TECHNIQUES FOR ANALYSIS/SYNTHESIS OF TIME
DOMAIN WAVEFORMS LEADING TO THE
IMPLEMENTATION OF A SINGING SYNTHESIS
ENGINE USING PHONETIC CONSTRUCTIONS

By
Trevor Monk

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
AT
DALHOUSSIE UNIVERSITY
HALIFAX, NOVA SCOTIA
AUGUST 1995

© Copyright by Trevor Monk, 1995
DALHOUSIE UNIVERSITY
DEPARTMENT OF
MATHEMATICS STATISTICS AND COMPUTING SCIENCE

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled "Techniques for Analysis/Synthesis of time domain waveforms leading to the implementation of a singing synthesis engine using phonetic constructions" by Trevor Monk in partial fulfillment of the requirements for the degree of Master of Science.

Dated: August 1995

Supervisor:  

Dr. Patrick Keast

Readers:

Dr. Sampalli Srinivas

Dr. Arthur E. Sedgwick
DALHOUSIE UNIVERSITY

Date: August 1995

Author: Trevor Monk

Title: Techniques for Analysis/Synthesis of time domain waveforms leading to the implementation of a singing synthesis engine using phonetic constructions

Department: Mathematics Statistics and Computing Science

Degree: M.Sc. Convocation: October Year: 1995

Permission is herewith granted to Dalhousie University to circulate and to have copied for non-commercial purposes, at its discretion, the above title upon the request of individuals or institutions.

__________________________
Signature of Author

THE AUTHOR RESERVES OTHER PUBLICATION RIGHTS, AND NEITHER THE THESIS NOR EXTENSIVE EXTRACTS FROM IT MAY BE PRINTED OR OTHERWISE REPRODUCED WITHOUT THE AUTHOR'S WRITTEN PERMISSION.

THE AUTHOR ATTESTS THAT PERMISSION HAS BEEN OBTAINED FOR THE USE OF ANY COPYRIGHTED MATERIAL APPEARING IN THIS THESIS (OTHER THAN BRIEF EXCERPTS REQUIRING ONLY PROPER ACKNOWLEDGEMENT IN SCHOLARLY WRITING) AND THAT ALL SUCH USE IS CLEARLY ACKNOWLEDGED.
Contents

List of Tables ........................................ vii
List of Figures ......................................... viii
Abstract .................................................. xii

1 Introduction .......................................... 1

2 Literature Review ..................................... 4

3 Fundamentals of Speech Synthesis .................. 9
   3.1 The Vocal Tract .................................... 9
       3.1.1 Places of Articulation ....................... 10
       3.1.2 Manner of Articulation ................... 11
   3.2 Connected Speech .................................. 13
       3.2.1 Phonetic transcription .................... 13
       3.2.2 Prosody/Suprasegmentals ............... 14
   3.3 Acoustic Phonetics ................................ 15
   3.4 Sampling ........................................ 17
       3.4.1 Linear versus Logarithmic quantization .. 19
       3.4.2 Storage Capacity and Cost considerations . 20
   3.5 Spectrographic Analysis and Sound Quality .... 21
4 Time Domain Pitch Estimation

4.1 Gold Rabiner algorithm ........................................ 24
  4.1.1 Filtering .................................................. 25
  4.1.2 Measures of peakedness .................................. 27
  4.1.3 Parallel Pitch Period Estimates ......................... 28
  4.1.4 Estimated Pitch Period Matrix ......................... 29
  4.1.5 Final Pitch Period Computation ....................... 30
4.2 Simple Inverse Filter Tracking - The SIFT algorithm ........ 33
  4.2.1 Pre-filtering ............................................. 34
  4.2.2 Down sampling ........................................... 34
  4.2.3 Differencing ............................................. 34
  4.2.4 Windowing ............................................... 35
  4.2.5 Determination of Inverse Filter Coefficients ........ 37
  4.2.6 Autocorrelation ......................................... 39

5 Pitch Modification: The PSOLA algorithm .................. 41

5.1 PSOLA synthesis ................................................. 43
5.2 PSOLA Implementation ........................................ 44
  5.2.1 Extracting a single pitch period ....................... 44
  5.2.2 Single Grain Synthesis ................................ 50
  5.2.3 Multi Grain Synthesis .................................. 51

6 Linear Predictive Coding (LPC) .............................. 53

6.0.4 Determination of Filter Coefficients .................... 55
6.1 Linear Predictive Synthesis ................................ 57
  6.1.1 Lattice Filters ......................................... 58

7 Implementation and System use ............................. 62

7.1 Hardware Platforms .......................................... 62
7.2 History and Overview of the System ....................... 62
7.3 Detailed System Description ............................... 64
7.3.1 The sample buffer ........................................... 64
7.3.2 The audio device ........................................... 65
7.3.3 PSOLA Analysis/Synthesis ................................. 68
7.3.4 LPC Analysis/Synthesis ................................. 71
7.3.5 The Musical Stave ........................................... 73
7.3.6 Entering a song ........................................... 75
7.3.7 Choosing and Recording Phonetic Elements ................. 76
7.3.8 The Synthesis Engine ........................................... 81

8 Conclusions and Future Development .......................... 83

A The Low Pass Filter ........................................... 85
A.1 Determination of the Filter Transfer function ................. 86
A.1.1 Stage 1 ........................................... 87
A.1.2 Stage 2 ........................................... 89
A.1.3 Plotting the Transfer function ........................................... 90

B Digital pre-emphasis ........................................... 93

C Lattice Filter Transfer function ........................................... 95

D The Sound Spectrograph ........................................... 104
D.1 The Fourier Transform ........................................... 104
D.1.1 Restrictions on the Fourier Transform ......................... 106
D.1.2 Graphing the frequency Spectrum ......................... 107
D.2 From the Fourier Transform to the Spectrograph ................. 107
D.2.1 Spectrograph examples: ........................................... 110

E Capture Scripts ........................................... 111

F Phonetic Symbols - The Arpabet ........................................... 113

Bibliography ........................................... 114
# List of Tables

3.1 Example of vowel sounds in relation to tongue position .......................... 13

4.1 Coincidence Window Widths (in samples) as a function of Pitch Period and Bias ................................................................. 31

4.2 Coincidence width as a percentage of pitch period as a function of bias 31

4.3 Example pitch period estimate table .................................................. 32

4.4 Final Coincidences for each estimator for each bias level .................. 32

A.1 Coefficients for 250, 600, and 900 Hz low pass filters ....................... 90

F.1 Phonetic symbols developed by US Dept. of Defence .......................... 113
# List of Figures

3.1 The Human Vocal Tract .................................................. 10  
3.2 Pressure Variation as a Function of Time .......................... 16  
3.3 Pitch period related to vocal oscillation .......................... 16  
3.4 Sampling a continuous time varying waveform ................... 18  
3.5 Transfer function from linear samples to logarithmic samples ($\mu$-law) .................................................. 19  
3.6 Frequency Spectrum for the vowel ‘ah’ as in ‘Father’ ............ 21  
3.7 Frequency Spectrum for the vowel ‘ee’ as in ‘see’ ................. 23  
3.8 Spectrographic plots of eight common vowel sounds sung in isolation .................................................. 23  

4.1 Block Diagram for the Gold Rabiner Pitch Tracking algorithm .... 25  
4.2 Unfiltered waveform ..................................................... 26  
4.3 Waveform filtered with 900 Hz low-pass filter ...................... 26  
4.4 Measures of Peakedness .................................................. 27  
4.5 Peak detection using variable blanking time and exponential decay .................................................. 28  
4.6 6x6 Matrix of Pitch Period Estimates ................................. 29  
4.7 Unwindowed sampled data ............................................. 35  
4.8 Sampled data windowed using a rectangular window ............... 35  
4.9 Waveform discontinuities caused by rectangular windowing .... 36  
4.10 The Hamming Window ................................................. 37  
4.11 Pre-processing stages of the SIFT algorithm ....................... 38  
4.12 Autocorrelation of a periodic time varying waveform .......... 40  

5.1 A typical section of voiced speech .................................... 42
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2</td>
<td>A single pitch period extracted from the waveform shown above</td>
<td>42</td>
</tr>
<tr>
<td>5.3</td>
<td>A PSOLA waveform with decreased pitch resynthesized using a rectangular window</td>
<td>43</td>
</tr>
<tr>
<td>5.4</td>
<td>Hanning window for extracting pitch periods</td>
<td>44</td>
</tr>
<tr>
<td>5.5</td>
<td>An original waveform with its low pass filtered (250 Hz cutoff) counterpart</td>
<td>46</td>
</tr>
<tr>
<td>5.6</td>
<td>Filtered and Unfiltered waveforms showing compensated peak positions</td>
<td>46</td>
</tr>
<tr>
<td>5.7</td>
<td>A PSOLA grain extracted using a rectangular window</td>
<td>48</td>
</tr>
<tr>
<td>5.8</td>
<td>PSOLA grain modulated by the Hanning window</td>
<td>48</td>
</tr>
<tr>
<td>5.9</td>
<td>Overlapping PSOLA grains represented by their Hanning windows</td>
<td>49</td>
</tr>
<tr>
<td>5.10</td>
<td>PSOLA synthesis from a single grain</td>
<td>50</td>
</tr>
<tr>
<td>5.11</td>
<td>PSOLA synthesis from multiple grains</td>
<td>51</td>
</tr>
<tr>
<td>5.12</td>
<td>Spectrogram of output of Multi grain PSOLA synthesis of eight common vowels.</td>
<td>52</td>
</tr>
<tr>
<td>6.1</td>
<td>Lattice Filter for Reproducing a Speech Wave from Reflection Coefficients</td>
<td>58</td>
</tr>
<tr>
<td>6.2</td>
<td>A Single Stage of the Lattice Filter</td>
<td>58</td>
</tr>
<tr>
<td>6.3</td>
<td>Four stage Lattice Filter as used in the SIFT pitch tracking algorithm</td>
<td>59</td>
</tr>
<tr>
<td>6.4</td>
<td>Spectrogram of LPC synthesis of eight common vowels</td>
<td>61</td>
</tr>
<tr>
<td>7.1</td>
<td>User interface to the sample buffer</td>
<td>64</td>
</tr>
<tr>
<td>7.2</td>
<td>User interface to the PSOLA algorithm</td>
<td>69</td>
</tr>
<tr>
<td>7.3</td>
<td>User interface to the LPC algorithm</td>
<td>71</td>
</tr>
<tr>
<td>7.4</td>
<td>The musical stave</td>
<td>74</td>
</tr>
<tr>
<td>7.5</td>
<td>The note editor interface</td>
<td>75</td>
</tr>
<tr>
<td>7.6</td>
<td>The phonetic data interface</td>
<td>78</td>
</tr>
<tr>
<td>7.7</td>
<td>The note synthesis engine</td>
<td>81</td>
</tr>
<tr>
<td>A.1</td>
<td>Ideal 900 Hz Low Pass Filter</td>
<td>85</td>
</tr>
<tr>
<td>A.2</td>
<td>Block diagram for the digital low pass filter</td>
<td>86</td>
</tr>
</tbody>
</table>
This thesis is dedicated to my parents for their support and encouragement over the years.
Abstract

Techniques for implementing a singing synthesizer are discussed. The major organs of
the human vocal tract are studied with reference to the manner in which we produce
speech sounds. The storage and analysis of these sounds are discussed with particular
emphasis on fundamental frequency estimation and spectrographic analysis. Two al-
gorithms for fundamental frequency estimation are implemented: Gold and Rabiner’s
algorithm, and the Simple Inverse Filter Algorithm (SIFT). The spectrographic anal-
ysis uses the Fourier Transform to visually display the frequency spectrum of the
human voice. The use of digital filters for preprocessing digital speech signals is also
examined in detail.

Two algorithms for synthesis of speech sounds are also implemented: Pitch Syn-
chronous Overlap Add (PSOLA) uses a method of granular synthesis — synthesis
of the human voice by combining small grains of sound extracted from real speech.
Alternatively, Linear Predictive Coding (LPC) predicts the next value of the speech
waveform based on several previous values. Used iteratively this procedure can recon-
struct an entire speech waveform using pitch, gain, and prediction coefficient values.
A stored library of phonetic elements encoded with either PSOLA or LPC can be
used to construct words to be sung in real time by the synthesiser.
Chapter 1

Introduction

For many years, speech analysis and synthesis have been a major area of research in Computer Science. Scientists have been analysing speech and its production for a much longer period. The study of speech in terms of its production is called Articulatory Phonetics, while the more specific study of the nature of speech sounds is called Acoustic Phonetics. Phonetics is a subject in its own right, it is not simply a tool for computer scientists attempting to model human speech production. A Phonetician studies how sounds are used to convey meaning in a language and also studies the mechanics of the sound production process. Chapter 3 describes the mechanical production of English speech.

The role of Computer Science in the analysis and synthesis of speech has primarily been to find ways to allow computers to interact with people in a more comfortable and useful way, that is, in a way we are more used to interacting with each other, by talking and listening. It seems to have long been a dream of the human race to make machines talk. This is indicated by a multitude of science fiction movies. Despite much research using computers and electronic machines over the past three decades, however, human quality speech production and recognition is still a dream. Science has certainly provided a (possibly) reasonable facsimile of the human voice, and computers can recognise a limited subset of spoken words. But fluent speech that is comfortable to listen to, and the ability to recognise a constant stream of speech is
still beyond present day technology.

For some, speech production and recognition is a gimmick. But for others, the physically handicapped for example, this kind of research is valuable and worthwhile. Perhaps it is for this reason that science has persisted in what has proved to be one of the most difficult applications of computers in society. Some may consider synthesis of the singing voice to be a gimmick also, and in some respects it is. But the differences between synthesis of the singing voice and speech synthesis is only minor, so minor in fact that the differences are hardly noted at all until Chapter 7.

Singing synthesis as opposed to speech synthesis was chosen in part for its novelty. Very little speech synthesis research has been applied to singing synthesis. Also, considering singing synthesis provides a simplified starting point for the general study of electronic speech synthesis, since the researcher need not be as concerned with the prosodic features of speech. Prosodic features, also discussed in Chapter 3, are features superimposed over a section of connected speech, such as pitch and rhythm. Chapter 7 discusses the application of prosodic features to singing synthesis.

As part of my undergraduate thesis I developed a system for automatic transcription of the singing voice for teaching music students to sing music by sight [22]. Basically this involves the extraction of pitch and rhythm from sung user input and display of this on a musical stave. While this was quite successful, a completely self contained Computer Aided Instruction (CAI) system would be capable of singing the correct version of the music presented to the student. This motivated the work discussed in this thesis for developing a singing synthesis engine.

The goal of this project, therefore, is to synthesize singing of an acceptable quality in real time, given notes presented on a musical stave. Phonetic text associated with each note is also provided, indicating the word to be sung with the pitch and duration specified by that note. To this end, this thesis investigates the application of currently available algorithms and techniques to the area of singing synthesis. In particular alternate representations of digital speech are investigated which will allow pitch and duration modification of recorded vocal input. These alternate representations are
known as source-filter models of speech. They separate a speech signal into a vocal oscillation component and some type of synthesis filter. This type of separation is vital for ease of pitch/duration modification.

Chapter 2 gives a review of the literature associated with vocal synthesis which is relevant to this thesis. Chapter 3 attempts to give the reader an introduction to the mechanisms of human speech production, as well as an introduction to the electronic representation and storage of sound. These are the fundamentals of electronic speech synthesis. Chapter 4 discusses two algorithms used by the system for frequency (pitch) estimation of vocal sounds. Accurate pitch estimation is often vital for high quality speech and singing synthesis, and both of the algorithms discussed are used by the algorithms in subsequent Chapters. Two synthesis methods are examined in Chapters 5 and 6. Chapter 5 discusses a method of granular synthesis known as Pitch Synchronous Overlap Add (PSOLA) which reconstructs speech or singing from small grains extracted from vocal input. Chapter 6 examines the more analytic approach of linear prediction which has applications in many of the applied sciences. Chapter 7 ties everything together by discussing the implementation of the system in terms of the components discussed in the previous Chapters.

Appendix A analyzes a digital low pass filter used throughout this project. Appendix B analyzes the transfer function for the digital preemphasis technique discussed in Chapter 4. Appendix C provides a similar analysis for the lattice filter used in Chapter 6. Appendix D discusses the construction of the sound spectrograph, a tool used for visibly displaying the frequency nature of speech. Appendix E describes the format of the capture scripts used to extract multiple phonetic elements from multiple data files.
Chapter 2

Literature Review

Speech Synthesis has long been an active area of research. Attempts at speech synthesis date back to the late 18th century. In 1779 Christian Kratzenstein constructed a set of acoustic resonators to model the sounds of the five vowels a, e, i, o, u. Each resonator consisted of a cavity terminated at one end by a vibrating reed. The reed is caused to vibrate by passing an air stream across it [11]. A more complicated machine consisting of bellows and a soft leather resonator tube was invented by Wolfgang von Kempelen [11]. The shape of the tube was manipulated with one hand while the bellows forced air past a reed and out through the tube. The shape of the hand and thus the shape of the tube determined the vowel sound. A whistle and controls affecting airflow through it allowed synthesis of ‘s’ and ‘sh’ sounds. Both Kratzenstein’s and Kempelen’s machines create sounds by attempting to model the human vocal tract. For a description of the vocal tract and its production of sounds see Chapter 3.

Electronic technology facilitated a somewhat different approach to studies of human speech mechanisms. In the late 1930’s Homer Dudley invented an electronic device for frequency analysis/synthesis of the human voice [5]. The analysis stage decomposed the speech signal into an excitation source (vocal chord oscillation) and a spectral envelope (describing the frequency spectrum of the signal). This stage was implemented as a bank of filters. Each filter determined the amplitude of the frequency components within its range. The synthesis stage reconstructed the original
signal by adding waveforms produced at the outputs of frequency generators covering frequency bands identical to those of the analysis filters. The inputs to the synthesis frequency generators are the outputs of the analysis filters. The device was named the VOCODER (VOIce CODER). Another device known as the VODER (Voice Operation DEmonstratoR) developed by Dudley was demonstrated at the Worlds Fairs of New York and San Francisco [25]. The device used the same technique as the VOCODER, but the analysis stage was removed and replaced by a trained human operator using a keypad controlling the output of the frequency generators [5, 6]. Dolson discusses the theory of the vocoder and the filter-bank representation in [4].

These types of Source-Filter models of speech have been studied extensively since Dudley's invention of the VOCODER. Frequency analysis of speech and indeed many other time varying waveforms have persisted to the present day. A device called a sound spectrograph developed at Bell Telephone Laboratories allows detailed study of the frequency components of a sound wave [8]. In fact it has often been termed "visible speech". Originally the spectrograph traced spectral intensities of the frequency bands under consideration on a rotating paper covered drum. These spectrograms are now calculated electronically using Fourier transform analysis. Refer to Appendix D for details. Throughout this thesis you will notice electronically generated spectrographs.

A mechanism for reconstructing speech signals from "visible speech" (spectrograph plots) has also been constructed. Shadow patterns, corresponding to spectral intensities, on a rotating drum are rotated past light/dark sensors to activate frequency generators. A picture of this machine demonstrated in 1951 is shown in [6]. A similar mechanism is described in [10].

Some electronic systems attempt to directly model the vocal tract as a series of cylindrical sections placed end to end. Dunn used this technique in the development of an electrical vocal tract at Bell Telephone Laboratories in the late 1940's [7].

