can be rewritten as a linear filter, $E(z) = H(z)S(z)$, where E is the Z transform of the error, S is the Z transform of the signal, and H(z) is a filter defined by the LPCs. LPCs can be calculated several ways, which involve minimizing the error using either of these two equations.

Before we compute the LPCs however, we have to break our signal into small pieces to be processed individually. We split the signal into pieces 256 samples long, with an overlap of 64 samples. The overlap was necessary because speech as encoded by LPCs depends on previous values of the signal. It is continuous and overlaps prevent discontinuity. We also need a windowing function (in this case a Blackman window) to smooth out the signal so that there would not be a sharp discontinuity at the beginning or end of a frame. This function calculates values and then multiplies them by the values in our frame. To speed this up, we calculated all 256 values that we would be multiplying by and hard coded them into our program.

The method we used for calculating LPCs once we had our small samples of speech is the Levinson-Durbin method. This method is efficient and produces a stable filter that will always allow for a valid output [9]. The first step is an autocorrelation of the signal, followed by a recursive algorithm that computes the gain as well as a given order of coefficients. In the process of calculating LPCs, the algorithm also calculates reflection, or parcor, coefficients. These represent the same information as the LPCs in a slightly different format, and they are normalized between –1 and 1, but LPCs must be computed for resynthesis.

As the number of coefficients increases, so does the accuracy of the filter produced. In our project we ended up computing 20 LPCs. The resynthesized speech was of a much higher quality than it would be with fewer coefficients. There was a slightly noticeable improvement in quality up to 25 coefficients, but speech was perfectly recognizable with 20 LPCs, and the improvement with 25 was not significant enough to make up for the time it would take to run our program on 25% more data.

Pitch Detection

The first part of pitch detection is voicing determination. If a frame is unvoiced, then it will be unnecessary to determine the pitch for that frame. Voicing determination is by no means a trivial task. It is not easy to create a voicing determination algorithm (VDA) with a high degree of accuracy, and even a small percentage of inaccuracies can result in poor speech synthesis [1]. We use cepstral coefficients to determine if a frame is voiced or not. The cepstral coefficients can be considered as follows. H(z) is the model of the vocal tract, then we have

$$H(z)=\exp\left(\sum_{m=1}^{M} c(m)z^{-m}\right)$$

where M is the order and c(m) are the cepstral coefficients. For our purposes, M=27. We pad the coefficients with zeroes up to N where N is the length (512) of the frame we are looking at, and then perform an N point FFT. Call the result g. This gives us the magnitude of the log spectrum[3], from which we can evaluate a voicing index, v,

$$v=1/(14n)\left(\sum_{k=4n}^{17n} g(k)\right)$$

where n = N/256.
If v is above a threshold, 6.0 in our case, then the frame is classified as voiced; otherwise it is classified as unvoiced. There is no provision for detecting silence.

The method we use to calculate these coefficients is called unbiased estimation of the log spectrum (UELS). A brief overview of this method, and a method that may be superior in speed of performance but not in accuracy, can be found in [2]. A better explanation, referenced in [3], might be found in [4], but unfortunately a copy could not

be obtained. This is by far the most time consuming portion of our code. We had hoped to at least perform the FFT portions of the code on the EVM, but in order to do this, we needed to make the pitch code work with the EVM, which required it to run under Windows. We were not successful in porting the Unix C code to windows accept under Cygwin[5] which provides a Unix like shell in Windows.

Once we have made the unvoiced/voiced decision we need to extract the pitch. This is a comparatively simple operation. We use cepstral pitch detection. A form of cepstral pitch detection performed well in [6], and it was available in the SPTK[3] source code. Another option we considered, used by Group 18 in 1999, was the Gold-Rabiner parallel pitch detection algorithm[7]. The main advantage of using a time domain pitch detection algorithm is that the precise times when a pitch period begins and ends can be determined. As this was unimportant or our application, and we already had source code available, we used cepstral pitch detection.

The cepstrum is the inverse FFT of the natural log of the magnitude squared of the FFT of the input. If the frame is voiced, there will be a peak in the cepstrum at the frequency representing the pitch of the frame, and smaller peaks at the harmonics. The peak can then be detected by peak detection logic. We have a bound between 50 and 240 Hz. Anything outside this range is rare.

We considered a 512 sample section of speech. This corresponds to 46.4 ms of speech at our sampling rate of 11025 kHz. For pitch detection to work, it is necessary that the frame contain at least two pitch periods. Two pitch periods of 50 Hz will fit within the 46.4 ms window.

Overall, we were happy with the performance of the pitch detection algorithm. Resynthesized speech sounded very natural. Group 18 reported having problems with pitch. They blamed this on the Department of Defense glottal pulse function, but it could just as easily have been pitch detection or voiced/unvoiced distinction. The accuracy of the voice detection algorithm and the pitch detection algorithm in our project has allowed

us to eliminate these as reasons why the statistical method of morphing speech does not work as well as could be hoped.

Morphing

A person's voice can be modeled as air passing through your vocal chords, which give it a certain pitch, then passing through the rest of the vocal tract, which filters the signal to produce specific speech sounds. Speech is then either a periodic sequence of air pulses passing through the vocal tract, which produces voiced speech with a period equal to the period of the pulses, or is a random non-periodic turbulent burst of air passing through the vocal tract, which produces unvoiced speech. Due to individual differences in physiology, different people have distinct characteristics that define their speech. If the filter produced by a person's vocal tract can be accurately represented, then it can also be altered to represent another person's voice.

LPCs are one way to model this filter. They can be used to linearly construct speech given an input signal of a certain pitch, such as the compression wave coming from the vocal chords. The LPCs therefore can be looked at as defining both the speaker and the sounds. This makes them useful for both speaker identification and speech recognition programs, which have made use of LPCs with some degree of success. For the same reasons, they are potentially useful for speech morphing. If we can transform the LPCs in a speech sample, keeping the information about what the person is saying but altering the LPCs slightly to take on the characteristics of another person's voice, then we have a form of morphed speech. This form of morphing is limited, however, in that there are other qualities defining how a person speaks, besides their pitch and the shape and qualities of their vocal tract – for example, the intonations in their speech. Another drawback is that speech representation using LPCs is not entirely accurate. The all pole model provided by LPCs does not capture all qualities of the vocal tract and a pitch value does not capture all of the information about the excitation of the vocal tract.

We found that PARCOR coefficients worked well with morphing, whereas LPCs did not, although they contain the same information. This may be because every LPC coefficient is involved in the calculation of every other LPC coefficient. This is not true for PARCOR coefficients. In that case, only the previous PARCOR coefficients will affect the later ones. Distortion of one LPC coefficient will therefore cause a greater change then distortion of one PARCOR coefficient. Since the morphing process is not perfect, some distortion is to be expected, so the PARCOR coefficients are the better choice.

In order to morph the PARCOR coefficients, we need speech samples of the voices we are morphing between. Lets say we want to morph person A's voice to person B's voice – then we start with a speech sample from each person. For the two speech samples, we then determine the probability distribution of the parcor coefficients. Based on the experience of the previous group, we used a CDF for our probability distribution. In the morphing step, we input a new speech sample from person A. We calculate the parcor coefficients for the new sample, and find the CDF (lets call it x) of a coefficient, in the distribution we calculated for the original sample of person A's voice. We take the CDF value (x) that we just found and look at person B's original speech sample. Our new coefficients will be the coefficients from the speech sample of person B that have the same CDF x. Since our output LPCs come from person B's own speech, the output ought to sound like it is coming from person B, however because we mapped the CDF it also sounds like what person A was saying.

To compute the CDF, we separated each coefficient, 1-20, into separate vectors and simply sorted them. Because we were saving all of the parcor values, the CDF of each was just a fraction of its position in the sort. Therefore, we wrote a program to take in all of the parcor coefficients in a given speech sample, separate them by order, and run quicksort to sort the coefficients for each order. This was run on each of our training files. Then we wrote a program to perform the actual mapping. It takes in two sets of sorted coefficients from the training files, and a third set of coefficients to be morphed. For each coefficient to be morphed, it searches the correct order in the first sorted training