With the introduction of more sophisticated electronic computers, the focus of synthesis systems moved from electrical devices to digital computer synthesis systems. Digital representation of the speech wave is discussed in Chapter 3. "Principles
of Computer Speech” [29] gives an excellent account of computer representation of acoustic signals.

Digital techniques for speech synthesis are based both on models of the vocal tract and direct manipulation of time varying waveforms. Dudley’s VOCODER is easily implemented by a digital computer using the Discrete Fourier Transform (DFT) to determine the vocal tract spectral transfer function. The DFT is the digital analogue of the filter-bank implementation of the VOCODER [4].

Another source-filter method for speech synthesis is the successful Linear Predictive Coding (LPC) technique [1]. This technique provides much of the basis for the work undertaken in this thesis. Markel and Gray’s book “Linear Prediction of Speech” [21] describes the theory behind Linear Prediction in detail. The theory is summarised for the reader in Chapter 6. Linear Prediction is used mainly for speech compression and has even been implemented in dedicated speech synthesis hardware. The “Speak and Spell” toy used a speech chip utilising LPC to compress about 330 words or phrases (approximately 3 to 4 minutes of speech) into two 128 Kbit read only memories [29].

Linear Prediction is a well established area of speech synthesis. The same basic techniques have been in use since its introduction in the late 1960’s. Buzo et al. investigated a technique based on Linear Predictive coding utilizing vector quantization to minimize the amount of transmitted speech data [2]. Vector quantization involves storing a vector of information required for synthesis at both the source and destination of the speech data. Any subsequently analyzed data is converted to a vector and compared with the values in the codebook. The address of the closest matching vector in the codebook is transmitted in place of the actual synthesis data. At the receiving end the synthesis is vector is retrieved from the codebook based on the received address. This results in much improved speech compression.

Wong et al. also experimented with vector quantization to achieve an effective data transmission rate of 800 bits per second. [30]

Another method of synthesis based on LPC techniques is called CELP (Code
Excited Linear Prediction). Standard LPC algorithms use a single pulse excitation as input to a synthesis filter to reproduce speech. CELP uses a multipulse input to the synthesis filter. Again vector quantization is used to select an excitation pulse sequence from within a stored codebook. The codebook index is sent to the receiver which reads an excitation input sequence from its own identical codebook. Kleijn et. al. discuss the original CELP algorithm and introduce an improved technique for speech synthesis with much less computational complexity. [16]

Many algorithms use a technique known as granular synthesis to construct time-varying speech waveforms from small ‘grains’ of sound. The grains are sometimes called acoustical quanta. The length of a grain usually falls within the range 1-50 milliseconds. Grain density is measured in grains per second. High grain densities can be used to create complex evolving sounds such as human speech. A paper by Curtis Roads [26] gives a good introduction to granular synthesis. Granular synthesis can be coarse grained (few grains/second) or fine grained (thousands of grains per second). A real time granular synthesis system using a dedicated digital signal processing chip is described by Traux [27]. He describes three different unit grain models: a simple oscillator with specifiable waveform, frequency and duration; a slightly more complex frequency modulation oscillator pair; and grains based on sampled data. One aspect of the singing synthesis system described in this thesis is based on sampled data granular synthesis.

The PSOLA (Pitch Synchronous Overlap Add) algorithm due to Lent[19] may be regarded as a particular case of granular synthesis. The grains are typically large and vary in size depending on the value of the fundamental pitch period of the waveform being granulated. A grain is usually two pitch periods in length. The PSOLA algorithm is described in detail in Chapter 5.

Granular synthesis is a relatively new technique compared to filter-bank (Fourier Transform) and Linear Predictive Coding techniques. It is unclear which of the methods is better. Studies have been performed to judge the differences, both quantitatively and qualitatively, in output produced by LPC and PSOLA implementations.
van der Sluis found no significant difference in the two algorithms both in computational complexity and in the perceived quality of the output [28]. PSOLA has been judged superior to LPC in another study [23].

The wavelet transform is a more recent method of signal analysis and synthesis. It is related to granular synthesis. The grains are known as wavelets - functions limited in both the time and frequency domains. The wavelet transform decomposes an arbitrary function into a family of elementary wavelets by shifts in the time variable and also by dilations and compressions of both the time and frequency variables [17].

Formant wave function synthesis is used to directly calculate a time varying waveform without the need to separate the source and filter properties of the vocal tract. The waveform at period n is the sum of partial formant wave functions each modelling a formant or a more or less broad portion of the frequency spectrum. Formants and the frequency spectrum are discussed in Chapter 3. The formant wave functions are generated using four parameters, the most important of which are: the center frequency; frequency bandwidth; and amplitude [9].

The CHANT\textsuperscript{1} project [9] used formant wave functions for analysis and synthesis of the singing voice. It provides an environment where the user may specify algorithms to describe parameter evolution or simply set the parameter values explicitly. These parameters may include those discussed in the previous paragraph, or any one of the many other CHANT parameters. CHANT supports about 100 modifiable parameters which can be grouped into major categories, some of which are: fundamental frequency; random variations in fundamental frequency; vibrato; random variations in vibrato; and intensity.

\textsuperscript{1}CHANT is not an acronym
Chapter 3

Fundamentals of Speech Synthesis

3.1 The Vocal Tract

Figure 3.1 shows the major components of the vocal tract used in the production of speech. Almost all sounds are caused by the respiratory system forcing air up from the lungs and through the vocal tract. The air enters the vocal tract through the larynx passing through two muscular folds known as the vocal chords. If the vocal chords are sufficiently close together as the air passes through them they will vibrate. The frequency of vibration depends on the tension of the vocal chords.

The vocal chords do not vibrate during all speech sounds; in fact, there are two distinct classes of vocal sounds, voiced and voiceless. Voiced sounds are those during which the vocal chords vibrate. The vocal chords do not vibrate for unvoiced sounds. Place two fingers against your throat and say first ‘sssss’ and then ‘zzzzz’. Both of these use the same manner of articulation, called Alveolar, with either the tip or the blade of the tongue against the alveolar ridge. The only difference is the vibration of the vocal chords during the ‘z’ sound. You should feel the vibration with your fingers during the ‘z’ sound, but not during the ‘s’ sound. In general for English speech, the consonants are unvoiced and the vowels are voiced. There are exceptions however. For example m, n, and z are all voiced consonants.
3.1.1 Places of Articulation

- **Bilabial**: - made with lips
  - bilabial sounds occur most frequently at the beginning and end of words, eg, ‘power’ and ‘shop’

- **Labiodental**: - made with the lower lip and the upper front teeth
  - e.g. the consonant ‘f’ or ‘ph’ as in ‘p慈善’ or ‘function’

- **Dental**: - made with the tongue tip or blade and the upper front teeth
  - e.g. consonants ‘th’ as in ‘teeth’

- **Alveolar**: - made with the tongue tip or blade and the alveolar ridge
  - e.g. ‘time’, ‘deal’, ‘nice’, ‘sound’, ‘linear’

- **Retroflex**: - made with the tip of the tongue and the back of the alveolar ridge
  - e.g. ‘round’
Many English speakers do not use retroflex sounds at all. For example, as a New Zealander, I pronounce ‘car’ as ‘kār’ replacing the retroflex ‘r’ sound with an extended ‘ah’ vowel. This is also common for Australian speakers and some dialects in England.

- **Palato-Alveolar**: made with the tongue blade and the back of the alveolar ridge
  
e.g. ‘sh’ in ‘shine’.
  
The tip of the tongue may be down behind the lower front teeth or raised near the alveolar ridge.

- **Palatal**: made with the front of the tongue and hard palate.
  
e.g. ‘hūge’.
  
The placement of the vocal articulators is more difficult to detect for palatal articulation. Try placing your forefinger in your mouth with your nail against the alveolar ridge. Now say ‘huge’ slowly and notice the movement of your tongue towards the hard palate (where the tip of your forefinger is). Alternatively for the more germophobic, say ‘huge’ stopping just before the ‘u’ vowel and reverse the airflow in your mouth (suck in instead of blowing out). You should feel cold spots form on the front of the tongue and the hard palate.

- **Velar**: made with the back of the tongue and the soft palate.
  
The more guttural English sounds are formed at the back of the mouth using the back of the tongue against the soft palate, e.g. ‘kind’ and ‘bag’.

Now that we know where articulation can occur we must distinguish between the different types of articulation. Already we have made one major distinction between voiced and unvoiced sounds.

### 3.1.2 Manner of Articulation

We have listed the articulators involved in production of different consonants, but made no mention of the extent of the action causing the sounds. The articulators
may completely block off the vocal tract, allow a small airflow between them, or a relatively free airflow. The introduction of the following terms more clearly defines the nature of produced sounds.

- **Stop**: Complete closure of the articulators involved so the air stream cannot escape through the mouth. There are two different types of stop.
  - **Nasal stop**: The air-stream is prevented from passing through the oral cavity but the soft palate is down, allowing air to pass through to the nasal cavity. E.g. ‘nose’ (alveolar closure) and ‘mouth’ (bilabial closure).
  - **Oral stop**: Both the oral and nasal cavity are blocked. The soft palate is up, blocking airflow through the nasal cavity. Pressure builds up in the oral cavity causing a popping sound when it is released. E.g. ‘pop’ has an oral stop at both consonants. Oral stops are also commonly referred to as plosives.

- **Fricative**: Closeness of two articulators causes turbulent airflow introducing “white noise” or hiss in the vocal output. E.g. ‘f’ in ‘feather’ is caused by labiodental frication and ‘th’ in ‘things’ is an example of dental frication. The ‘ce’ in ‘once’ is an example of alveolar frication.

- **Approximant**: An approximant is similar to a fricative, but in this case, the approach of one articulator towards another is insufficient to produce this turbulent airflow which characterises a fricative. Examples of approximants are ‘once’, ‘we’, and ‘young’.

- **Lateral**: Obstruction of the air-stream along the center of the vocal tract with incomplete closure along either or both sides. The ‘l’ consonant is a good example. Try saying ‘la’, hold the initial ‘l’ sound and breath in, you should feel both edges of your tongue grow colder, indicating the flow of air along both sides of the vocal tract, but none through the center.
So far we have described the vocal tract and named the important vocal articulators. We have introduced terms describing both the places of articulation (Bilabial, Labiodental, alveolar, etc) and the manner of articulation (stops, fricatives, approximants, etc). We now have the tools to describe most of the consonant sounds produced by a speaker of English, although, so far in the discussion we have neglected to mention vowels. Production of vowel sounds is described by the position of the highest point of the tongue. This point is described as being front or back and high-mid or low. Table 3.1 below lists some vowel sound and gives an example word the sounds occur in. It then describes the tongue position during vocalisation of the vowel sound. Try saying each word and notice the position of your tongue.

<table>
<thead>
<tr>
<th>vowel</th>
<th>example</th>
<th>position</th>
</tr>
</thead>
<tbody>
<tr>
<td>ee</td>
<td>weed</td>
<td>front, high</td>
</tr>
<tr>
<td>i</td>
<td>hit</td>
<td>front, mid</td>
</tr>
<tr>
<td>ea</td>
<td>dead</td>
<td>front, mid</td>
</tr>
<tr>
<td>a</td>
<td>sad</td>
<td>front, low</td>
</tr>
<tr>
<td>ah</td>
<td>father</td>
<td>back, low</td>
</tr>
<tr>
<td>ou</td>
<td>wood</td>
<td>back, mid</td>
</tr>
<tr>
<td>oo</td>
<td>food</td>
<td>back, high</td>
</tr>
</tbody>
</table>

Table 3.1: Example of vowel sounds in relation to tongue position

Obviously the positions are only general and do not describe accurately the tongue position and vocal tract shape. The shape of the lips also plays a part in the formation of vowel sounds.

### 3.2 Connected Speech

#### 3.2.1 Phonetic transcription

Connected speech is made up of sentences. Each sentence is made up of words which in turn are formed from sequences of syllables. Syllables may be de-constructed into an even more basic unit called a phoneme. Phonemes are the most basic unit of speech.
They cannot be resolved into smaller identifiable units. Phonetic transcription is the process of writing speech systematically and unambiguously using a basic set of symbols. This is by no means a trivial task. The International Phonetic Association (IPA) founded in 1886 devised a phonetic alphabet applicable to all languages [24]. Appendix F shows a small list of phonemes developed under sponsorship of the US Department of Defense Advanced Research Projects Agency.

### 3.2.2 Prosody/Suprasegmentals

Phonetic transcription alone is insufficient to adequately describe English speech. Prosodic features, also known as suprasegmentals, must also be accounted for. An utterance is categorised as a group of words or a sentence. The most important of these features are pitch and rhythm. Without varying pitch and rhythm, synthesized speech sounds unnatural. As we speak, the rhythm of our voice changes over time. An excited or nervous person speaks rapidly, while a bored and uninterested person speaks slowly and monotonically. The pitch of our voice changes throughout an utterance. The pitch pattern in a sentence is known as the intonation. For example, an increase in pitch at the end of a sentence usually signifies a question. We use amplitude in conjunction with pitch and rhythm to stress particular words in a sentence and perhaps convey additional meaning. Amplitude is generally considered a very weak prosodic feature [29]. Consider the following sentence spoken in a monotone.

> He had a New Zealand accent.

It conveys some information, but not as much as it could. Stressing the words New Zealand gives the sentence a subtly different meaning:

> He had a New Zealand accent (I think you thought it was Australian).

The sentence in parentheses is implied by the stress placed on New Zealand. Prosodic features certainly play an important role in synthesizing realistic speech. Singing synthesis is similar but pitch and rhythm are specified differently in music. Assigning pitch and rhythm to the singing voice is described in detail in Chapter 7.

> Stress within a word in English is often used to distinguish between a noun and a
verb. Consider the word ‘extract’ used in two sentences:

“That is an interesting extract”

and

“Hold still while I extract that tooth”

In the first instance, extract is a noun and the emphasis is placed on the first syllable. In the second, extract is used as a verb and emphasis is placed on the last syllable. Stress is difficult to model in speech synthesis. A stressed syllable is often, but not always, louder than an unstressed syllable. It is usually, but not always, on a higher pitch. The most obvious cue for a stressed syllable is that it frequently contains a longer vowel than usual. Opinion is divided on this subject [18, 20].

All the suprasegmental features, pitch, rhythm, and stress are evaluated by the listener relative to the surrounding words in a sentence. Emphasis can be placed on a word regardless of the overall loudness or pitch of a person’s voice. It is enough that a word be sufficiently different from those surrounding it to make it stand out.

### 3.3 Acoustic Phonetics

So far we have been discussing speech physiologically, making reference to the mechanisms in the vocal tract. While it is important to have a good understanding of the fundamentals of the human speech mechanism, this thesis however is concerned primarily with phonetics from an acoustical point of view, that is, the analysis of sound waves as they are perceived by the human ear. The sound waves produced in the vocal tract are pressure waves. Air passed through the larynx is “chopped up” as the vocal chords vibrate. When the vocal chords close a pressure minimum is formed, and when they open the pressure is maximised. The pressure variation is further modified by the articulators in the vocal tract. Pressure waves radiate from our mouths to our ears vibrating our ear drums. If we were able to graph the movement of our ear drum with respect to time we would get a continuous waveform similar to the one shown in Figure 3.2.
Figure 3.2: Pressure Variation as a Function of Time

If the vocal chords are vibrating (voiced speech) we would see periodicity in the pressure wave. For a section of unvoiced sound the pressure wave appears random. Notice the periodicity in the waveform above. It is taken from a section of sound in which I am singing 'la'. In this case the fundamental frequency—the frequency of vibration of the vocal chords—can be seen clearly marked on the graph. More specifically this is the fundamental pitch period, $T$, measured in seconds. This is a measure of the time between two successive pressure maximums, or alternatively, the time between two successive openings of the vocal chords.

Figure 3.3: Pitch period related to vocal oscillation

The fundamental frequency (measured in Hz or Cycles Per Second (CPS)) is defined as the inverse of the fundamental pitch period. This relation holds in general—any frequency is defined as the inverse of a pitch period, and vice versa.

$$f = \frac{1}{T} \quad (3.1)$$
This relationship is important. It tells us that if we want to change the pitch of a sound, which we must do in speech/singing synthesis with regularity, we must change the duration between pressure maximums. One technique known as PSOLA (Pitch Synchronous Overlap Add), discussed in Chapter 6 directly modifies the pitch period in the time domain. For the remainder of this thesis pitch and frequency will be used synonymously. Some make a distinction from a musical point of view, but we do not for our purposes.

With the introduction of computers, new techniques for analysing and synthesizing sound waves have been developed. A computer ‘listens’ to its environment with the aid of a microphone. A microphone is an electronic equivalent to the human ear. It converts air pressure waves into a corresponding voltage wave with respect to time. A pressure change produces a corresponding voltage change. These changes in voltage are continuous and directly proportional to the changes in pressure. Digital computers cannot process analogue waveforms. The analogue signal must be converted into digital form.

3.4 Sampling

Sampling or Digitizing is the process of converting a continuous waveform into a sequence of discrete measurements. A measurement may only be made with finite precision resulting in discrete amplitude values for each measurement. Discretization in amplitude is known as quantization. Figure 3.4 illustrates the sampling of a time varying waveform.

The duration between measurements of the waveform, $T_s$, is the sampling period. Normally we refer to the sampling frequency $f_s$, the inverse of the sampling period, measured in samples per second. The singing synthesis system uses a sampling frequency of 8000 samples/s ($f_s = 8000$). Therefore the duration between measurements is 125 microseconds. The amplitude of each sample is quantized into 1 of $N$ possible amplitude levels. A digital computer uses binary digits to represent numbers. To
represent $N$ different values we need $n$ binary digits, where:

$$2^n = N$$

Taking logs of both sides we get

$$\log 2^n = \log N$$

$$\Rightarrow n \log 2 = \log N$$

$$\Rightarrow n = \frac{\log N}{\log 2}$$

For example, the singing synthesis system allows for 256 different quantization levels, so it uses $n = 8$ bits (binary digits) for each measurement of the sound wave.

$$n = \frac{\log 256}{\log 2} = 8$$

For comparison, a musical compact disk which also stores sound digitally uses a sampling rate of 44,100 samples per second and 16 bit quantization ($2^{16} = 65536$ different possible amplitude levels).

![Waveform Sampling](image)

Figure 3.4: Sampling a continuous time varying waveform
3.4.1 Linear versus Logarithmic quantization

Quite obviously the quality of sound recorded by the singing synthesis program will not be as good as that of a musical compact disk. It is improved somewhat using logarithmic quantization. Linear quantization divides the amplitude space into \( N \) discrete units in steps of \( 1/N \). Logarithmic quantization however allows more quantization values at low amplitudes. It can be shown that logarithmic quantization has a signal to noise ratio which is independent of the amplitude of the input signal \[29\]. One such logarithmic quantization is known as \( \mu \)-law encoding and is used for transmission of digitized speech over the North American telephone system. Another logarithmic technique known as A-law encoding is the European standard for telephone speech transmission. The transfer function mapping linear samples to logarithmic \( \mu \)-law samples is shown in Figure 3.5.

![Transfer function from linear samples to logarithmic samples (\( \mu \)-law)](image)

Figure 3.5: Transfer function from linear samples to logarithmic samples (\( \mu \)-law)

The transfer function is given by:

\[
y'_\mu(x') = \frac{\ln(1 + \mu |x'|)}{\ln(1 + \mu)} \cdot \text{sign} x'
\] (3.2)

where \( x' = x/x_{\text{max}} \), \( y' = y/y_{\text{max}} \), and \( \mu = 255 \) \[14\].