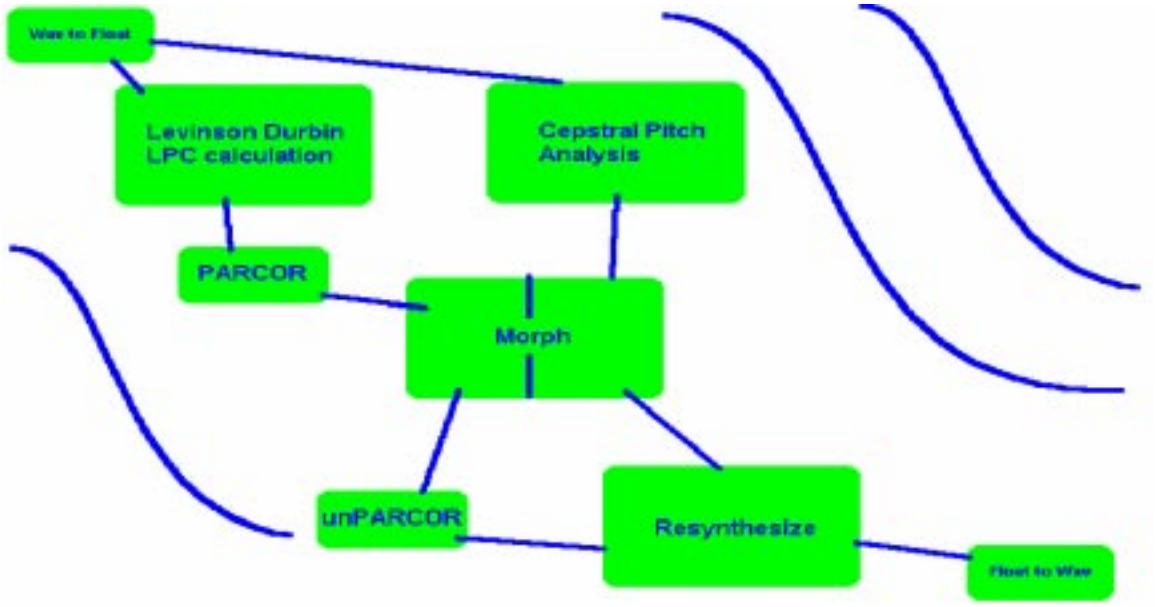
set for the closest match to the input coefficient. When the program finds the value in the sorted data set, it takes its position in the array, and normalizes by dividing that number by the number of samples in the first sorted data to get the CDF. Then it multiplies the CDF by the number of samples in the second data set. It outputs the parcor value in that position in the second data set. Finally, to speed up our algorithm, we changed the code to use a binary search.

We also morphed pitch using the same process, although implementation was easier with only one set of values for pitch (as opposed to 20 sets of coefficients). This allowed us to not only raise or lower the pitch to correspond to our target speaker's average pitch, but also to allow for people who have a wider or narrower range.

Data flow

We used wav files as a means of input and output for our sound files. Our program consisted of two steps, a training step, run for each voice, in which the computer prepares the information about one of the voices it is morphing, and a morphing step in which a new speech sample is given and the computer maps the input voice to the target voice. For the training step, we must first convert our wav file to lists of floats using SoX. Then we compute the parcor coefficients and pitch. The parcor coefficients are computed as part of the process of computing LPCs, however, because we had to convert them back and forth in order to resynthesize speech, it became easier to use a conversion program we found in the signal processing toolkit. Finally, we separate the coefficients by order and sort them, and also sort the pitch values.

For the morphing step, we convert a new input file to floats, and then calculate the pitch and coefficients. These two aspects of the speech signal are processed seperately; each is morphed using its own sets of sorted files. The new pitch is used to create an excitation signal, filtered using the LPCs, to create a new voice signal. This output signal is converted back to wav format.


Wav to Float
Levinson Durbin
LPC calculation
Cepstral Pitch
Analysis
PARCOR
Morph
unPARCOR
Resynthesize
Float to Wav

Formats

The primary means of recoding data that we used was the wav file. This format is a direct representation of the signal which is not compressed. It consists of a header determining what the sampling rate is, whether the file is stereo or mono, the number of channels, the number of bits per sample, and other header information that is more computer specific. A good description of wav files can be found at:

http://www.intersrv.com/~dcross/wavio.html.