Signal values at low amplitudes are represented by much larger values in the \( \mu \)-law encoding scheme, making them less sensitive to the effects of small random variations in the signal (noise). 8 bit logarithmic encoding is equivalent in quality to 12 bit
linear quantization [14]. Many hardware platforms sample speech using logarithmic encoding. Sun SPARCStations, on which this project was developed, are one example. For the purposes of signal processing, however, it is important for the speech to be quantized uniformly (linearly), since all of the theory is based on linear systems. This necessitates an extra processing step to convert $\mu$-law encoded samples to linear samples before analysis. In practice this may be performed sample by sample, before each sample is needed. Witten gives a more detailed analysis of linear vs. logarithmic encoding [29].

### 3.4.2 Storage Capacity and Cost considerations

It is worth noting the reasons for choosing a low sampling rate and a small number of quantization bits when the technology obviously exists for very high quality audio input and output, such as that delivered by compact disk technology. While it is possible to sample at 44.1 kHz with 16 bit linear samples on some computers, the hardware is still very much more expensive and less easily available than standard 8 kHz, 8 bit sampling systems. More importantly for high sampling frequency large word-length systems, storage capacity becomes an issue. To store a one minute recording at 44.1 kHz using 16 bit quantization, we need $44100 \text{ samples/second} \times 2 \text{ bytes/sample} \times 60 \text{ seconds} = 5,292,000 \text{ bytes of storage}$. That is 5.17 Mbytes. Considering the average PC only has 4 Mbytes of Random Access Memory, this is clearly a problem. At an 8 kHz sampling rate with 8 bit quantization, the storage requirement is reduced to $8000 \text{ samples/second} \times 1 \text{ byte/sample} \times 60 \text{ seconds} = 480,000 \text{ bytes, or approximately 470 kbytes}$.

Processing time is also a major issue. The time to process 44.1 kHz samples is much greater than the processing time required for 8 kHz samples recorded for the same duration. Assuming the processing time for a 16 bit sample is similar to that of an 8 bit sample, the processing time for a 44.1 kHz sample buffer will be 5.51 times longer than the processing time for an 8 kHz buffer (since the 44.1 kHz buffer contains 5.51 times more samples than the 8 kHz buffer).
3.5 Spectrographic Analysis and Sound Quality

Sound quality is not a measure of how “good” a sound is, it is a qualitative classification of a sound. A sound contains not only the fundamental frequency given by the oscillation of the vocal chords, but also a number of overtones created by the vocal tract itself. The shape of the vocal tract determines the frequencies of those overtones, known as formants. These formants give a sound its distinctive quality regardless of whether the sound is voiced or voiceless. For example, consider the vowel sounds ‘ah’ and ‘ee’ in the words ‘father’ and ‘me’.

First say the sounds ‘ah’ followed by ‘ee’, and then whisper them. In a whispered voice the vocal chords are not oscillating (there is no fundamental frequency) although the sounds still retain their characteristic quality due to the frequencies of the overtones or formants. The frequency domain graphs in Figures 3.6 and 3.7 show the relative amplitudes of the frequency components in the sounds ah and ee.

![Frequency Spectrum for 'ah'](image)

Figure 3.6: Frequency Spectrum for the vowel ‘ah’ as in ‘Father’

The sound spectrograph, discussed in Appendix D is another mechanism for studying the qualitative nature of speech sounds. The spectrograph is capable of showing the formant structure of speech. Put simply, a sound spectrograph is a frequency vs. time graph, showing multiple frequencies per unit time. The intensities of the frequencies at each unit time are indicated by a colour or gray scale. In this implementation, a colour scale is used. The scale runs from blue through green through yellow,
orange, and finally to red. 128 different intensities are supported. Hot colors such as red show frequency components with high amplitude in the signal. Cold colors such as green and blue show frequency components with low amplitude. On a color plot it is easy to pick out the formant frequencies as concentrated bands of color. Figure 3.8 shows the spectrographic plots of eight vowel sounds sung in isolation. White horizontal lines are drawn in over top of the spectrographs to illustrate the positions of the formants more clearly.
Figure 3.7: Frequency Spectrum for the vowel ‘ee’ as in ‘see’

Figure 3.8: Spectrographic plots of eight common vowel sounds sung in isolation
Chapter 4

Time Domain Pitch Estimation

An important task in the processing of sound waveforms is the estimation of the fundamental frequency, or the pitch of the sound. Chapter 3 showed that speech often consists of more than one frequency, the fundamental frequency plus those of the formants or resonant frequencies of the vocal tract. These formant frequencies cause considerable complications in the estimation of the fundamental frequency. Many pitch tracking algorithms use the distance between major peaks of the time domain waveform to determine the pitch period $T$, and hence the fundamental frequency $f$, using the relation

$$f = 1/T$$

(4.1)

Determining the pitch period is not trivial however.

4.1 Gold Rabiner algorithm

A common and successful pitch determination technique is that of Gold and Rabiner [12]. This is a parallel algorithm utilizing six different pitch period estimators and suitably combining the results to obtain an accurate estimate of the pitch period. The algorithm can be divided into four distinct processes as shown in Figure 4.1. \(^1\)

The four processes, to be discussed in detail below, are:

\(^1\)The diagrams describing the Gold Rabiner algorithm are taken from [12].

24
singing/speech

Filter Signal-peak Processor
PPE1 PPE2 PPE3 PPE4 PPE5 PPE6

Final Pitch Period Computation

Final Pitch period

Figure 4.1: Block Diagram for the Gold Rabiner Pitch Tracking algorithm

- Filtering the input speech signal
- Generation of the six functions of peakedness of the filtered signal
- Six identical pitch period estimators, each working on one of the six measures of peakedness
- Final pitch period computation based on the results from each of the simple pitch period estimators

### 4.1.1 Filtering

The filter design for the algorithm is not critical. The primary purpose of the filter is to select the first formant region of speech and reduce the amplitude of higher frequency components (those of the higher formants for example), which complicate the speech waveform and make fundamental frequency estimation difficult. This type of filter is known as a low pass filter, allowing low frequencies to pass through and attenuating frequencies above a particular cutoff frequency. Filter design is beyond
the scope of this thesis. Appendix A, however, discusses the low pass filter used in this thesis, and shows how to determine the filter frequency domain transfer function give then time domain filter equations. A good book on the subject of digital filter design is called “Digital Filters” by R. W. Hamming. The filter used here has a low pass cutoff of 900 Hz, allowing up to two formants to pass through. Figures 4.2 and 4.3 show the effect this low pass filter has on a typical speech waveform.

![Unfiltered waveform](image1)

**Figure 4.2: Unfiltered waveform**

![Low pass filtered waveform](image2)

**Figure 4.3: Waveform filtered with 900 Hz low-pass filter**
4.1.2 Measures of peakedness

The measures of peakedness determined from a filtered speech signal are shown in Figure 4.4.

The measures are described as follows:

\( m_1 \) - Peak height

\( m_2 \) - Valley-to-Peak height

\( m_3 \) - Peak-to-Peak height (if greater than zero)

\( m_4 \) - Valley depth

\( m_5 \) - Peak-to-Valley depth

\( m_6 \) - Valley-to-Valley depth (if greater than zero)

![Figure 4.4: Measures of Peakedness](image)

Measurements \( m_1 \) to \( m_3 \) are generated at each positive peak while those of \( m_4 \) to \( m_6 \) are generated at each negative peak. All of these measurements are converted into positive pulse trains - one pulse train for each of the six measures of peakedness. Figure 4.5 shows a series of pulses associated with one of the six measures of peakedness. Measurements \( m_3 \) and \( m_6 \) must always be greater than zero, that is if a current peak or valley is not as large as a previous peak or valley, its corresponding measurement is set to zero. These measurements were chosen on the basis of two extreme cases:
• when only the fundamental frequency is present, the pulse trains of measurements $m_1$, $m_2$, $m_4$, and $m_5$ indicate the fundamental pitch period by the durations between pulses.

• when a strong second harmonic is present (first formant), the pulse trains of measurements $m_3$ and $m_6$ indicate the correct pitch period by the durations between pulses.

4.1.3 Parallel Pitch Period Estimates

As mentioned previously, each of the functions of peakedness provides a pulse train as input to a simple pitch period estimator. Following each detected pulse there is a blanking interval during which no other pulse may be detected. Following this blanking interval there is an exponential decay during which the amplitude of any pulse must exceed the value of the decay curve for that pulse to be detected. This is illustrated in Figure 4.5. The three most recent pitch period estimates are marked as $P_n$, $P_{n-1}$, and $P_{n-2}$.

![Diagram](image)

Figure 4.5: Peak detection using variable blanking time and exponential decay

A running pitch average is maintained, where

$$P_{av}(n) = \frac{P_{av}(n-1) + P_n}{2}$$

(4.2)
and

\[ P_{av}(0) = P_0 \]  \quad (4.3)

\( P_{av}(n) \) is the new average pitch period. \( P_{av}(n - 1) \) is the previous average pitch period, and \( P_n \) is the newly calculated pitch period.

The variable blanking time is given by:

\[ \tau = 0.3 * P_{av} \]  \quad (4.4)

The value of the variable exponential decay at time \( t \) is given by

\[ y = y_0 e^{-kt} \]  \quad (4.5)

where \( y_0 \) is the amplitude of the previous pulse, \( t \) is the elapsed time since the end of the blanking interval, and \( k = 0.695/P_{av}(n - 1) \). The parameters 0.3 and 0.695 are heuristically determined values [12].

### 4.1.4 Estimated Pitch Period Matrix

The final pitch period computation uses a 6x6 matrix of current and previous pitch period estimates.

\[
\begin{array}{ccccccc}
\text{PPE No.} & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
\text{row 1} & P_{11} & P_{12} & P_{13} & P_{14} & P_{15} & P_{16} \\
\text{row 2} & P_{21} & P_{22} & P_{23} & P_{24} & P_{25} & P_{26} \\
\text{row 3} & P_{31} & P_{32} & P_{33} & P_{34} & P_{35} & P_{36} \\
\end{array}
\]

**Figure 4.6: 6x6 Matrix of Pitch Period Estimates**
The first row of Figure 4.6 gives the most recent pitch estimates. The second and third rows give estimates at time \( n - 1 \) and \( n - 2 \) respectively. The remaining three rows are calculated as follows:

\[
P_{4k} = P_{1k} + P_{2k} \quad k = 1, \ldots, 6
\]

(4.6)

\[
P_{5k} = P_{2k} + P_{3k} \quad k = 1, \ldots, 6
\]

(4.7)

\[
P_{6k} = P_{1k} + P_{2k} + P_{3k} \quad k = 1, \ldots, 6
\]

(4.8)

That is, the fourth row is the sum of the first and second rows, the fifth row is the sum of the second and third rows, and the sixth row is the sum of the first three rows. The purpose of the last three rows of the pitch period matrix is to correctly determine the fundamental frequency when the individual pitch period estimators detect the second or third harmonics. The second harmonic is twice the fundamental frequency (half the fundamental pitch period), the third harmonic is three times the fundamental frequency (one third of the fundamental pitch period). If a pitch period estimator detects a second or third harmonic, the entries in the last three rows will give the correct pitch period.

### 4.1.5 Final Pitch Period Computation

The final pitch period is chosen from one of the six most recent pitch period estimates - one of the elements on the first row of the pitch period matrix (see Figure 4.6). The chosen pitch period is that with the greatest number of coincidences in the pitch period matrix. Two pitch periods coincide if they differ by less than a specified amount. This amount is called the coincidence window width. Table 4.1 shows a table of coincidence window widths (measured in samples) as a function of pitch period and bias.

The purpose of the bias will be discussed below. A coincidence width of 1 sample is \( 1/8000 = 125\mu s \). The widths in each row of table 4.1 are roughly proportional to the pitch periods of that row.
Table 4.1: Coincidence Window Widths (in samples) as a function of Pitch Period and Bias

For example, at bias level 1:

\[
\begin{align*}
1 \cdot 125\mu s & : 125\mu s/3.1ms = 0.0403 \\
2 \cdot 125\mu s & : 250\mu s/6.3ms = 0.0397 \\
4 \cdot 125\mu s & : 500\mu s/12.7ms = 0.0393 \\
8 \cdot 125\mu s & : 1ms/25.5ms = 0.0392
\end{align*}
\]

So at bias level 1 the coincidence window width is approximately 4% of the fundamental pitch period. Similarly at bias level 2 the coincidence width is approximately 8% of the fundamental pitch period. Table 4.2 shows the coincidence window width as a percentage of pitch period for each bias level.

Table 4.2: Coincidence width as a percentage of pitch period as a function of bias

Obviously the higher the bias level, the greater chance of two pitch periods in table 4.6 coinciding. For each pitch period estimate \( P_{11}, \ldots, P_{16} \) in Table 4.6, the
number of coincidences in the table are calculated once for each of the bias levels. At each bias level, the bias value is subtracted from the total coincidences for that level. The grand total of all coincidence counts for each bias level is stored for each pitch period estimate \( P_{11}, \ldots, P_{16} \). The winning estimate is that with the highest grand total.

Consider the example below showing a typical calculation. The pitch period estimates are shown in Table 4.3.

Each of the pitch period estimates are stored as a number of samples. For example: \( P_{11} = 51 \times 125 \mu s = 6.375 \) ms. This corresponds to a fundamental frequency of 156.8 Hz. The coincidence window width at bias level 1 for this pitch period (taken from Table 4.1) is 4 samples * 125 \( \mu s = 500 \mu s \) (row 3, column 1 of table 4.1). There are 7 coincidences in the table: \( P_{11}, P_{14}, P_{15}, P_{21}, P_{25}, P_{32}, \text{ and } P_{35} \). Each of these is within four samples of \( P_{11} \). Subtracting a bias value of 1 gives a total of 6 coincidences at bias level 1.

Table 4.4 shows the coincidences for each pitch period estimator for each bias level:

\[
\begin{array}{ccccccc}
\text{bias} & P_{11} & P_{12} & P_{13} & P_{14} & P_{15} & P_{16} \\
1 & 6 & 1 & 2 & 2 & 6 & 1 \\
2 & 7 & 2 & 2 & 4 & 7 & 0 \\
5 & 5 & 3 & 0 & 2 & 6 & 3 \\
7 & 5 & 6 & -1 & 2 & 5 & 5 \\
\text{total} & 23 & 12 & 3 & 10 & 24 & 9 \\
\end{array}
\]

Table 4.4: Final Coincidences for each estimator for each bias level
The estimate with the highest grand total is $P_{15}$ with a coincidence total of 24. This gives a fundamental pitch period of $P_{15} = 52 \times 125 \mu s = 6.5$ ms. The fundamental frequency is therefore $f = 1/6.5ms = 153.85$ Hz.

While the Gold Rabiner algorithm is described as a parallel algorithm it need not necessarily be implemented as such. In fact, in this implementation the algorithm is written for a general purpose serial computer. This provides greater flexibility for porting across different platforms since no special hardware or operating system support is required.

4.2 Simple Inverse Filter Tracking - The SIFT algorithm

SIFT is a pitch tracking technique based on general speech processing theory known as Linear Predictive Coding. Some LPC theory is discussed in Chapter 6. The SIFT algorithm has two main stages; the first is to pre-process the speech signal using a technique called ‘spectral flattening’ to reduce the effects of the interaction between the fundamental and the formant frequencies. If the harmonics of the fundamental frequency can be readjusted to a constant value then the resulting waveform will have sharp pulses at intervals of the fundamental pitch period, without any formant structure between the peaks.

The first stage consists of the following processes, discussed in detail below:

- pre-filtering
- down sampling
- differencing
- windowing
- determination of the Inverse Filter coefficients
• Inverse Filtering

• windowing

This algorithm differs from Gold and Rabiner’s in that emphasis is placed on pre-processing of the input speech waveform to remove formant effects, followed by a relatively simple algorithm for measurement of the fundamental frequency using autocorrelation.

4.2.1 Pre-filtering

The pre-filtering stage is a simple digital filter (see Appendix A). Frequencies above 900 Hz are attenuated, i.e. it is a low pass filter with a cutoff frequency of 900 Hz. You will notice that the description of this filter is the same as that for the Gold Rabiner filter. In fact, the same filter code is used for both pitch trackers.

4.2.2 Down sampling

Down sampling is designed to reduce the effective sampling frequency, in this case to 2000 Hz (down from 8000 Hz). This reduces the computational load of the algorithm and allows processing of the frequency range between 0 and 1000 Hz. Down Sampling is achieved using a process known as decimation. [14].

4.2.3 Differencing

The speech is differenced to accentuate the region of the second formant of the speech to be analysed. Appendix B discusses the differencing pre-emphasis filter used in the SIFT algorithm. Voiced speech radiated from the lips is attenuated. Higher frequencies are attenuated more than lower frequencies. The differencing results in “spectral equalisation”, ensuring similar energy levels over the full bandwidth of speech.
4.2.4 Windowing

Any finite sequence of time varying data can be thought of as a window of samples from an infinite data series. The concept of a window is illustrated in Figures 4.7 and 4.8 below.

![Unwindowed Data](image)

Figure 4.7: Unwindowed sampled data

Many signal processing techniques use the concept of a window, including many digital filter implementations. Analysis of a finite section of a waveform is equivalent to setting the waveform to zero outside of a rectangular window.

![Windowed Data](image)

Figure 4.8: Sampled data windowed using a rectangular window

The term “rectangular” usually implies that the amplitude of the samples within the window are unmodified by the effect of the window. It is possible however, that the window’s upper and lower bounds truncate some sample values. Usually
Figure 4.9: Waveform discontinuities caused by rectangular windowing

A rectangular window is only implied by the examination of a finite section of a waveform and therefore the upper and lower bounds are infinite.

The application of a rectangular window to a speech signal whether by implication or intention can have unfortunate consequences for fundamental frequency estimation, or any other spectral analysis for that matter. The discontinuities at the edges of the window (see Figure 4.9) introduce unwanted high frequency components into the signal [29]. The audible effects of discontinuities in a sound wave will be discussed in more detail later. It is often necessary to examine speech frame by frame\(^2\) however, so we must find ways to remove the discontinuities. This is done by using a non-rectangular window which attenuates the sample amplitudes near the edges of the window.

One such window is the Hamming window, shown in Figure 4.10. The window is defined by the function:

\[
f(x) = w \ast (0.54 - 0.46 \cos \frac{2\pi x}{N - 1})
\]

(4.9)

The scale factor \(w\) is included to compensate for the signal energy loss caused by attenuating the samples at the edges of the window. The parameter \(N\) is the window width.

To reduce the effects of windowing the signal, each frame overlaps the previous one

\(^{2}\text{a frame is another word for a window.}\)
by 100 samples. The frame is 300 samples in length, so each analysis frame consists of 200 new samples and 100 samples from the previous frame.