We ended up using Sound eXchange (SoX) to translate between the wav file and a raw list of signed floats and vice versa.

How we approached this project

The first step that we successfully accomplished was translating the wav file into a list of ints. We did this using the C++ classes we found at the wavio.html site above. However, translating this code to one that would output a binary list of floats instead of an ASCII list of ints proved to be more challenging than we had originally anticipated. So, this code was abandoned in lieu of the more usable and applicable SoX program.

At the same time we researched possibilities for encoding speech. We learned that we would have to find a way to represent both the sound of a person's voice and the pitch of it. We started investigating possibilities like Code Excited Linear Prediction (CELP) and Levinson-Durbin Linear Predictive Coding. While, CELP seemed like an exciting new way to represent speech that had not been done before, it came into question whether this method of linear prediction would apply to morphing. We could not find any paper or mention of CELP being used for voice transformation. What makes CELP a better speech encoder for compression purposes is that it preserves more information about the speech signal than LPCs, but we wanted to morph the LPCs because they could model the vocal tract as described above. The additional information in CELP makes for

a better resynthesized voice, but it was not necessarily the type of information we wanted to morph. Though references to voice transformation are few and far between, either LPCs or reflection coefficients seemed to be the most promising methods to accomplish our goal, and the Levinson-Durbin algorithm seemed a good way to compute them. In the Levinson-Durbin algorithm, reflection coefficients are explicitly referred to as one of the intermediate steps in calculating the LPCs. The '99 group makes mention of calculating the reflection coefficients, but it is unclear whether they actually morphed the reflection coefficients or the LPCs. We found a Speech Processing Toolkit (SPTK) in the internet at:

http://kt-lab.ics.nitech.ac.jp/~tokuda/SPTK/

This contained the algorithm for Levinson-Durbin LPCs.

Pitch is something of great importance to the representation of speech. The previous 1999 group used a method called Gold-Rabiner. Doing a bit more research into the Gold-Rabiner algorithm, we found another pitch determination algorithm that was more accurate, if more time consuming. Cepstral pitch detection and UELS voicing determination showed much promise and had not been done before in 18-551.

So, we set to work. Using the SPTK as a guide, we endeavored to make the LPC and pitch code work on the EVM. This step spanned into weeks and the project fell further and further behind schedule. Morphing began to be a concern as neither pitch or LPCs would run successfully on the EVM. We considered the morphing question and started implementing it using data strictly from the SPTK programs which worked exclusively on unix.

Using sorts and direct mapping of cumulative density functions, the morph *should* have worked, but it still was producing pops and squeaks and fuzz that sounded like the pattern of speech we had fed into the algorithm to morph to. The fuzz indicated to us that that the pitch morph was in fact working, but the LPCs, which contained the bulk of the

information about each word, did not function yet. We didn't understand the problems we were having making the code run on Windows. Finding Cygwin allowed us to successfully run both the Pitch detection and the LPC calculation under Windows. Cygwin runs a Unix shell under the Windows operating system.

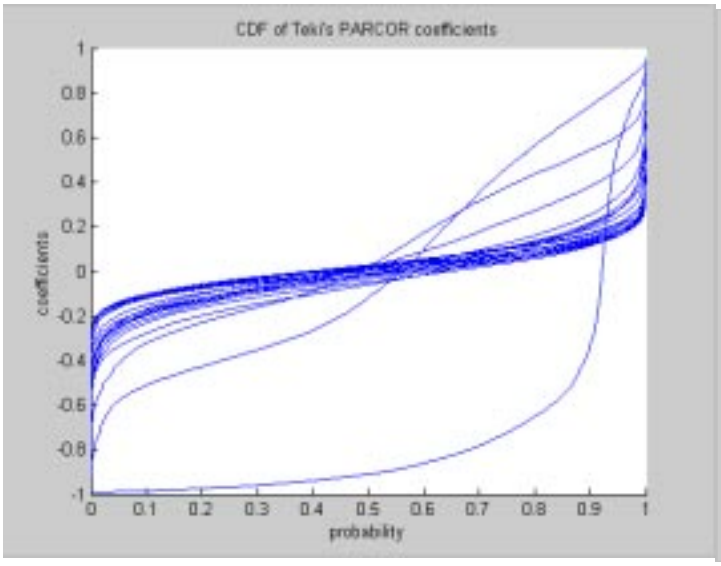
Despite the moderate success in calculating both pitch and LPCs, the morph still wasn't working. We reconsidered the problem and went back to the LPC algorithm theory. We searched for more references about LPCs. Thanks to Joseph Picone of http://www.isip.msstate.edu/, we found a much clearer description what was involved in the Levinson-Durbin algorithm, but more importantly we were reminded about the existence of reflection coefficients. In trying to get the LPCs to work, our earlier plan of trying to morph the reflection coefficients had been left aside. We found that there was a function in the SPTK specifically used for switching between LPCs and reflection coefficients. The SPTK, as well as many other sources, refer to the reflection coefficients as PARCOR coefficients. Rather than use even more memory and output the reflection coefficients from the LPC function, we decided to use the SPTK function and use a little more time to recalculate them. Almost miraculously, the morph worked and there was much rejoicing.