![Graph of the Hamming Window](image)

Figure 4.10: The Hamming Window

### 4.2.5 Determination of Inverse Filter Coefficients

The inverse filter is generated using linear prediction techniques. Linear prediction and determination of the inverse filter are described in detail in Chapter 6. The resulting inverse filter, when applied to the signal used to generate the inverse filter, produces a spectrally flattened output signal. The inverse filter used is a fourth order filter. That is, it uses four filter coefficients, \(a_1, \ldots, a_4\). Markel and Gray state that four coefficients are sufficient to cover the range of the first two formants of speech—from 0-1000 Hz [21]. The inverse filter is physically realised as a four stage lattice filter. The Lattice filter is discussed more fully in Chapter 6. Appendix C determines the transfer function of the four stage lattice filter. Figure 4.11 shows the different stages involved in the determination and application of the inverse filter. Figure 4.11(a) shows an input frame of data to be inverse filtered, (b) shows the same frame after 4:1 decimation, (c) shows the decimated frame after differencing, (d) shows the differenced frame after windowing. The frame shown in Figure 4.11(d) is used to determine the coefficients of the inverse filter. The coefficients themselves are uninteresting, but they can be used to plot the transfer function determined
using Maple as described in Appendix C. The transfer function resulting from these coefficients is shown in Figure 4.11(e). Applying the inverse filter generated from the frame shown in (d) to the decimated data in (b) yields the inverse filtered frame of data shown in (e). While this does not look significantly different from the decimated frame in (b), its frequency spectrum is remarkably different. The frequency spectrums of frames (b), (d), and (e) are shown in Figures 4.11(g), (h), and (i) respectively. Figure 4.11(i) shows a considerably flattened speech spectrum when compared to frames (g) and (h).

![Frequency Spectrum Diagram](image)

**Figure 4.11**: Pre-processing stages of the SIFT algorithm
The second stage of the SIFT algorithm involves determination of the fundamental frequency from the pre-processed data from stage 1 shown in Figure 4.11(f). The primary analysis technique of the second stage is autocorrelation.

### 4.2.6 Autocorrelation

The autocorrelation function of a signal is given by:

\[ \varphi(k) = \sum_{n=-\infty}^{\infty} x(n)x(n + k) \]  

(4.10)

where \( k \) is the lag value, and \( x(n) \) is amplitude.

The short time autocorrelation is given by:

\[ \varphi(k) = \sum_{n=0}^{N-1-k} x(n)x(n + k) \]  

(4.11)

for

\[ k = 0, \ldots, N - 1 \]  

(4.12)

The autocorrelation function exhibits peaks at lags which correspond to the pitch period and multiples of it. The zero lag autocorrelation \((k = 0)\) is simply the window of samples multiplied by itself. Obviously when \( k = 0 \), peaks in the signal will be multiplied by themselves resulting in a corresponding peak in the autocorrelation function. The zero lag autocorrelation will exhibit a peak, but is generally of little use, other than to serve as a reference for other detected peaks. A periodic waveform of frequency, \( f \), will exhibit a peak at lag \( k = 1/f \), therefore \( \varphi(k) \) will be a local maximum when \( k = 1/f \).

The size of the window, \( N \), depends on the range of pitch periods to be detected. The SIFT algorithm is suitable for determining the pitch of a signal for pitches in the range 50Hz \( \rightarrow 250 \) Hz. This corresponds to lags of 20 ms \( \rightarrow 40 \) ms, or alternatively (for a 2000 Hz sampling rate), between 40 samples and 8 samples. For a window to contain at least two pitch periods even for the lowest frequency under consideration, it must be at least 40 ms long. The window used in this implementation is 300 samples (37.5 ms) long with lags calculated from \( k=0 \) to \( k=75 \).
The autocorrelation output is then examined to determine the peak position. Quadratic interpolation is used to increase the pitch resolution which counters the effects of down sampling the waveform in stage 1 [21].

Figure 4.12: Autocorrelation of a periodic time varying waveform
Chapter 5

Pitch Modification: The PSOLA algorithm

One of the primary goals of this project is the modification of the fundamental frequency of time domain waveforms while retaining the natural sounding quality of the speech. Recall from Chapter 3, that to modify the fundamental frequency, we must change the length of the fundamental pitch period. One technique to accomplish pitch modification is to alter the sampling rate of the signal on playback. This results in compression or expansion of the waveform, causing both an increase or decrease in the length of the speech (and all pitch periods contained within it), as well as an increase or decrease of the fundamental frequency. The major disadvantage of this method is that the spectral envelope of the sound is modified along with the fundamental frequency, resulting in a chipmunk-like sound [19]. Traditional techniques use Fourier Analysis to determine the frequency spectrum of a section of the speech wave. Subsequent resynthesis can retain the formant structure of the speech wave, while modifying the pitch. Fourier Analysis/Resynthesis is a computationally complex process however, and is not suitable for real-time, or near real-time implementations. Linear Predictive Coding, discussed in Chapter 6 can also modify the pitch of a speech wave while retaining the correct formant structure, and is much faster than the Fourier analysis method.
A recent study uses a technique called Pitch Synchronous Overlap Add (PSOLA) for pitch period modification [19]. The algorithm reconstructs a speech signal by extracting and modifying single pitch period segments of the waveform. A typical voiced speech waveform is shown in Figure 5.1 below.

![Waveform for typical voiced speech](image1)

**Figure 5.1:** A typical section of voiced speech

A single pitch period extracted from the waveform using a rectangular window is shown in Figure 5.2.

![Waveform for single pitch period](image2)

**Figure 5.2:** A single pitch period extracted from the waveform shown above
5.1 PSOLA synthesis

The waveform can be resynthesised by adding the extracted pitch period to a delayed version of itself. If the delay is equal to the pitch period then no pitch modification will occur. Increasing the pitch period (decreasing the frequency), simply requires increasing the delay, while decreasing the pitch period (increasing the frequency) requires decreasing the delay. An increase in the delay corresponds to padding the extracted pitch period with zeros to the end of the new pitch period length. A resynthesized waveform is shown in Figure 5.3

![Figure 5.3: A PSOLA waveform with decreased pitch resynthesized using a rectangular window](image)

Obviously the analysis/synthesis technique is grossly oversimplified. Recall from the discussion in Chapter 3, that windowing a signal using a rectangular window results in the introduction of high frequency components due to the sharp transitions caused by discontinuities in the waveform. These discontinuities can be clearly seen in Figure 5.3. It is not immediately obvious, but discontinuities can also occur when decreasing the pitch period. The addition of large valued samples in the overlapping portions of two extracted pitch periods can result in an overflow. In a signed fixed length binary representation, addition of two sufficiently large positive numbers results in a negative number when overflow occurs. The overflow causes a rapid transition in the synthesized speech waveform, resulting in the introduction of
high frequency components to the speech spectrum. These high frequencies cause noticeable pops and clicks in the synthesized speech.

5.2 PSOLA Implementation

5.2.1 Extracting a single pitch period

The Gold Rabiner pitch tracking algorithm discussed in Chapter 4 is used to determine the fundamental frequency of the waveform to be analyzed. An accurate estimate of the fundamental frequency is important to determine the size of the window used to extract a single pitch period of the waveform. A Hanning window\(^1\) [15] with width twice the fundamental frequency is used. The window is shown in Figure 5.4.

![Hanning Window](image)

**Figure 5.4:** Hanning window for extracting pitch periods

The window is defined by the function:

\[
f(x) = w \times (0.50 - 0.50 \times \cos \frac{2\pi x}{N - 1})
\]  

(5.1)

The scale factor \(w\) is included to compensate for the signal energy loss caused by attenuating the samples at the edges of the window. The parameter \(N\) is the window width. Notice that this window differs from the Hamming window only in the values

---

\(^1\)A Hanning window differs from the Hamming window discussed previously
of its coefficients. However the effects of this difference in the frequency domain are quite remarkable and are discussed in detail in [15].

This window (width 100 samples) would be appropriate for extracting a pitch period of length 50 samples. These widths correspond to 12.5ms and 6.25ms respectively (assuming a sampling frequency of 8000 Hz). A pitch period of 6.25ms corresponds to a fundamental frequency of 160 Hz. Obviously a longer pitch period requires a longer Hanning window. Ideally the Hanning window should be centered around the major peak in each pitch period. The implementation of the Gold Rabiner algorithm returns the position of this major peak as well as the estimated fundamental frequency. A slightly different filter cutoff is used from that described in Chapter 4. The cutoff is lowered to 250 Hz to retain only the fundamental frequency. While the pitch range is restricted, the chances of an incorrect estimate of the fundamental frequency is reduced. The Gold Rabiner algorithm sometimes makes octave errors in the frequency estimation. That is, the frequency detected is either twice as large, or half as large, as it should be. An incorrect frequency estimate of this magnitude, resulting in an extraction window width of an inappropriate size, can be disastrous to the quality of the synthesized singing. A rapid frequency transition of an octave would be replicated on resynthesis, causing the output to sound unnatural, and in most cases laughable. High quality frequency analysis and accurate detection of signal peaks is important for accurate resynthesis of an input waveform [19].

Figure 5.5 shows a small portion of two signals. One is the original signal, the other the resulting signal after low pass filtering the first signal with a 250 Hz cutoff.

The graph immediately illustrates a big problem caused by low pass filtering of the original waveform. Low pass filtering by definition filters out high frequency components. It is these high frequency components however that give a sound its crisp clear quality. Low pass filtered speech or singing sounds muffled and indistinct. Therefore the original waveform must be used for the extraction of pitch periods which will subsequently be used for the resynthesis of the singing or speech, since resynthesis from the filtered waveform will also sound muffled. We have already
Figure 5.5: An original waveform with its low pass filtered (250 Hz cutoff) counterpart mentioned that accurate detection of the signal peaks for positioning of the Hanning window is essential to ensure high quality synthesis. It can be noticed however that the peaks in the filtered signal no longer correspond to the peaks in the unfiltered signal. Although the estimated pitch and therefore the length of the Hanning window will be correct, the position of the Hanning window in the unfiltered signal will be incorrect. Figure 5.5 would indicate that a constant (negative) offset added to the peak locations returned by the Gold Rabiner algorithm would compensate for the filtered waveform “shift”. In practice this compensation, as shown in Figure 5.6 appears to work well.

Figure 5.6: Filtered and Unfiltered waveforms showing compensated peak positions

The algorithm so far has been concerned only with periodic waveforms, i.e. voiced
speech. Obviously much of speech and singing is unvoiced speech. The algorithm must deal with this also. An upper and lower bound is placed on the pitch that the Gold Rabiner algorithm returns. For any pitch within the range of 50 Hz to 220 Hz, the section of the waveform near the pitch mark is said to be voiced. For any detected pitch outside this range, the section of the waveform is said to be unvoiced. Obviously this is a gross generalisation, since many female and children’s voices have a high pitch well above the 220 Hz mark. The range bounds are purely arbitrary. This kind of pitch restriction is not uncommon. Many pitch detection algorithms, including the SIFT algorithm are designed only for relatively low frequency analysis, excluding female and children’s voices. It should be noted however, that the Gold Rabiner algorithm implemented here can (with reasonable accuracy) track the fundamental frequency of a singing voice up to 880 Hz. It is restricted here only to obtain an indication of the voiced or unvoiced nature of male singing/speech in the absence of other techniques for voicing decisions. Apologies are given to women and children.

Unvoiced pitch marks separated by 10 ms (80 samples at an 8 kHz sampling rate) are generated for the unvoiced sections of the waveform preceding, between, and following voiced sections. These marks will also form the center for Hanning windows to extract portions of the waveform. The Hanning windows for unvoiced speech marks are 20 ms (160 samples) in length. Upon resynthesis, the distance between unvoiced speech marks is maintained at 10 ms regardless of any pitch modification factor. It would be meaningless to modify the pitch of an unvoiced section of the waveform, since this will not affect the fundamental frequency at all.

Extracting a pitch period of speech is trivial given a pitch mark and a window width. Figure 5.7 below shows a PSOLA grain extracted using a rectangular window.

The same grain after multiplication by the Hanning window is illustrated in Figure 5.8. The Hanning window is shown for reference, scaled by a factor of 100. The cost of storing a granulated waveform is high, much higher than that of storing it in a standard sampled form. Assuming equally spaced pitch marks, the grain data alone

---

2Could it be that the lack of high frequency pitch detection algorithms stems from the fact that most researchers in analysis and synthesis of speech are men?
Figure 5.7: A PSOLA grain extracted using a rectangular window

is a factor of two larger than the sampled data. Since every grain covers two pitch periods and pitch marks occur every pitch period, this corresponds to a granular overlap of 50%. The grains are stored in a linked list implementation which requires an extra 40 bytes per grain.

Figure 5.8: PSOLA grain modulated by the Hanning window.

Traditional techniques analyze an existing waveform and synthesize a new one without storing the grains as an intermediate representation. This, however, would require accurate fundamental frequency estimation, pitch period extraction, and re-synthesis all in real time to be effective for the purposes of this project. Lent suggests a real time implementation of the PSOLA algorithm is an achievable goal [19].
With the waveform stored in granular form, resynthesis of the waveform is relatively trivial and quite fast. For an original waveform of length N samples (stored in granular form totalling 2N samples), reconstruction at the same pitch and duration requires overlapping each frame with the previous one. The size of the overlap is 50% of the grain size. Since each overlapping sample of grain $i$ must be added to the overlapped samples of grain $i-1$, a total of N additions over all grains is required, since half of all grains are overlapped \(^3\). Obviously, every sample of every grain is stored once (whether it be as an argument to an addition or not). Thus 2N stores are required. This makes re-synthesis $O(N+2N) = O(3N) = O(N)$. Decreasing the pitch, decreases the overlap and therefore the number of additions. No overlap at all of course requires no additions, although 2N stores are still required. As an upper bound, as the pitch period tends to 0, every grain would be centered at the same position. Therefore synthesis would require 2N additions and 2N stores. Figure 5.9 shows the first few grains (represented by their Hanning windows). Each is overlapped by 50%.

---

\(^3\)A constant factor is omitted here—no additions take place for the first grain

Figure 5.9: Overlapping PSOLA grains represented by their Hanning windows.
5.2.2 Single Grain Synthesis

Two types of granular synthesis were attempted. The first uses only a single grain to synthesize a constant tone—a monophonic vowel sound such as 'ah' for example. Single grain synthesis requires the knowledge of the recommended pitch of the output waveform. A continuous wave consists of many pitch periods, and the grain extraction algorithm needs some criterion for picking any one particular grain. With knowledge of the required pitch of the synthesized waveform the grain extraction algorithm uses the Gold Rabiner algorithm to determine the frequency of the input wave at every major pitch peak. The peak with the closest fundamental frequency to that specified, is chosen and extracted using the Hanning window. This usually requires user knowledge of the frequency of the input signal. Waveform synthesis using only a single grain involves adding the grain to a delayed version of itself. The delay (for unmodified pitch) is again 50% of the grain length. A typical result of this type of PSOLA synthesis is shown in Figure 5.10.

![Waveform Synthesized using a single PSOLA grain](image.png)

Figure 5.10: PSOLA synthesis from a single grain.

Single grain synthesis sounds very synthetic. There is no variation in the tonal quality of the synthesized sound at all. Attempts to randomize the amplitude and frequency of the waveform, by making small changes to a grain scale factor and the grain delay were unsuccessful. Larger changes cause very noticeable roughness, including clicks and pops, in the synthesized speech. A vibrato with a small randomization
factor could possibly be implemented to improve the quality [9]. Single grain synthesis was largely abandoned in favour of multi-grain synthesis, which produces much higher quality output over a large range of sounds.

5.2.3 Multi Grain Synthesis

Multi grain synthesis uses many different grains to synthesize a new speech wave. Each grain is extracted in series—one for each successive voiced pitch mark as detected by the pitch tracker. If the distance between pitch marks is larger than 20ms (160 samples assuming 8kHz sampling), that is, the detected pitch is less than 50 Hz, then the unvoiced section of speech between the current and previous voiced pitch mark is filled with unvoiced marks separated by 10 ms delays. The grains between the last and current marks are extracted in order and stored in the linked list. The pitch tracker is then requested to determine the next voiced pitch mark and the entire process repeats until the end of the input signal is reached. Re-synthesis involves scanning through the grain list and writing each grain separated by a single pitch period delay. Any grain overlap results in an addition and store operation. A section of a typical synthesized waveform is illustrated in Figure 5.11.

![Section of a synthesized PSOLA waveform (multiple grains)](image)

Figure 5.11: PSOLA synthesis from multiple grains.
Figure 5.12 below shows the spectrograph of the output of the PSOLA algorithm using Multi Grain synthesis on eight common vowels sung in isolation. The sung vowels are the same as those used in Figure 3.8 in Chapter 3. This enables a direct comparison between the two spectrographs and verifies that the PSOLA algorithm preserves the spectral nature of speech sound upon resynthesis. As in Figure 3.8 the horizontal lines show the positions of the formants more clearly.

![Spectrograph of output of Multi grain PSOLA synthesis of eight common vowels.](image)

**Figure 5.12:** Spectrograph of output of Multi grain PSOLA synthesis of eight common vowels.
Chapter 6

Linear Predictive Coding (LPC)

Linear Predictive Coding was inspired by the need to represent the speech wave in terms of a small number of slowly varying parameters without using Fourier analysis to convert the speech into the frequency domain. The Fourier Transform is slow and suffers from the frequency-time tradeoff limitation. Linear Prediction has become a dominant technique since its introduction in the 1960’s [1]. Linear Predictive Coding is a source-filter speech model. Recall that a source filter model of speech separates the excitation function, the source (the vocal chord oscillation or the effective pitch), from the filter (the vocal tract). This is of particular importance for us, since we need to store sounds corresponding to English phonemes separately from their pitch.

There are two methods in common use for LPC. These are: the covariance method and the autocorrelation method. We use the autocorrelation method in this project, since it produces more stable output speech than the covariance method [29]. The covariance method will not be discussed. A detailed comparison between the two techniques is made in [29, 20, 21].

LPC is based on the principle of the prediction of a data sample, given previous data in a time series. It is obviously applicable for speech since a sampled waveform is a time series. Using only one previous sample to predict the following we form an equation such as:

\[ x(n) = a_1 x(n - 1) + e(n) \quad (6.1) \]
where $a_1$ is a slowly varying parameter, $x(n)$ and $x(n - 1)$ are the current and previous samples, and $e(n)$ is the error—the difference between $x(n)$ and its true value. Rearranging we get:

$$e(n) = x(n) - a_1 x(n - 1)$$ (6.2)

Using $p$ past samples to generate $x(n)$ we get:

$$e(n) = x(n) - a_1 x(n - 1) - a_2 x(n - 2) - \cdots - a_p x(n - p) = x(n) - \sum_{k=1}^{p} a_k x(n - k)$$ (6.3)

The LPC parameters $a_1, \ldots, a_p$ are adapted to minimise the error signal. Their frequency of calculation is less than the sampling frequency. The LPC parameters are usually calculated once for every 10-25 ms of speech [29]. Transmission of these parameters along with the amplitude and pitch of the error signal every 10-25 ms is an effective means of low bit rate speech transmission. Telephone quality speech is transmitted at 8000 samples per second, or 64000 bits per second (bps) assuming 8 bit samples. LPC speech has been transmitted at bit rates as low as 600 bps although the quality is reputed to be poor [29]. The “speak and spell” toy discussed in Chapter 2 has an average data rate of 1200 bps and results in understandable speech. For us the speech compression qualities of the LPC algorithm are not of major importance as the cost of semiconductor memory is relatively cheap. We still desire to keep the storage requirements to a minimum. This project uses LPC analysis and synthesis with 12 predictor coefficients, $a_1, \ldots, a_{12}$. Each coefficient is represented in double precision (64 bits). For transmission we would also need two extra double precision quantities, the pitch and the gain of the error signal. These quantities are calculated for every 25 ms of the speech signal. This results in an effective data rate of 17,920 bps or 2240 samples per second (assuming 8 bit samples). Although this doesn’t compare to the 1200 bps achieved with the “speak and spell” toy, it is still a considerable increase over direct waveform storage.
6.0.4 Determination of Filter Coefficients

The problem that remains is to determine the LPC coefficients. Typically they are chosen so as to minimise the sum of the squared error signal

\[ M = \sum_n e(n)^2 = \sum_n \left[ x(n) - \sum_{k=1}^p a_k x(n-k) \right]^2 \]  \hspace{1cm} (6.4)

Differentiating with respect to the prediction coefficients and setting the derivatives to zero we get

\[ \frac{dM}{da_j} = -2 \sum_n x(n-j) \left[ x(n) - \sum_{k=1}^p a_k x(n-k) \right] = 0 \]

\[ = -2 \sum_n x(n-j)x(n) + 2 \sum_n x(n-j) \sum_{k=1}^p a_k x(n-k) = 0 \]

So

\[ \sum_{k=1}^p a_k \sum_n x(n-j)x(n-k) = \sum_n x(n)x(n-j) \quad j = 1, 2, \ldots, p \]  \hspace{1cm} (6.5)

This gives a set of \( p \) linear equations for \( p \) unknowns.

Let

\[ C_{jk} = \sum_n x(n-j)x(n-k) \]  \hspace{1cm} (6.6)

\( j = 1 \) : \( a_1 C_{11} + a_2 C_{12} + \ldots + a_p C_{1p} = C_{01} \)

\( j = 2 \) : \( a_1 C_{21} + a_2 C_{22} + \ldots + a_p C_{2p} = C_{02} \)

\( j = 3 \) : \( a_1 C_{31} + a_2 C_{32} + \ldots + a_p C_{3p} = C_{03} \)

\[ \vdots \]

\( j = p \) : \( a_1 C_{p1} + a_2 C_{p2} + \ldots + a_p C_{pp} = C_{0p} \)

This gives a matrix equation of the form:

\[
\begin{bmatrix}
C_{11} & C_{12} & \ldots & C_{1p} \\
C_{21} & C_{22} & \ldots & C_{2p} \\
\vdots & \vdots & \ddots & \vdots \\
C_{p1} & C_{p2} & \ldots & C_{pp}
\end{bmatrix}
\begin{bmatrix}
a_1 \\
a_2 \\
\vdots \\
a_p
\end{bmatrix}
= 
\begin{bmatrix}
C_{01} \\
C_{02} \\
\vdots \\
C_{0p}
\end{bmatrix}
\]
So far, we have neglected to specify the limits of the \( n \)-summation. A doubly infinite summation with \( x(n) = 0 \) for \( n < 0 \) and \( n \geq N \) gives

\[
C_{jk} = \sum_{n=-\infty}^{\infty} x(n-j)x(n-k) = \sum_{n=-\infty}^{N-1-|j-k|} x(n)x(n + |j-k|)
\]

Let \( m = |j - k| \) and let

\[
R(m) = \sum_{n=0}^{N-1-m} x(n)x(n + m) \tag{6.7}
\]

This is the standard autocorrelation function discussed in Chapter 4. The matrix equation then becomes

\[
\begin{bmatrix}
R(0) & R(1) & \cdots & R(p-1) \\
R(1) & R(0) & \cdots & R(p-2) \\
\vdots & \vdots & \ddots & \vdots \\
R(p-1) & R(p-2) & \cdots & R(0)
\end{bmatrix}
\begin{bmatrix}
a_1 \\
a_2 \\
\vdots \\
a_p
\end{bmatrix}
= \begin{bmatrix}
R(1) \\
R(2) \\
\vdots \\
R(p)
\end{bmatrix}
\]

There exist several standard methods for determining the prediction coefficients from the matrix equation shown above, such as Gaussian elimination. These operations however have high computational complexity, \( O(p^3) \). The complexity can be reduced to \( O(p^2) \) if we notice the special properties of the matrix. The elements of the matrix are symmetric, and the elements along any diagonal are identical. This type of matrix is called a Toeplitz matrix.

An elegant solution to the system of equations (known as the Yule-Walker equations [13]) is due to Levinson’s and Durbin’s efforts. The algorithm is shown below [20].

\[
E_0 = R(0)
\]
\[ k_i = - \left[ R(i) + \sum_{j=1}^{i-1} a_j^{(i-1)} R(i - j) \right] / E_{i-1} \]

\[ a_i^{(i)} = k_i \]

\[ a_j^i = a_j^{(i-1)} + k_i a_j^{(i-1)}, 1 \leq j \leq i - 1 \]

\[ E_i = (1 - k_i^2) E_{i-1} \]

The equations with superscripts in parenthesis are solved recursively for \( i = 1, \ldots, p \). That is, \( a_j^{(i-1)} \) is the value of \( a_j \) at step \( i - 1 \). The intermediate quantities \( k_i, 1 \leq i \leq p \) are known as reflection coefficients, or partial correlation coefficients.

In Golub and van Loan, the algorithm is presented differently. They do not retain the vectors \( E \), and \( k \) as we do. The use of the vector \( k \) is discussed in the following section.

### 6.1 Linear Predictive Synthesis

We may easily reproduce the speech signal from the error signal and the LPC coefficients using

\[ x(n) = e(n) + \sum_{k=1}^{p} a_k x(n - k) \tag{6.8} \]

The error signal is unavailable however. Instead it is parameterized into the source type, voiced or unvoiced, the pitch (if voiced), and the amplitude. For voiced speech, the error signal is reproduced using an impulse repeated at the fundamental frequency of the voiced sound. For unvoiced sounds, the error signal is represented by “white noise”.

LPC coefficients can often be unstable, that is, they cause the re-synthesized speech to oscillate wildly [21]. Instability is often caused by truncating the coefficients for transmission. To resolve this problem, the LPC coefficients can be replaced by the reflection coefficients. These reflection coefficients are generated as an intermediate step in the Levinson Durbin algorithm for solving the matrix equation for the autocorrelation method of Linear Prediction. The term reflection coefficient comes
from transmission line theory. The reflection coefficients are related to the reflection and transmission parameters at the junctions of an acoustic tube model of the vocal tract [20]. The reflection coefficients are bounded by ±1 for stable filter, and thus it is easy to detect an instability in the output.

6.1.1 Lattice Filters

There is a single stage procedure for reproducing speech directly from reflection coefficients [29]. The structure used for speech re-synthesis is called a lattice filter, shown in Figure 6.1. The diagrams and notation described here are taken from Witten [29].

![Figure 6.1: Lattice Filter for Reproducing a Speech Wave from Reflection Coefficients](image)

The excitation source enters the filter at the top left, and is propagated along the top forward signal path. It is modified by past output values propagated along the bottom backward signal path. The lattice filter is divided into a number of stages equal to the number of reflection coefficients (12 in this case). Each stage consists of two multiplications, two additions and one unit delay. The unit delay is represented in z-transform notation as $z^{-1}$. A single stage of the lattice filter is shown in Figure 6.2.

![Figure 6.2: A Single Stage of the Lattice Filter](image)
The signal flow is defined by the following two relationships:

\[ y^+ = x^+ - k z^{-1} y^- \]  
\[ x^- = k y^+ + z^{-1} y^- \]  

Put simply, this means that the \( y^+ \) output is the sum of the \( x^+ \) input and the \( y^- \) value delayed one time step, multiplied by a reflection coefficient. Similarly \( x^- \) is the sum of the \( y^+ \) output and the \( y^- \) value delayed one time step, multiplied by the negative of the same reflection coefficient. The unit delay operator, \( z^{-1} \), may be removed from the above operations by rewriting the lattice stage inputs and outputs in terms of their values at time \( n \) and \( n-1 \) as shown below.

\[ y^+(n) = x^+(n) - k y^-(n-1) \]  
\[ x^-(n) = k y^+(n) + y^-(n-1) \]

Representing the function of a single stage as a matrix gives:

\[
\begin{bmatrix}
X^+ \\
X^-
\end{bmatrix} =
\begin{bmatrix}
1 & -k z^{-1} \\
-k & z^{-1}
\end{bmatrix}
\begin{bmatrix}
Y^+ \\
Y^-
\end{bmatrix}
\]  

(6.13)

This allows the definition of a multistage lattice, such as the four stage lattice used in the SIFT algorithm discussed in Chapter 4. The four stage lattice filter is shown below in Figure 6.3.

![Four stage Lattice Filter](image_url)

Figure 6.3: Four stage Lattice Filter as used in the SIFT pitch tracking algorithm

The four stage lattice filter is described by the following matrix equations:
\[
\begin{bmatrix}
X_5^+ \\
X_5^-
\end{bmatrix} =
\begin{bmatrix}
1 & -k_4 z^{-1} \\
-k_4 & z^{-1}
\end{bmatrix}
\begin{bmatrix}
X_4^+ \\
X_4^-
\end{bmatrix}
\]
\[
\begin{bmatrix}
X_4^+ \\
X_4^-
\end{bmatrix} =
\begin{bmatrix}
1 & -k_3 z^{-1} \\
-k_3 & z^{-1}
\end{bmatrix}
\begin{bmatrix}
X_3^+ \\
X_3^-
\end{bmatrix}
\]
\[
\begin{bmatrix}
X_3^+ \\
X_3^-
\end{bmatrix} =
\begin{bmatrix}
1 & -k_2 z^{-1} \\
-k_2 & z^{-1}
\end{bmatrix}
\begin{bmatrix}
X_2^+ \\
X_2^-
\end{bmatrix}
\]
\[
\begin{bmatrix}
X_2^+ \\
X_2^-
\end{bmatrix} =
\begin{bmatrix}
1 & -k_1 z^{-1} \\
-k_1 & z^{-1}
\end{bmatrix}
\begin{bmatrix}
X_1^+ \\
X_1^-
\end{bmatrix}
\]

Substituting each equation into the previous results in the following matrix equation:

\[
\begin{bmatrix}
X_5^+ \\
X_5^-
\end{bmatrix} =
\begin{bmatrix}
1 & -k_4 z^{-1} \\
-k_4 & z^{-1}
\end{bmatrix}
\begin{bmatrix}
1 & -k_3 z^{-1} \\
-k_3 & z^{-1}
\end{bmatrix}
\begin{bmatrix}
1 & -k_2 z^{-1} \\
-k_2 & z^{-1}
\end{bmatrix}
\begin{bmatrix}
1 & -k_1 z^{-1} \\
-k_1 & z^{-1}
\end{bmatrix}
\begin{bmatrix}
X_1^+ \\
X_1^-
\end{bmatrix}
\]

The value of \(X_5^-\) is not used and so may be discarded. This gives \(X_{input} = X_5^+\) and \(X_{output} = X_1^+ = X_1^-\), since \(X_1^-\) and \(X_1^+\) are connected at the output. The filter transfer function \(\frac{X_{output}}{X_{input}}\) for particular values of the reflection coefficients can be computed by rearranging the matrix equation. This, however, becomes quite unwieldly for lattice filters longer than three stages. Appendix C shows the calculation performed using Maple.
Figure 6.4 below shows a spectrograph of the output of the LPC algorithm resynthesizing eight common vowels sung in isolation. The sung vowels are the same as those used in Figures 3.8 and 5.12 in Chapters 3 and 5. This enables a direct comparison between the three spectrographs and verifies that the LPC algorithm preserves the spectral nature of speech sound upon resynthesis. You will notice that the spectrograph for the Linear Predictive Coded vowels appears “cleaner” than the recorded and PSOLA synthesized vowels. The harmonics of the fundamental frequency are much more pronounced and more clearly defined than the those of the previous two spectrographs.

![Spectrograph of LPC synthesis of eight common vowels.](image)

Figure 6.4: Spectrograph of LPC synthesis of eight common vowels.
Chapter 7

Implementation and System use

7.1 Hardware Platforms

The software for this thesis was developed on a variety of Sun SPARCStations. The system has been tested on SPARCStation IPX, SPARCStation 2, and SPARCStation 10 workstations. The SPARCStation IPX and SPARCStation 2 workstations support 8 bit logarithmic (either μ-law or A-law) recording and playback at an 8 kHz sampling rate. The SPARCStation 10 supports a much greater variety of audio formats, with up to 16 bit linear samples at a 48 kHz sampling rate. Chapter 3 discusses sampling rates and the differences between linear and logarithmic sampling. For more information regarding the workstation sound hardware see the Sun audio on-line manual pages using the man command. Type: man audio for general information; man audioamd for information about the AM79C30A chip used in the SPARCStation IPX, and SPARCStation 2 machines; and man dbri for information about the Dual Basic Rate ISDN chipset used in the SPARCStation 10 machine.

7.2 History and Overview of the System

Parts of the system were originally developed for use as an electronic Sight Singing Tutor [22]. Sight Singing is the process of singing an unfamiliar piece of music when
presented with only a musical score and the pitch of the starting note. The system would allow the user to record his or her rendition of the notes displayed, and then process the recorded input to produce another sequence of notes corresponding to the notes sung by the user. The system would produce a numeric comparison of the two note sequences, using a dynamic programming approach. Almost all of the original software is preserved in the program developed for this thesis. The software was ported from the Macintosh, on which it was developed, to the Sun workstations. The user interface of course changed dramatically but the pitch tracking algorithm (Gold/Rabiner—discussed in Chapter 4), and the dynamic programming code were preserved. In fact, the entire functionality of the Sight Singing Tutor is preserved in the new system.

The current system is much more than a Sight Singing Tutor however. It has evolved into a signal processing workbench providing many of the tools required for speech analysis/synthesis. Obviously, in this case the emphasis is placed on singing synthesis, but the fundamentals of speech and singing analysis/synthesis are similar. The workbench provides the facilities for recording and playback of multiple buffers of sound. Each buffer is labeled and switching between buffers is a simple matter of selecting the buffer name from a scrolling list. RMS (Root Mean Square) analysis allows determination of signal power which can be used to automatically find the start and end points of each note in the sample. Frequency analysis both in the time and frequency domains is possible. The Gold Rabiner algorithm is provided to perform frequency analysis (in the time domain) on the currently selected buffer. This produces a graph of fundamental frequency vs. time. As with the sample buffers, multiple frequency and RMS power buffers are supported also, accessed via a scrolling list. Frequency analysis in the frequency domain is possible using Fourier Transform routines. A multi-colour sound spectrograph, constructed from time delayed Fourier transforms, provides a multiple frequency vs. time graph. Again, multiple spectrographs are supported via a scrolling list. Appendix D gives a detailed description of the Discrete Fourier Transform and its use for constructing the sound spectrograph.
The current sample buffer can be low pass filtered using the filter described in Appendix A. The most important developments are the implementations of the PSOLA and LPC algorithms. The PSOLA algorithm was discussed in Chapter 5, and the LPC algorithm was discussed in Chapter 6. The interfaces to each of these algorithms will be discussed in detail in later sections. Two of the most important interfaces to the current system are the musical stave where the music to be sung by the system is displayed and the phoneme window, where the phonetic elements of the computer’s singing voice are captured and displayed. The following section will explain the parts of the system in detail.

7.3 Detailed System Description

7.3.1 The sample buffer

The sample buffer, despite its name, is more than a buffer for storing sound for playback. It provides a visual interface for viewing sound waveforms as well. Sounds can be saved to disk for permanent storage, and retrieved for use at a later time. Figure 7.1 shows a picture of the sample buffer interface.

![Sample Buffer Interface](image)

Figure 7.1: User interface to the sample buffer
The interface provides much information for the user. The smaller window on the left shows a closeup zoom of the waveform where the user’s cursor is positioned (the cursor is not visible in the Figure). In this case the cursor points to the region inside the two vertical lines in the larger window which shows the full sound waveform. Notice these same two vertical lines in the zoom window. These vertical lines show the current range of the buffer. All of the system’s functions which act on the sample buffer (filtering, frequency analysis, etc.) act only on the region selected by these two lines. The positions of the lines can be altered using the mouse, or by entering the range start and end values. The scrollbar along the bottom of the large window allows different sections of the waveform to be viewed. The zoom slider on the right hand side of the large window controls the extent of the zoom factor of the large window. Setting a high zoom value shows a smaller section of the waveform in more detail. A low zoom value lets the user see more of the waveform, but in less detail. The information at the top of the window shows the name of the waveform on disk, the total length of the waveform (in seconds), the current selection range (in seconds), and the selection size (in bytes). An information area allows the user freedom to label the sample to provide more information about its origins. Control sliders allow the user to change the play and record volumes. These controls affect the entire system, not just the sample buffer. The row of buttons above the waveform windows allow (from left to right) current buffer selection; play/record controls; current buffer output, and full range selection. The buffer output dumps the waveform to an ASCII file suitable for plotting by a graphing package such as Gnuplot. The waveforms in this thesis were produced in this way.

7.3.2 The audio device

Audio output under Unix is achieved by writing sound data to the audio device. Devices in a Unix operating system are treated like pseudo-files. The audio device file is `/dev/audio`. So, for example, a file of raw sound data could be played using the unix command: `cat sound.data > /dev/audio`, provided the audio device was
configured to the correct audio format.

**Opening the audio device**

The audio device is opened just like an ordinary file. A file descriptor is returned if the audio device is successfully opened. For example, to open the audio device for output only the following code segment would be used:

```c
audio_fd = open("/dev/audio", O_WRONLY);
```

The audio device can be accessed both synchronously and asynchronously. For synchronous access read and write accesses to the audio device will block until the data transfer is complete. Asynchronous access allows a read or write call to return immediately before the data transfer is complete. For example, to open the audio device for synchronous output we would use this code segment:

```c
audio_fd = open("/dev/audio", O_WRONLY | O_SYNC);
```

**Configuring the audio device**

Before input or output of audio data, the audio device must be configured for the required mode of operation. The operating characteristics include the sampling rate, the number of bits per sample, and the encoding mechanism. The audio device is configured using the Unix `ioctl` system call. A file descriptor is needed to configure the audio device, so the file must first be opened. For example, to set the audio device up with 8 kHz sampling rate, one sound channel, and 8 bit μ-law samples, the following C code is executed:

```c
audio_printf_t play, rec;
audio_info_t info;
int audio_fd;

/* open the audio device */
audio_fd = open("/dev/audio",O_RDWR, O_SYNC);
```
/* set up the play and record configurations */
play.sample_rate = 8000;
play.channels = 1;
play.bytes_per_unit = 1;
play.samples_per_unit = 1;
play.encoding = AUDIO_ENCODING_ULAW;
rec = play;

/* clear the audio info structure to non harmful values */
AUDIO_INITINFO(&info);

/* set the play/record configurations */
/* in the audio info structure */
info.play = play;
info.record = rec;

/* initialize the audio device */
ioctl(audio_fd, AUDIO_SETINFO, &info);

Playing and Recording data

With a properly opened and configured audio device, synchronous playing and recording of sound is trivial. This is a simple matter of writing to and reading from the audio device. To play a buffer of sound the following source code is used:

write(audio_fd, buffer, size);

To record a buffer of sound the following source code is used:

read(audio_fd, buffer, size);

where audio_fd is a handle to an open, and correctly configured, audio device; buffer is a pointer to a buffer of eight bit samples, and size is the number of samples
to read or write.

Asynchronous recording is not as simple as this however. Although the same calling sequence is used, the size of the read or written data is undetermined. Both the read and write procedures return the size of the data actually read or written. This is probably not the same as the requested size since the procedure calls return immediately. For asynchronous operation, the audio device can be requested to send a signal to the calling process when it has more data available (recording), is ready for my data (playing), or undergoes a change of state (possibly by another process). The notification request is made with the following procedure call:

\texttt{ioctl(Audio\_fd, I\_SETSIG, S\_INPUT\_S\_OUTPUT\_S\_MSG)}

The signal sent by the audio device is the SIGPOLL signal. The calling process sets up a handler procedure to catch this signal and perform the required action, which is usually either writing more data to the audio device, or reading more data from it. The advantage of asynchronous procedure calls is that the calling process need not wait while the audio device is recording or playing data. It is free to perform other tasks. The sample buffer record and play buttons provide asynchronous recording and playback. This allows interactive user control of recording and playback such as pausing or halting a recording or playback session in midstream.