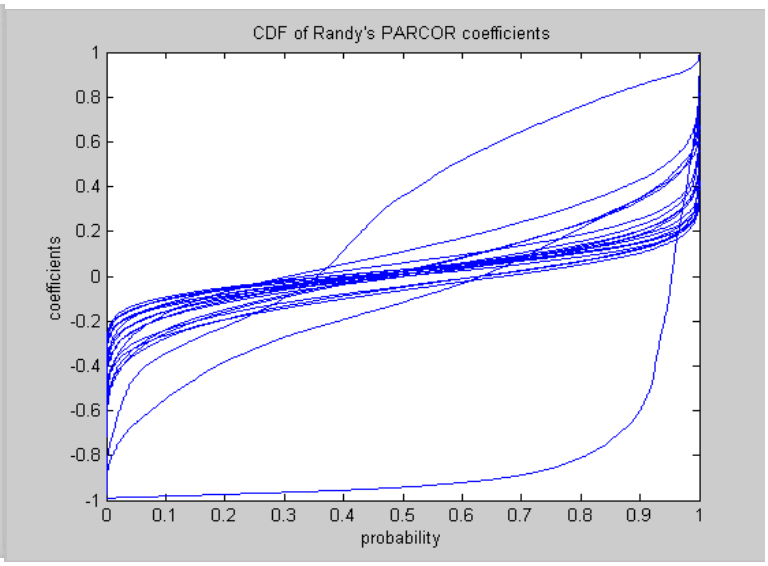
The final step was testing the morph to see the quality of the algorithm. We invited quite a few people to read. They each chose their own reading and the morphing worked, somewhat. It was a very temperamental thing. For some combinations of people, the morph was very good, however on others, it didn't seem to work at all. We tried to establish some sort of rule for when it worked and when it didn't, but there was no evidence pointing at anything mathematically speaking. Subjectively, we could guess at possibilities like if the person's training same was read with feeling, lots of pauses and jumps in pitch and emphasis, it was less likely to work. Perhaps, if the person had a very distinct voice, it was an easier target voice than a source voice. But it was all conjecture; we didn't have enough samples to wager on anything with any confidence.

Results

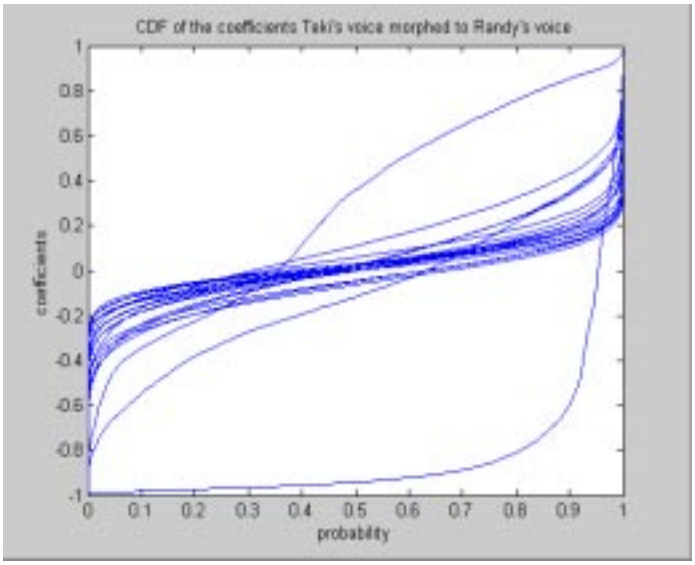
Our morphed speech was always clear and easy to understand, a considerable improvement over the previous project. This may have been due to a better pitch detection algorithm and to using more coefficients for our LPCs. However, the result did not always sound like the target speaker. It was certainly different from the source speaker, but not always different enough. Sometimes it would sound more like a combination of the two voices. We believe that we implemented everything correctly, but that speech morphing will not work for all kinds of input speech.

We made graphs in matlab of the CDFs of the coefficients in our training files and output files. We found that the speech our program produced did in fact have PARCOR coefficients very similar to the training file for our target speaker, as expected since we were taking the same coefficients from that file. They are slightly different because the speaker is now saying something different and so some coefficients will be used from the training file and some won't, changing the distribution. This shows us that our morphing worked as it was supposed to.





Data from original speech samples



Data from morphed speech

Our results sounded particularly good for two of our speakers, whose CDFs are shown above. They had similar intonations in their voices, but the target speaker's speech was very distinctive. One reason other morphs did not work as well may have been that intonation is a very important characteristic of people voices which cannot be morphed by this method. This is in part what allows people to do imitations of other voices. People speak in certain ways that can be recognized by distinct intonations, accents, pauses and emphases that have nothing to do with their average pitch or the characteristics of their vocal tract. Many of the speech samples we used to test our program were read with a great deal of feeling, which may have only emphasized the differences.

Another potential problem with our speech samples is that people paused frequently in their speech. These pauses were encoded and sorted into the training files just as the speech was. This may skew the sorted file slightly. Also, if there is background noise that can be picked up during these pauses it may affect the pitch. We did not include unvoiced speech in our pitch sorting or morphing (which could have skewed the results considerably), so for the most part the pauses in speech would have little effect if they were seen as unvoiced. But, if there was background noise loud enough to be heard when the speaker pauses, there is a potential for problems with pitch morphing. If pauses are a problem, they are difficult to deal with except by carefully selecting training files. One cannot simply remove or ignore the pauses in the speech because as mentioned earlier LPCs work for a continuous waveform.

[1] S. Ahmadi and A. S. Spanias,"Cepstrum-Based Pitch Detection Using a New Statistical V/UV Classification Algorithm," *IEEE Trans. on Speech and Audio Processing,* vol. 7, pp. 333-338, May 1999.

[2] K. Tokuda, T. Kobayashi, S. Imai,"Adaptive Cepstral Analysis of Speech", *IEEE Trans. on Speech and Audio Processing,* vol. 3, pp. 481-489, November 1995

[3] *Reference Manual for The Speech Processing Toolkit Ver. 2.0* http://kt-lab.ics.nitech.ac.jp/~tokuda/SPTK/ , currently unavailable so it is attached.

[4] S.Imai and C.Furuichi,"Unbiased estimator of log spectrum and its application to speech signal processing,"*Signal Processing IV:Theory and Applications*,Vol.1, pp.203 .206,Elsevire,North-Holland,1988.

[5] Cygwin Bash Shell, http://www.cygwin.com

[6] L. R. Rabiner, M. J. Cheng, A. E. Rosenberg, and C. A. McGonegal, "A comprehensive performance study of several pitch detection algorithms," *IEEE Trans. Acourst., Speech, Signal Processing,* vol. ASSP-24, pp. 399-417, Oct. 1976.

[7] B. Gold, L. Rabiner,"Parallel Processing Techniques for Estimating Pitch Periods of Speech in the Time Domain," *J. Acoust. Soc. Amer.*, vol. 46, Feb. 1969.

[8] W. Hess, *Pitch Determination of Speech Signals.* Berlin, Germany: Springer-Verlag, 1983

[9] J. Picone, "Signal Modeling Techniques In Speech Recognition," *Proceedings of the IEEE*, vol. 81, no. 9, pp. 1215-1247, September 1993. http://www.isip.msstate.edu/publications/journals/ieee_proceedings/1993/signal_modeling/paper_v2.pdf