### 7.3.3 PSOLA Analysis/Synthesis

The PSOLA algorithm was discussed in Chapter 5. This section discusses the user interface to the PSOLA algorithm, and some specific implementation considerations. The interface to the PSOLA algorithm is shown in Figure 7.2. As with all functions in the system, the PSOLA algorithm operates on the currently selected sample buffer range. So the full sample, or just a small section of it, could be captured and represented in granular form. The interface in Figure 7.2 is for experimentation only, that is, the granular information is not retained. The interface allows the user to select synthesis options and then analyze and resynthesize the selected region of the sample buffer. The newly synthesized waveform is then added to the sample buffer list, and
displayed in the sample buffer ready for subsequent playback or analysis.

![PSOLA options](image)

Figure 7.2: User interface to the PSOLA algorithm

The interface supports many options which were developed primarily to improve the realism of the output of single grain PSOLA synthesis. They can also be applied to multi-grain synthesis, although they appear to do more harm than good. The interface allows the user to choose to add Frequency or Amplitude modulation to the PSOLA synthesis. In this context, *modulation* means to modify the constant frequency and amplitude by a fixed or random amount. This differs slightly from the electrical engineering concept of carrier wave modulation, or modulation by a frequency or amplitude envelope.
Frequency Modulation

The user may select the modulation probability from 0 to 100%. Selecting a 0 probability is equivalent to turning frequency modulation off completely. This means that for probability $n$, each grain has an $n\%$ chance of being modulated. Frequency modulation, in this context, means to adjust the spacing between successive grain centers, so in fact it is the distance between grains that is modulated and not the grain itself. In Fixed Frequency Modulation, the grain center distance ($\Delta c$) is multiplied by the user specified percentage, $p\%$ (from the Fixed Modulation Percentage control), giving a new grain center distance:

$$\Delta c_{\text{new}} = \frac{p}{100} \times \Delta c$$  \hspace{1cm} (7.1)

where

$$\Delta c = \frac{1}{\text{Pitch}}$$  \hspace{1cm} (7.2)

The Pitch is also user specified at the top of the PSOLA interface. Random Frequency Modulation modifies the grain center distance by a random amount. The maximum modification is user specified by the Random Modulation Percentage control. So for example, setting the slider to 15% allows a frequency deviation of plus or minus 15%. The actual deviation is random within this range.

Amplitude Modulation

As with Frequency Modulation, Amplitude Modulation takes place with user specified probability. Each grain has $n\%$ probability of amplitude modification. Again, two types of modulation are supported, fixed and random. Both types modify the grain itself, or more specifically, the values of the grain data. Let the grain data elements be represented by $g_i, i = 1..N$, where $N$ is the grain length. For Fixed Amplitude modulation:

$$g_{i_{\text{new}}} = \frac{\text{gain}}{100} \times g_i$$  \hspace{1cm} (7.3)

where the gain percentage is taken from the amplitude Fixed Modulation Percentage control. Random amplitude modulation is similar except the gain modification factor,
gain is random. The upper bound of the random gain modulation is given by the *Random Modulation Percentage* control. So for example, setting the slider to 10% allows a maximum gain modification of plus or minus 10%. It should be noted that all values in the grain are modified by the *same* amount. A random gain factor is generated in the range of plus or minus 10% and then applied to the grain as a whole.

### 7.3.4 LPC Analysis/Synthesis

The theory behind Linear Predictive Coding was discussed in detail in Chapter 6. This section discusses the implementation specific considerations and the user interface. The user interface for specifying LPC analysis/synthesis options is shown below in Figure 7.3.

![LPC Analysis/Synthesis Interface](image)

**Figure 7.3: User interface to the LPC algorithm**

This interface operates similarly to the PSOLA interface described in the previous section. Again the selected range of the sample buffer may be analyzed and resynthesized, creating a new sample buffer. The internal LPC representation of the waveform is discarded. The interface allows the user to modify LPC parameters and resynthesize a waveform to determine a successful combination of operating parameters.
Recall from Chapter 6 that the LPC algorithm operates on windowed speech data. The user may alter the analysis window length, the input frame length, and the output frame length using the first three text edit controls in the window. In this case the analysis window length is set to 300 samples, with an input and output frame length of 200 samples. This corresponds to the discussion in Chapter 6. An analysis window length of 300 and an input frame length of 200 samples gives an overlap of 100 samples (i.e. 100 samples from the previous analysis window are used in the current analysis window). In this case the input to output frame length ratio is 1, (number of input samples = number of output samples). Increasing the number of output samples over the number of input samples results in slower speech/singing output, while decreasing the number of output samples results in faster speech/singing. This is useful for fitting a given LPC encoded signal into a fixed time span (as is the case when singing a word throughout the duration of a note).

The interface also allows the option to pre-analyze a number of frames. Although the LPC coefficients of these frames will not be used for resynthesis, calculating them anyway may be useful to minimise any initial transitional effects when analyzing a small number of frames. As with any analysis technique which relies on past information, initial transitional instabilities may cause inaccuracies in the coefficients of the first one or two frames.

The type of pitch tracker used by the LPC algorithm may also be specified. Although the SIFT algorithm is recommended for use with the LPC algorithm, one of two versions of the Gold Rabiner pitch tracking algorithm may also be used. Neither of these works as effectively as the SIFT algorithm however. It is interesting to hear the effects of inaccurate pitch tracking on LPC encoded singing or speech. Both implementations of the Gold Rabiner algorithm make frequent errors in the fundamental frequency estimation.

The remaining LPC controls are to vary the effect of the LPC synthesis. The first is the whisper control. Selecting whisper ignores all pitch information, setting the fundamental frequency to zero. This corresponds to no movement of the vocal
chords whatsoever. When a person whispers, the vocal chords do not move at all. The whisper attenuation factor controls the loudness of the whisper. The smaller the value, the louder the whisper. This is necessary because a whisper with the same amplitude as normal speech sounds out of place. Obviously the whisper controls are included for effect only, since they have no practical use in a singing synthesis system. It is interesting to hear the effect of removing pitch information from an LPC coded signal however. The interface also supports pitch modification. Multiplying the pitch value of each frame by the pitch shift factor can either raise or lower the pitch of the speech. Monotonic speech is possible by setting the pitch of every frame in a waveform to the same value. The remaining two controls on the LPC interface are fairly self explanatory. The Analyze/Synthesize button on the left initiates the analysis synthesis process and the CopyToSampleBuffer button on the right copies the synthesized output to the sample buffer where it can be viewed and played.

7.3.5 The Musical Stave

Until now the techniques discussed have been applied to speech only. The system’s application to music has been largely neglected. Chapter 3 discussed the prosodic features of speech. Prosodic features include pitch and rhythm. Most speech synthesis systems perform poorly due to improper treatment of these prosodic features. Many systems do not assign changing pitch to synthesized speech, resulting in a monotonic/robotic sound. A singing synthesis system removes much of the burden of prosodic features from the system designer. On a musical stave, pitch and rhythm are expressed explicitly in the lengths and pitches of the notes. No other external specifications of prosodic features are necessary at all! Actually, this is not strictly true. A piece of music may often contain much more than just the notes on a stave. Often other cues such as slurs between notes, vibrato, relative volume and emphasis are specified also. The system described here uses only a simple specification of notes and rests. Despite its simplicity however, it shows promising results.

The Musical Stave is the system’s interface for the user to specify the notes and
words of a song to be sung. As a holdover from the Sight Singing Tutor, the stave maintains two separate note lists; the melody and the voice note lists. The melody note list is where the song to be sung by the user would be displayed, and the voice note list would contain the computer’s transcription of the user’s singing. Both note lists are retained in the current system so that it may be used as both a Sight Singing Tutor and a singing synthesis system. Joining these two systems together is quite useful since the user may sing a song to the system, and after correcting any transcription errors, assign words to the notes so the system may sing the song. The system can synthesize either note list, but only the voice note list allows the user to edit individual notes.

The Musical Stave with a song on the voice note list is shown in Figure 7.4 below:

![Musical Stave](image)

**Figure 7.4: The musical stave**

The stave shows the contents of both the voice and melody note lists. In this case the melody note list is empty, and the voice note list contains the words and notes for the “happy birthday” song. Both note lists can contain rests also, a 2/16 rest being shown at the far right of the voice note lists. It should be noted that only a very small portion of the note lists can be shown at one time. The slider at the bottom of the stave allows the user to scroll left and right through the song.
7.3.6 Entering a song

There are several ways to enter a melody, perhaps the most appealing is to sing it to the system yourself. The user may record part or all of a song using an external microphone attached to the audio device. There are some difficulties with this technique however. First, it is essential that the user enters each note singing ‘Du’. The system uses the partial alveolar stop on the ‘D’ to segment the user input into its individual notes. Even so, the system occasionally makes mistakes and skips notes. This requires the user to correct the notes by hand using the note editor shown in Figure 7.5. For this reason it is advisable to enter only a few notes at a time.

![Figure 7.5: The note editor interface](image)

The note editor is pulled up when the user clicks on a note from the voice note list. Figure 7.5 shows the interface pulled up after clicking on the third note in the voice note list of the stave from Figure 7.4. The interface indicates whether the element being edited is a note or a rest. In the case it is a note, its Midi note number is displayed indicating the note’s pitch. Midi is an electronic music standard used for interfacing electronic instruments. The pitch of the note can be altered by changing the midi note number. An increase/decrease of one corresponds to an increase/decrease of the pitch by one semitone. The note duration is displayed and can also be altered by clicking on the picture of the note corresponding to the new
duration. Here the note is 3/16 in length—a dotted quaver. A similar technique allows a change in rest duration. For example, to change the third note in the song to a quarter rest the user would click on the third note to bring up the note editor interface, change the note type to ‘Rest’ and then click on the fourth rest from the left in the rest types selector. Nothing is actually changed until the ‘Apply’ button is clicked. New notes can also be added following the currently edited note (using the ‘InsertAfter’ button), and also at the end of the voice note list (using the ‘Add’ button). The Phonetic Transcription text field of the note editor interface shows the phonetic text associated with the currently edited note. The phonetic text tells the computer what to sing during the note. Here, the system will sing ‘birth’ from the word birthday.

Another technique for entering songs is to use an electronic keyboard attached directly to the audio device. Essentially this is the same as the first technique except an external microphone is not necessary. The user must still record the input using the sample buffer interface shown in Figure 7.1. Again, recording only a few notes at a time is advisable, since considerable editing is usually required. It is important to note that careful selection of the instrument sound is necessary. Some electronic keyboard sounds are inappropriate for a number of reasons. Some sounds to avoid have the following properties which confuse the automatic transcription system: rapid note decay, more than one major frequency component, excessive vibrato, and/or high frequency. Desirable sound qualities are low frequency, rapid attack combined with relatively large amplitude at the beginning of the note, and slow amplitude decay to the end of the note. This will result in a low frequency signal with high relative RMS power at note boundaries. It may be difficult to find a synthesizer sound with these properties, and some experimentation may be necessary.

### 7.3.7 Choosing and Recording Phonetic Elements

At this stage the reader may be under the misconception that once a song is entered, with words associated with each note, the system can be directed to start singing.
Unfortunately this is not even remotely the case. Although the electronic singer has
information about the pitch and duration of the notes to be sung, it has no information
about the structure of the words themselves. Human singers store information in their
brains for the articulatory movements associated with the words they read from the
sheet music, which they can then reproduce for the required pitch and duration. The
system also needs such a database and a means of synthesizing the required sound at
the pre-specified pitch and for the appropriate duration. Chapters 5 and 6 discussed
the means of synthesis, but the knowledge (database) of the word structure is still
unknown. It is the user’s responsibility for creating such a database.

There are both advantages and disadvantages for allowing the users to create their
own database. The most important advantage is that multiple databases may be con-
structed allowing for different singing styles and voices. Most people with interest in
such an electronic singing system would probably not want to hear only this author’s
voice. Perhaps the greatest disadvantage however, is the considerable difficulty of
synthesizing the singing voice well. The choice of phonetic elements is much more
critical than the choice of the synthesis system itself. The phonetic elements and the
way they are recorded and subsequently extracted from the recording, influence the
resulting singing quality considerably. It is for this reason perhaps more than any
other that speech synthesis is regarded by many as a very difficult area of computer
science. Allowing user-defined databases shifts some of the burden of speech synthesis
from the system designer to the users. Obviously this need not always be the case;
some users may never design their own database preferring to use one designed by
someone else. The facility exists for saving/loading of phonetic elements to/from disk
to allow ease of data sharing.

The user is provided with an interface to use in conjunction with the sample
buffer interface for extracting phonetic elements. The interface is shown in Figure 7.6
below. Controls are provided for loading and saving and clearing the entire phonetic
element list, as well as playing and deleting individual phonetic elements. The large
window at the base of the interface shows the currently selected phonetic element.
Other phonetic elements can be selected by clicking on their name in the scrolling list. The currently displayed phonetic element may also be copied into the sample buffer for more detailed analysis. Each phonetic element has a user specifiable name and description.

![Phonetic Data Interface](image)

Figure 7.6: The phonetic data interface

The interface shows a list of phonemes, one of which can be displayed at a time. Chapter 3 described phonemes as “... the most basic units of speech. They cannot be resolved into smaller identifiable units...”. For our purposes this need not be the case, and in fact should not be the case, for reasons discussed later. Here, for example, the phonetic element is an entire word ‘birth’. Again, the choice of the phoneme size is dependent on the user’s requirements. Phoneme sizes need not be consistent even within the same database. This is illustrated in Figure 7.6 which consists mostly of entire words with the exception of the word ‘happy’ which is split into two phonetic elements ‘ha’ and ‘pe’. Of course this is necessitated by the fact that happy is spread
over two notes, but this need not always be the case. Words within a single note can be split into smaller phonetic elements also. The word ‘birth’ for example could be split into three phonetic elements, a plosive ‘b’; a long vowel ‘er’; and the dental fricative ‘th’. Other users may choose to break the word down further and possibly include a retroflex ‘r’ sound.

Capturing Phonetic Elements from the Sample Buffer

Extracting a phonetic element, be it a word or something smaller, from the sample buffer is termed ‘capturing’ the phonetic element. Obviously the user must record a word, phrase, or sentence first. The size of the sample buffer is unimportant. It is important to record a phonetic element within a context. For example to capture the dental fricative from the word ‘birth’ it is unrealistic to simply record a lone ‘th’ sound and expect a natural sounding word when it is combined with other phonetic elements also recorded out of context. After all, representing words and sentences as a series of phonetic elements is only an approximation to connected speech. Connected speech by definition has no inconsistencies between so called ‘phonetic elements’. Speech flows gradually from one sound to another just as the vocal articulators generating the speech flow gradually from one position to the next. This is the largest single problem in speech synthesis today, and one of the major reasons why synthesized speech quality is so poor. The influences of adjacent sounds on each other are known as co-articulation effects. These co-articulation effects result in a sound taking on some of the characteristic sound of its neighbour. In order to create high quality synthetic singing or speech, these co-articulation effects must be minimised.

The effects are greatest at the phonetic level (using the definition from Chapter 3, but persist even between individual words. Co-articulation effects between words are more of a prosodic nature and are less noticeable in a singing synthesis environment since the prosody of the song is fixed to a certain extent by the notes on the stave. A way to minimise co-articulation effects is to attempt to split phonetic elements in the middle of a phoneme where co-articulation effects are less pronounced.
For example, when splitting ‘birth’ into smaller phonetic elements, the split could be made in the middle of the long vowel associating some vowel sound with the initial plosive and some with the final dental fricative. A dot notation is used to separate the phonetic elements within a word, i.e. ‘bur:urth’. A longer vowel could be constructed by changing the phonetic text to: “bur.ur.th. Please note, the long ‘ur’ vowel contains no retroflex sound at all, despite the ‘r’ in the phonetic text. This illustrates the difficulty of selecting a phonetic alphabet for which the text of each phonetic element visibly reflects its sound. A retroflex ‘birth’ might be transcribed as ‘bur.ur.urrth’. The last phonetic element in the word is quite complex containing what might be considered as three separate phonemes. It is unclear however where to draw the dividing line between them so as to reduce the effects of co-articulation. Users who believe they can create the ideal set of phonetic elements have the means to experiment however. It is certainly an area which merits much closer evaluation and experimentation than is possible with the time available for this thesis.

Once the user has selected a portion of the sample buffer to capture and decided on a name for the resulting phonetic element, capturing the phoneme is trivial. The user should first determine the method of synthesis, either LPC or PSOLA, from the pull-down menu on the phoneme interface (Figure 7.6). Multi-grain PSOLA analysis/synthesis is listed as PsolaN (N grain PSOLA synthesis). Then, clicking on the button labeled capture causes the system to automatically encode the selected region of the sample buffer using the selected encoding technique (LPC or PSOLA) and store this in the phonetic element list ready for subsequent resynthesis.

Alternatively, the system can automatically capture multiple phonetic elements from multiple files using a capture script. The format for a capture script and an example capture script are shown in Appendix E. The capture script tells the system to load in specified audio files and extract the named phonemes from the specified positions in the capture script. For example, from the file Eleanor2.au the phoneme ‘RF’ would be extracted from the range 8687 to 9486. The duration of this phoneme is almost 100 ms, making this one of the shortest phonemes in the capture script.
The durations range from 100 ms to nearly 1000 ms. The capture scripts greatly decrease the database construction time, especially if the phonemes and offsets are previously known. For the case where two separate phonetic databases are required, one encoded using LPC and the other encoded using PSOLA, the capture scripts are invaluable. The capture scripts ensure that the same raw data are used for creating each phonetic element regardless of the encoding scheme.

### 7.3.8 The Synthesis Engine

Having installed a database of phonetic elements, either by loading a previously created database, capturing each individually, or using a capture script, the user is now in a position to play the song. From the user's point of view this is a simple matter of pressing a button to initiate the synthesis engine.

The synthesis engine consists of two processes working cooperatively and communicating through a Unix pipe. One process scans the note list synthesizing the phonetic text associated with each note at the appropriate pitch and duration. Meanwhile the other process waits for data to arrive at its end of the pipe. When the first process has finished each note synthesis it sends the synthesis data in \( \mu \)-law audio format, together with the data length, into the pipe. The waiting process reads this data from the pipe and spools it to the audio device without modification.

![Diagram](image)

Figure 7.7: The note synthesis engine

It then enters a wait state until more data is available. Reading and writing
data to the pipe is similar to reading and writing data to the audio device, the same procedure calls are used. The only difference is that the file descriptor points to an open pipe rather than the audio device. The synthesis engine may be viewed conceptually in Figure 7.7.

The synthesis method is determined by the encoding technique used for the phonetic element. Each phonetic element could be encoded as either PSOLA or LPC, but all elements within a single note must use the same encoding technique or the note will be skipped. This restriction is imposed to ensure consistency throughout the whole note. It prevents discontinuities in the output waveform, which would otherwise cause pops and clicks in the singing voice.

Once the encoding technique is established and the phonetic text checked for consistency, the note synthesis begins. The lattice filter is used to resynthesize the LPC encoded phonetic elements, see Chapter 6. Granular synthesis is used to resynthesize the PSOLA encoded phonetic elements, see Chapter 5.
Chapter 8

Conclusions and Future Development

From its beginnings as an automatic transcription system for use in sight singing instruction the system has developed into a real time singing synthesis system with facilities for spectral analysis and fundamental frequency analysis among others. When development began I was warned against the possibility of building a “swiss army knife” — a system with too much functionality. An experimental digital signal processing system however tends to grow with a life of its own, to develop functionality as it is needed. While some of the functionality may appear superfluous to some users it is provided to allow experimentation for those who wish to build their own phonetic databases.

A similar justification may be applied to the implementation of two encoding techniques, LPC and PSOLA, instead of only one. Primarily, it allows the user to use the encoding scheme of his/her own choice. Linear Predictive Coding is a well established signal processing technique for analysis and synthesis of the human voice. Pitch Synchronous Overlap Add is a much more recent technique and as such is less well known. The purpose of this study was not to compare the two algorithms to determine which is superior to the other. Studies of this type have already been performed [28, 23]. In subjective tests involving human subjects, van der Sluis [28] was unable to determine
a clearly preferred algorithm. He concluded, at a 95% confidence level, that there is no significant difference between LPC and PSOLA. Hamon [23] judged PSOLA superior to LPC in objective tests. There are obvious tradeoffs however. LPC results in a compressed representation of the phonetic elements, which is desirable for a large database, or for speech/singing transmission. PSOLA, however, has much greater storage requirements. PSOLA requires less computation on resynthesis, using only integer arithmetic.

The importance of both algorithms is their ability to modify the pitch and duration of speech/singing. Without this property neither algorithm would be appropriate for this application. Both are time domain algorithms, not requiring the computationally expensive transformation to the frequency domain for processing. Both therefore are fast enough for real time synthesis on a low cost microcomputer.

There are many avenues for future work on this project. Studies could be performed comparing the subjective quality of different choices of phonetic elements, perhaps resulting in a standard set of phonetic elements for singing synthesis. An international standard exists for phonetic transcription from speech [24]. This seems inappropriate for speech or singing synthesis however because it does not account for co-articulation effects between phonetic elements.

Another possibility of particular interest is to use Midi to interface the system with other electronic musical instruments to create a completely electronic band. Midi is capable of interfacing computers, electronic keyboards, electronic drum machines and many other instruments for automated music production. Although the electronic singer may not be of sufficient quality for lead vocals, it may certainly be capable of background vocals.

There are possibilities for commercial development of the ideas implemented in this thesis for entertainment purposes. Special purpose hardware for singing synthesis could be used in singing toys, greeting cards, and children’s games. There is already a large market for sound-making toys for children. The quality is usually poor and there is still much room for improvement.
Appendix A

The Low Pass Filter

Digital filters are an important aspect of Digital Signal Processing. The Low Pass filter, which will be discussed in detail in this appendix, is used frequently in fundamental frequency estimation (see Chapter 4). The ideal low pass filter removes all frequencies above a specified cutoff frequency from a signal. This is illustrated by the frequency domain graph in Figure A.1 below.

![Frequency Domain Graph](image)

Figure A.1: Ideal 900 Hz Low Pass Filter

An ideal filter is not possible to implement in practice.

The low pass filter used in this project is defined by the following equations:
\[ y_n = a_5 u_n - a_3 y_{n-1} - a_4 y_{n-2} \]  
(A.1)

where

\[ u_n = a_2 x_n - a_1 u_{n-1} \]  
(A.2)

This is a two stage digital filter as shown in the block diagram in Figure A.2. \( x_n \) is the filter input at time \( n \), \( u_n \) is the output of the first stage of the filter at time \( n \) (equation A.2), and \( y_n \) represents the output of the second stage at time \( n \) (equation A.1).

![Block diagram for the digital low pass filter](image)

Figure A.2: Block diagram for the digital low pass filter

Recall, from Chapter 3, the discussion of sampling a time varying waveform. A digital filter takes as input a sampled time varying waveform, and produces a filtered version of this same waveform. The filter takes one sampled datum, \( x_n \), at a time and produces its filtered replacement datum, \( y_n \). Filtering an entire waveform requires replacing each sample with its filtered counterpart.

### A.1 Determination of the Filter Transfer Function

The transfer function determines the relationship between the filter input and its output. This is usually expressed in the frequency domain. Obviously equations A.2 and A.1 constitute a transfer function for the filter in the time domain. A time domain transfer function is little use to us however since it does not describe the frequency modification characteristics of the filter. Figure A.1 shows an ideal transfer function (in the frequency domain) for a 900Hz low pass filter. All frequencies are multiplied by unity (passed through unchanged) below the filter cutoff, and multiplied by zero
(removed) above the filter cutoff. The purpose of this section is to determine the frequency domain transfer function for the filter defined by equations A.2 and A.1.

Since this is a two stage filter, it is sensible to analyze one stage at a time. Let $H_1(\omega)$ be the transfer function for the first stage of the filter, and $H_2(\omega)$ be the transfer function for the second stage of the filter. This gives a transfer function for the entire filter of

$$H(\omega) = H_1(\omega)H_2(\omega) \quad (A.3)$$

To determine the frequency domain transfer function we assume the input is a time varying waveform with only one frequency component. Therefore we write:

$$x_n = e^{i\omega n} \quad (A.4)$$

This is a complex quantity with both real and imaginary parts. The famous Euler identities state:

$$\cos \omega + i \sin \omega = e^{i\omega} \quad (A.5)$$

$$\cos \omega - i \sin \omega = e^{-i\omega} \quad (A.6)$$

In both cases $\cos \omega$ is the real part, and $\sin \omega$ is the imaginary part.

### A.1.1 Stage 1

Let us consider stage 1, substituting $x_n = e^{i\omega n}$ into equation A.2 gives:

$$H_1(\omega)e^{i\omega n} = a_2e^{i\omega n} - a_1H_1(\omega)e^{i\omega(n-1)} \quad (A.7)$$

Factoring out $e^{i\omega n}$ gives:

$$H_1(\omega) = a_2 - a_1H_1(\omega)e^{-i\omega}$$

$$\Rightarrow \quad H_1(\omega)(1 + a_1e^{-i\omega}) = a_2$$

$$\Rightarrow \quad H_1(\omega) = \frac{a_1}{(1 + a_1e^{-i\omega})}$$

The transfer function has complex parts due to the quantity $e^{-i\omega}$. When we plot the transfer function, we plot the magnitude of the transfer function, $|H(\omega)|$. 

The magnitude can be calculated using the relationship

\[ |H(\omega)|^2 = H(\omega)H(-\omega) \quad (A.8) \]

This is a real quantity with no complex part.

\[
|H_1(\omega)|^2 = H_1(\omega)H_1(-\omega) = \frac{a_2}{(1 + a_1 e^{-i\omega})} \cdot \frac{a_2}{(1 + a_1 e^{i\omega})} \\
= \frac{a_2^2}{1 + a_1 e^{-i\omega} + a_1 e^{i\omega} + a_1^2} \\
= \frac{a_2}{1 + a_1(e^{-i\omega} + e^{i\omega}) + a_1^2} \\
= \frac{a_2^2}{1 + a_1^2 + 2a_1 \cos \omega},
\]

since

\[ e^{-i\omega} + e^{i\omega} = \cos \omega - i \sin \omega + \cos \omega + i \sin \omega = 2 \cos \omega \]

The magnitude of the transfer function is therefore:

\[ |H_1(\omega)| = \sqrt{\frac{a_2^2}{1 + a_1^2 + 2a_1 \cos \omega}} \quad (A.9) \]
A.1.2 Stage 2

The technique used to determine the transfer function for stage 2 is the same as that used for stage 1. Setting $u_n = e^{i\omega n}$ and $y_n = H_2(\omega)e^{i\omega n}$ in equation A.1 we get:

$$H_2(e^{i\omega}) = a_5e^{i\omega} - a_3H_2(\omega)e^{-i\omega} - a_4H_2(\omega)e^{i\omega} - 2i\omega$$

$$\Rightarrow \quad H_2(\omega) = a_5 - a_3H_2(\omega)e^{-i\omega} - a_4H_2(\omega)e^{2i\omega}$$

$$\Rightarrow \quad H_2(\omega) + a_3H_2(\omega)e^{-i\omega} + a_4H_2(\omega)e^{-2i\omega} = a_5$$

$$\Rightarrow \quad H_2(\omega)(1 + a_2e^{-i\omega} + a_4e^{-2i\omega} = a_5$$

$$\Rightarrow \quad H_2(\omega) = \frac{a_5}{1 + a_3e^{-i\omega} + a_4e^{-2i\omega}}$$

Multiplying the transfer function by its complex conjugate and substituting using the Euler identities to determine the square of the magnitude yields:

$$|H_2(\omega)|^2 = \frac{a_5}{(1 + a_3e^{-i\omega} + a_4e^{-2i\omega})(1 + a_3e^{i\omega} + a_4e^{2i\omega})}$$

$$= \frac{a_5^2}{1 + a_3e^{i\omega} + a_4e^{2i\omega} + a_3e^{-i\omega} + a_4e^{-2i\omega} + a_3a_4e^{i\omega} + a_4a_3e^{-i\omega} + a_3^2 + a_4^2}$$

$$= \frac{a_5^2}{1 + a_3(e^{i\omega} + e^{-i\omega}) + a_4(e^{2i\omega} + e^{-2i\omega}) + a_3^2 + a_4^2 + a_3a_4(e^{i\omega} + e^{-i\omega})}$$

$$= \frac{a_5^2}{1 + 2a_3\cos\omega + 2a_4\cos2\omega + a_3^2 + a_4^2 + 2a_3a_4\cos\omega}$$

Therefore the magnitude of the transfer function is:

$$|H_2(\omega)| = \sqrt{\frac{a_5^2}{1 + 2a_3\cos\omega + 2a_4\cos2\omega + a_3^2 + a_4^2 + 2a_3a_4\cos\omega}} \quad (A.10)$$
A.1.3 Plotting the Transfer function

Combining the magnitudes of the transfer functions for stage 1 and stage 2 gives the magnitude of the transfer function for the low pass filter.

\[ |H(\omega)| = |H_1(\omega)| \cdot |H_2(\omega)| \]  \hspace{1cm} (A.11)

We now have all the information we need to plot the magnitude of the transfer function, \(|H(\omega)|\). Table A.1 gives the values of the coefficients for three filter configurations.

<table>
<thead>
<tr>
<th></th>
<th>250</th>
<th>600</th>
<th>900</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a_1)</td>
<td>-0.930126</td>
<td>-0.840427</td>
<td>-0.770460</td>
</tr>
<tr>
<td>(a_2)</td>
<td>+0.069874</td>
<td>+0.159573</td>
<td>+0.229540</td>
</tr>
<tr>
<td>(a_3)</td>
<td>-1.897266</td>
<td>-1.662751</td>
<td>-1.394894</td>
</tr>
<tr>
<td>(a_4)</td>
<td>+0.930126</td>
<td>+0.840427</td>
<td>+0.770460</td>
</tr>
<tr>
<td>(a_5)</td>
<td>+0.032860</td>
<td>+0.177676</td>
<td>+0.375566</td>
</tr>
</tbody>
</table>

Table A.1: Coefficients for 250, 600, and 900 Hz low pass filters

Using the coefficients for the 900 Hz low pass filter and plotting equation A.3 gives the transfer function shown below in Figure A.3.

Figure A.3: Transfer function, \(|H(\omega)|\), for the 900 Hz Low Pass Filter
You will notice that frequencies around half the filter cutoff (450 Hz in this case) are attenuated also. This is an undesirable, but tolerable, side effect. All digital filters differ from the ideal in some respect. Filters are usually categorised by their pass band and stop band ripple, and the slope of the transition between the pass and stop bands. Filter transfer functions are frequently plotted using a logarithmic scale which allows detailed examination of both the pass and stop band ripple. Figure A.4 below shows the transfer function for the 900 Hz low pass filter plotted using a decibel (dB) scale. The decibel scale is defined as 20 log 10(magnitude).

![Graph of 900 Hz Low Pass Filter](image)

Figure A.4: Transfer function, measured in dB, for the 900 Hz Low Pass Filter

The passband attenuation is approximately 2 dB at 450 Hz. The maximum stop band attenuation is approximately 35 dB at 4 kHz. The slope of the transition region attenuation is approximately 20 dB/octave. A distance of one octave represents the distance from \( f \) to \( 2f \). For example 1000 Hz to 2000 Hz is a distance of one octave. Figures A.5 and A.6 show the transfer functions for the 600 Hz and 250 Hz low pass filters. Notice the transfer characteristics are different for the three different filters. The passband ripple is similar but the transition region slopes and the maximum attenuation of the stop bands are quite different.
Figure A.5: Transfer function, measured in dB, for the 600 Hz Low Pass Filter

Figure A.6: Transfer function, measured in dB, for the 250 Hz Low Pass Filter
Appendix B

Digital pre-emphasis

Digital pre-emphasis by differencing the speech wave is used to emphasize the area of the second formant of speech. The differencing equation used is as follows:

\[ y_n = x_n - ax_{n-1} \]  \hspace{1cm} \text{(B.1)}

giving a 6dB/octave lift. This means that with each doubling of frequency (1 octave) the signal power increases by a factor of four; the amplitude by a factor of 16 (6 dB). Digital pre-emphasis is used to counteract a natural 6dB/octave decay in speech radiated from the lips [29]. The value of the constant \( a \) is not critical and typical values lie between 0.9 and 1. Following the reasoning from appendix A we substitute

\[ x_n = e^{i\omega n} \]  \hspace{1cm} \text{(B.2)}

and

\[ y_n = H(\omega)e^{i\omega n} \]  \hspace{1cm} \text{(B.3)}

into the differencing equation. This gives the following:

\[ H(\omega)e^{i\omega n} = e^{i\omega n} - ae^{i\omega(n-1)} = e^{i\omega n}(1 - ae^{-i\omega}) \]

So the transfer function for the pre-emphasis filter is:

\[ H(\omega) = 1 - ae^{-i\omega} \]  \hspace{1cm} \text{(B.4)}
Determining the magnitude of the transfer function:

\[
|H(\omega)|^2 = H(\omega)H(-\omega) \\
= (1 - ae^{-i\omega})(1 - ae^{i\omega}) \\
= 1 - a(e^{i\omega} + e^{-i\omega}) + a^2e^{-i\omega}e^{i\omega} \\
= 1 - a(\cos \omega + i\sin \omega + \cos \omega - i\sin \omega) + a^2 \\
= 1 - 2a \cos \omega + a^2,
\]

and therefore,

\[
|H(\omega)| = \sqrt{1 - 2a \cos \omega + a^2}
\]  \hspace{1cm} (B.5)

Substituting

\[
\omega = \frac{2\pi f}{f_s}
\]  \hspace{1cm} (B.6)

and displaying as a log magnitude plot, yields the graph in Figure B.1.

![Frequency Response for the Digital Pre-emphasis filter](image)

**Figure B.1:** Transfer function for the Digital Pre-emphasis filter
Appendix C

Lattice Filter Transfer function

This appendix shows the calculations to determine the transfer function for the 4 stage lattice filter used for the SIFT algorithm. A 12 stage lattice filter is used to synthesize a singing voice from 12 reflection coefficients. A similar analysis could be performed for this filter also. XMaple V was used to perform the calculations and plot the graphs shown at the end of this appendix. Both the maple commands and the maple output are shown. Maple commands are in courier type.

Represent each stage as a matrix

\[
S_1 := \begin{bmatrix}
1 & -\frac{k_4}{z} \\
-k_4 & \frac{1}{z}
\end{bmatrix}
\]

\[
S_2 := \begin{bmatrix}
1 & -\frac{k_3}{z} \\
-k_3 & \frac{1}{z}
\end{bmatrix}
\]

> S[1] := convert(array(0..1,0..1,[1,-k[4]*(1/z)],[-k[4],(1/z)]),matrix);

> S[2] := convert(array(0..1,0..1,[1,-k[3]*(1/z)],[-k[3],(1/z)]),matrix);
\[
S_3 := \begin{bmatrix}
1 & -\frac{k_2}{z} \\
-k_2 & \frac{1}{z}
\end{bmatrix}
\]

\[
S_4 := \begin{bmatrix}
1 & -\frac{k_1}{z} \\
-k_1 & \frac{1}{z}
\end{bmatrix}
\]

Multiply the stages together so we may extract the transfer function

\[
H := \text{linalg[multiply]}(S_1, S_2, S_3, S_4);
\]

\[
H := \left[(z^3 + k_1 k_3 z^2 + k_2 k_3 z^2 + z k_2 k_4 + k_1 k_2 z k_4 k_3 + k_1 k_3 z + k_1 k_4) / z^3, -\left(k_1 z^3 + k_1 z^2 k_4 k_3 + k_1 z^2 k_2 k_3 + k_1 z k_2 k_4 + k_2 z^2 + k_2 z k_4 k_3 + k_3 z + k_1 k_4\right) / z^3\right]
\]

\[
\left[-\left(k_1 z^3 + k_3 z^2 + k_2 z^2 k_4 k_3 + k_2 z + k_1 k_2 z k_4 k_3 + k_1 k_2 k_3 z + k_1 k_4 k_3 z + k_1 k_3 z + k_1 k_4 k_3 z + k_1 k_3 z + 1\right) / z^4\right]
\]

Recall the discussion of the lattice filter from Chapter 6. The lattice filter transfer function is given by the top row of the matrix product, \( H \). Here we extract the top row of the product and reassign it to \( H \).

\[
H := \text{linalg[row]}(H, 1);
\]

\[
H := \left[(z^3 + k_1 k_3 z^2 + k_2 k_3 z^2 + z k_2 k_4 + k_1 k_2 z k_4 k_3 + k_1 k_3 z + k_1 k_4) / z^3, -\left(k_1 z^3 + k_1 z^2 k_4 k_3 + k_1 z^2 k_2 k_3 + k_1 z k_2 k_4 + k_2 z^2 + k_2 z k_4 k_3 + k_3 z + k_1 k_4\right) / z^3\right]
\]

\[
\left[-\left(k_1 z^3 + k_3 z^2 + k_2 z^2 k_4 k_3 + k_2 z + k_1 k_2 z k_4 k_3 + k_1 k_2 k_3 z + k_1 k_4 k_3 z + k_1 k_3 z + k_1 k_4 k_3 z + k_1 k_3 z + 1\right) / z^4\right]
\]
Combine the two columns into a non-vector expression

\[ u := \text{array([1,0])}; \quad u := [1 0] \]

\[ v := \text{array([0,1])}; \quad v := [0 1] \]

\[ H := \text{linalg[dotprod]}(u,H) + \text{linalg[dotprod]}(v,H); \]

\[ H := 1 + \frac{k_1 k_3}{z} + \frac{k_2 k_3}{z^2} + \frac{k_2 k_4}{z} + \frac{k_1 k_2}{z} + \frac{k_1 k_2 k_4 k_3}{z^2} + \frac{k_1 k_3}{z^2} + \frac{k_1 k_4}{z} - \frac{k_1}{z} \\
- \frac{k_1 k_3 k_4}{z^2} \quad - \frac{k_1 k_2 k_3}{z^2} \quad - \frac{k_1 k_2 k_4}{z^3} \quad - \frac{k_2}{z^2} \quad - \frac{k_2 k_4 k_3}{z^3} \quad - \frac{k_3}{z^3} \quad - \frac{k_4}{z^4} \]

Collect like terms in \( z \) to obtain the transfer function for the four stage lattice filter

\[ H := \text{collect}(H,z); \]

\[ H := 1 + \frac{k_1 k_2 - k_1 + k_4 k_3 + k_2 k_3}{z} \\
+ \frac{-k_2 + k_1 k_2 k_4 k_3 + k_1 k_3 - k_1 k_4 k_3 - k_1 k_2 k_3 + k_2 k_4}{z^2} \\
+ \frac{-k_2 k_4 k_3 - k_3 + k_1 k_4 - k_1 k_2 k_4}{z^3} - \frac{k_4}{z^4} \]

Alternatively, \( H \) can be expressed as:

\[ H := \text{subsop}(2 = a[1]/z, 3 = a[2]/z^2, 4 = a[3]/z^3, 5 = a[4]/z^4, H); \]

\[ H := 1 + \frac{a_1}{z} + \frac{a_2}{z^2} + \frac{a_3}{z^3} + \frac{a_4}{z^4} \]
where

\[
\]

\[
a_1 := k_1 k_2 - k_1 + k_4 k_3 + k_2 k_3
\]

\[
\]

\[
a_2 := -k_2 + k_1 k_2 k_4 k_3 + k_1 k_3 - 2 k_1 k_4 k_3 - k_1 k_2 k_3 + k_2 k_4
\]

\[
\]

\[
a_3 := -k_2 k_4 k_3 - k_3 + k_1 k_4 - k_1 k_2 k_4
\]

\[
\]

\[
a_4 := -k_4
\]

This is equivalent to the direct evaluation by linear prediction coefficients. So, for a four stage filter, the equations above show the relationship between the linear prediction coefficients and the reflection coefficients. Since we are working mainly with reflection coefficients we will continue the analysis using them rather than the linear prediction coefficients, although the mathematics is quite verbose. (The interested reader could comment out the equations for the linear prediction coefficients above to continue the analysis in terms of the prediction coefficients).

Expressing the Z-transform in terms of its sines and cosines complex expression

\[
> z := \cos(w) + I*\sin(w) ; \\
\]

\[
z := \cos( w ) + I \sin( w )
\]

We make the transfer function, \( H \), into a maple function in \( w \), so we can calculate its magnitude function
\[ H := \text{student}[\text{makeproc}](H, w); \]

\[
H := w \rightarrow 1 + \frac{k_1 k_2 - k_1 + k_3 k_4 + k_2 k_3}{\cos(w) + i \sin(w)} + \frac{-k_2 + k_1 k_2 k_4 k_5 + k_4 k_3 - 2 k_1 k_4 k_3 - k_1 k_2 k_3 + k_2 k_4}{(\cos(w) + i \sin(w))^2} + \frac{-k_2 k_4 k_3 - k_3 + k_4 k_3 - k_2 k_4}{(\cos(w) + i \sin(w))^3} - \frac{k_4}{(\cos(w) + i \sin(w))^4}
\]

The magnitude of the transfer function is therefore:

\[
M := \sqrt{\text{collect}(\text{simplify}(H(w) \cdot H(-w), \cos(w)))};
\]

\[
M := \left(2k_1 k_4 k_5 + 1 + 2k_1^2 k_2^2 k_4 + k_1^2 k_2^2 k_4^2 - 2k_4 + k_4^2 k_3^2 + 2k_2 \right.
+ 4k_4 k_3^2 k_2 - 4k_2 k_4 - 4k_1 k_3 + k_2^2 k_4^2 + 2k_1^2 k_4 + k_2^2 k_3^2 + k_1^2 k_4^2
- 2k_2^2 k_4 + \left(-4k_2^2 k_3^2 k_4 + 8k_2 k_4 - 4k_1^2 k_2^2 k_4 - 8k_2^2 k_3^2 k_1 k_3
- 4k_4^2 k_3^2 k_2 + 8k_1 k_2 k_4 - 8k_1 k_2 k_3 + 16k_1 k_2 k_4 k_3 - 4k_2 k_3^2
- 4k_2 + 16k_4 - 4k_4 k_3^2 + 8k_1 k_3 + 12k_1 k_2^2 k_3 - 8k_2^2 k_3 k_1 k_2
- 4k_2 k_4^2 - 4k_1^2 k_4 - 12k_1 k_4 k_3 \right) \cos(w)^2 - 4k_1^2 k_2 k_4 + \left(6k_1 k_2^2 k_3 k_2 - 2k_1^2 k_3 - 2k_1 k_2^2 k_3 k_2^2 k_3 - 2k_1 k_4^2 + 4k_1 k_2^2 k_3^2 k_4
+ 4k_1 k_2^2 k_4 k_3 - 2k_2^2 k_4 k_3^2 k_1 - 10k_1^2 k_2 k_4 k_3 - 10k_1 k_3^2 k_2 k_4
+ 6k_2 k_4^2 k_3^2 k_1 + 6k_3 - 2k_1 + 4k_4 k_3 - 12k_1 k_4 + 4k_2 k_3 + 4k_1 k_2
- 2k_1 k_3^2 - 2k_2^2 k_1 + 6k_4 k_3 + 8k_1 k_2 k_4 + 8k_2 k_4 k_3 - 2k_2^2 k_3
+ 4k_4 k_3 k_2^2 - 2k_2^2 k_3 k_1 + 4k_3^2 k_2 - 4k_1 k_2^2 k_3 - 2k_2^2 k_4 k_1\right)
- 2k_2^2 k_4^2 k_3 + 6k_1 k_4 k_3 + 6k_3 k_2 k_3 - 2k_1 k_2^2 k_3 + 4k_3^2 k_1 k_2
+ 4k_2^2 k_1 k_4 + 4k_1^2 k_2 k_3 + 4k_2^2 k_4 k_3 - 4k_4^2 k_2 k_3 \right) \cos(w)
- 2k_2^2 k_4^2 k_2
+ \left(-16k_1 k_2 k_4 + 16k_1 k_4 - 8k_4^2 k_3 - 8k_3 - 16k_2 k_4 k_3 \right) \cos(w)^3
+ 2k_1 k_2 k_4 k_3 - 16k_4 \cos(w)^4 - 2k_1^2 k_2 - 2k_4^2 k_3 k_1 k_2
+ k_1^2 k_2^2 k_4^2 k_2 k_3 + 2k_2^2 k_3^2 k_4 + k_1^2 k_2 k_3 + k_2^2 - 4k_1^2 k_2 k_4^2 k_3^2
+ 2k_2 k_4^2 + 2k_3^2 k_2 + k_1^2 + 2k_4^2 k_3^2 k_2 + 2k_3 k_2^2 + k_1^2 + 2k_2
+ k_1^2 k_2^2 + k_3^2 k_2^2 + 2k_2^2 k_4^2 k_1 k_3 + k_1^2 k_2 k_3^2 + 6k_1^2 k_3^2 k_2 k_4
- 6k_1 k_4^2 k_3 - 4k_1^2 k_2 k_3 - 2k_1^2 k_2 k_3 + 4k_1^2 k_4^2 k_3^2
- 2k_1^2 k_2^2 k_3^2 k_4 + 4k_2^2 k_1 k_3 \right)^{1/2}
\]
Accounting for the sampling frequency (8000 Hz) and converting to a frequency scale, we set \( w \) equal to:

\[
> w := 2\pi f / 8000;
\]

\[
w := \frac{1}{4000} \pi f
\]

Now we convert the magnitude function to a maple function in \( f \):

\[
> M := \text{student}[\text{makeproc}](M,f);
\]

\[
M := f \rightarrow \left( 2 k_1 k_4 k_3 + 1 + 2 k_1^2 k_2^2 k_4 + k_1^2 k_2^2 k_4^2 - 2 k_4 + k_1^2 k_3^2 + 2 k_2 \\
+ 4 k_1 k_3^2 k_2 - 4 k_2 k_4 - 4 k_1 k_3 + k_2^2 k_4^2 + 2 k_1^2 k_4 + k_2^2 k_3^2 + k_1^2 k_4^2 \\
- 2 k_2^2 k_4 - 4 k_1^2 k_2 k_4^2 - 2 k_1 k_2 k_4 k_3 - 2 k_1^2 k_2 \\
- 2 k_1^2 k_3 k_1 k_2 + k_1^2 k_2^2 k_4^2 + 2 k_2^2 k_3^2 k_4 + k_1^2 k_3^2 k_2^2 + k_2^2 \\
- 4 k_1^2 k_2 k_4^2 k_3^2 + 2 k_2 k_4^2 + 2 k_4 k_3^2 + k_3^2 + 2 k_4^2 k_3^2 k_2 + 2 k_2 k_3^2 \\
+ k_1^2 + k_2^2 - 16 k_4 \cos \left( \frac{1}{4000} \pi f \right) + k_1^2 k_2^2 + k_2^2 k_4^2 k_3^2 \\
+ 4 k_2^2 k_4^2 k_1 k_3 + k_1^2 k_2^2 k_3^2 + 6 k_1^2 k_2^2 k_2 k_4 - 6 k_1 k_4 k_3^2 + ( \\
- 4 k_2^2 k_4^2 k_1 k_3 + 8 k_2 k_4 - 4 k_1^2 k_2 k_4^2 - 8 k_2 k_4 k_1 k_3 - 4 k_4^2 k_3^2 k_2 \\
+ 8 k_1 k_3 k_4 - 8 k_1 k_2 k_3 + 16 k_1 k_2 k_4 k_3 - 4 k_2 k_3^2 - 4 k_2^2 + 16 k_4 \\
- 4 k_1 k_3^2 + 8 k_1 k_3 + 12 k_1 k_4^2 k_3 - 8 k_4^2 k_3 k_1 k_2 - 4 k_4 k_2 k_4 - 4 k_1^2 k_4 \\
- 12 k_1 k_4 k_3) \cos \left( \frac{1}{4000} \pi f \right)^2 - 4 k_1^2 k_3^2 k_4 + (6 k_1^2 k_1 k_2 k_3 \\
- 2 k_1 k_3^2 - 2 k_1^2 k_2 k_4^2 k_3^2 - 2 k_1 k_4^2 + 4 k_1 k_2^2 k_3^2 k_4 + 4 k_1^2 k_2 k_3^2 k_4 \\
- 2 k_2 k_4^2 k_3^2 k_1 - 10 k_1^2 k_2 k_4 k_3 - 10 k_1 k_3^2 k_2 k_4 + 6 k_2 k_4^2 k_3^2 k_1 \\
+ 6 k_3 - 2 k_1 + 4 k_4 k_3 - 12 k_1 k_4 + 4 k_2 k_3 + 4 k_1 k_2 - 2 k_1 k_3^2 \\
- 2 k_2^2 k_1^2 + 6 k_4^2 k_3 + 8 k_1 k_2 k_4 + 8 k_2 k_4 k_3 - 2 k_2^2 k_3 + 4 k_1 k_2 k_4^2 \\
- 2 k_2 k_4^2 k_3^2 k_1 + 4 k_2 k_4^2 k_3 - 4 k_1^2 k_4^2 k_3 - 2 k_2^2 k_4^2 k_1 - 2 k_2^2 k_4^2 k_3 \\
+ 6 k_1 k_4 k_3 + 6 k_4 k_3^2 k_1 - 2 k_1^2 k_2^2 k_3 + 4 k_3^2 k_1 k_2 + 4 k_2^2 k_1 k_4 \\
+ 4 k_1^2 k_2 k_3 + 4 k_2^2 k_4 k_3 - 4 k_4^2 k_3^2 k_1) \cos \left( \frac{1}{4000} \pi f \right) - 2 k_1^2 k_3^2 k_2 \\
+ 4 k_1^2 k_2^2 k_3^2 +
\]
\[
\left( -16k_1k_2k_4 + 16k_1k_4 - 8k_4^2k_3 - 8k_3 - 16k_2k_4k_3 \right) \\
\cos \left( \frac{1}{4000} \pi f \right)^3 - 2k_1^2k_2^2k_3^2k_4 + 4k_2^2k_1k_3 \right)^{1/2}
\]

Setting typical values for a, b, c, and d we can obtain a magnitude plot of the transfer function. These values were obtained directly from the singing synthesis system for a small section of a sung 'la'.

\>
k[1] := -0.699759; \\
k_1 := -0.699759
\>
k[2] := 0.519896; \\
k_2 := 0.519896
\>
k[3] := -0.122464; \\
k_3 := -0.122464
\>
k[4] := 0.549578; \\
k_4 := 0.549578
\>
plot(M(f), f=0..4000);

Alternatively we may plot the transfer function on a logarithmic magnitude scale. Here the decibel (dB) scale is used.

\>
L := 20*\log10(M(f)); \\
L := 20\log10 \left( \left( 1.185423136 \cos \left( \frac{1}{4000} \pi f \right) \right) \\
+ 8.297591587 \cos \left( \frac{1}{4000} \pi f \right)^2 - 8.793248 \cos \left( \frac{1}{4000} \pi f \right)^4 \right)
\[-1.118680486 \cos\left(\frac{1}{4000} \pi f\right)^3 + .5624966733^{1/2}\]

> \textbf{L} := \textbf{student}[\textbf{makeproc}](\textbf{L}, \textbf{f});

\[L := f \rightarrow 20 \log_{10}\left(\left(1.185423136 \cos\left(\frac{1}{4000} \pi f\right)\right.\right.\]
\[+ \; 8.297591587 \cos\left(\frac{1}{4000} \pi f\right)^2 - \; 8.793248 \cos\left(\frac{1}{4000} \pi f\right)^4\]
\[\left. - \; 1.118680486 \cos\left(\frac{1}{4000} \pi f\right)^3 + .5624966733^{1/2}\right)\]

> \textbf{plot}([\textbf{L}(\textbf{f}), \textbf{f}=0..4000]);
The graph of the transfer function (plotted by XMaple) is shown in figure C.1 below:

Figure C.1: A transfer function for a four stage lattice filter

The log magnitude plot (measured in decibels) is also shown below:

Figure C.2: A log magnitude transfer function for a four stage lattice filter
Appendix D

The Sound Spectrograph

The sound spectrograph is an important tool for analysing the frequency spectrum of sounds. Put simply, a sound spectrograph is a frequency vs. time graph, showing multiple frequencies per unit time. The intensities of the frequencies at each unit time are indicated by a colour or gray scale. In this implementation, a colour scale is used. The scale runs from blue through green through yellow, orange, and finally to red. 128 different intensities are supported.

D.1 The Fourier Transform

The Fourier transform provides the mathematical basis for the sound spectrograph. The Fourier transform maps a continuous time domain function into a continuous frequency domain function. The Inverse Fourier Transform maps a continuous frequency domain signal into a continuous time domain signal. The Fourier Transform (FT) is given by:

\[ X(f) = \int_{-\infty}^{\infty} x(t) e^{-i2\pi ft} dt \]  

(D.1)

and the Inverse Fourier Transform (IFT) is given by

\[ x(t) = \int_{-\infty}^{\infty} X(f) e^{i2\pi ft} df \]  

(D.2)
The relationship between the Fourier Transform and its inverse is illustrated in Figure D.1.

![Fourier Transform Diagram](image)

**Figure D.1: The operation of the Fourier Transform and its Inverse**

The function \( e^{-i\theta} = \cos \theta - i \sin \theta \), where \( i^2 = -1 \), represents the complex number \((\cos \theta, -\sin \theta)\). Therefore the function represents points on the circle in the complex plane.

A discrete version of the Fourier Transform exists, called the Discrete Fourier Transform (DFT). It also has an inverse called the Inverse Discrete Fourier Transform (IDFT). These are shown below.

\[
X(f) = \sum_{n=0}^{N-1} x(n)e^{-i2\pi fn/N} \quad \text{(DFT)}
\]

\[
x(n) = \frac{1}{N} \sum_{f=0}^{N-1} X(f)e^{+i2\pi fn/N} \quad \text{(IDFT)}
\]

These are, in fact, simply rectangular rule approximations to the integrals [3]. The DFT takes a discrete time varying signal, such as those encountered frequently throughout this thesis, and converts it into its discrete frequency representation. Often the time varying input signal is represented as time varying complex data with no imaginary component. This allows the complex valued output of the DFT to replace the real input values. This is known as an in-place Fourier Transform. It almost seems that the Fourier Transform gives us information for nothing. Providing \( N \) real values as input generates \( N \) complex values as output. It can be shown however that only the first \( N/2 \) values of the output are relevant. The last half of the complex output values are simply the complex conjugate of the complex input values but in reverse order. That is, the output is reflected about the midpoint, \( N/2 \).

The DFT returns \( N \) frequency components, equally spaced, between 0 Hz and the
sampling frequency, which in our case is 8000 Hz. Only the first half of the spectrum is usable because of the reflection about the midpoint. For example, for a 256 point \((N = 256)\) Fourier Transform, the resulting spectrum gives 256 discrete frequency components evenly spaced between 0 and 8000 Hz. The frequency resolution is \(8000/256 = 31.25\) Hz. The frequency resolution for a DFT is generally quite low making the Fourier Transform a poor technique for fundamental frequency estimation, although it does give a good illustration of the formant behavior of speech. Each point, or frequency “bin” as they are sometimes called, is a complex number with real and imaginary parts.

### D.1.1 Restrictions on the Fourier Transform

Several restrictions must be placed on the use of the Fourier Transform for estimating the frequency domain of a time varying waveform. The Fourier Transform assumes that the input signal is periodic. Unfortunately typical speech signals are not periodic but can be described as quasi-periodic. That is, the pitch period will not differ significantly from one pitch period to the next. The pitch is a slowly varying parameter of the speech. Nevertheless, it is by no means constant. Obviously the periodicity assumption will be violated for large \(N\). The vocal articulators cannot change significantly in less than 10-25ms, therefore it is usually assumed that the speech waveform is relatively constant for durations of less than 10-25ms. This gives an upper bound on the number of sample points to provide as input to the DFT function of 200 (at an 8000 Hz sampling rate). Recall the discussion from Chapter 4 regarding windowing. Spectral analysis of a finite rectangular windowed waveform results in a spectrum with unrealistic high frequency components caused by the sharp transitions in the input waveform at the edges of the rectangular window. The transitional effects can be removed by using a window such as a Hamming window, which attenuates the signal at the edges of the window. The attenuation can be compensated for by overlapping each window with its neighbour by 50%, similar to the technique used for PSOLA synthesis which was discussed in Chapter 5. The Fourier Transform is \(O(N^2)\), so a
256 point DFT requires 65536 complex multiplications and additions. While this is not a great problem now, as computers are becoming more powerful, it was a serious limitation for scientists and engineers in the past. An algorithm known as the Fast Fourier Transform (FFT) based on a divide and conquer technique is \( O(N \log_2 N) \). Therefore a 256 point FFT can be computed using only 2048 multiplications and additions. A brief description of the FFT algorithm can be found in [29].

D.1.2 Graphing the frequency Spectrum

Log-magnitude plots are typically used to display the output of a Fourier Transform applied to speech data. The normalized magnitude of each of frequency “bins” is plotted on a logarithmic scale on the y axis against the frequency bin number on the x axis. The frequency bin number corresponds to a range of frequencies, that is, bin 0 corresponds to frequencies 0-31.25 Hz, bin 2 corresponds to frequencies 31.25-62.5 Hz, and so on. So the x-axis labels can be displayed as frequencies instead of bin numbers. In addition, the plot could be a line or a bar plot, but this is a matter of preference. The formula for calculating the logarithm of the normalized amplitude is given by:

\[
    r(f) = 10 \log_{10} \left( \frac{x(f)^2 + y(f)^2}{N^2} \right)
\]

where \( x(f) \) and \( y(f) \) are the real and imaginary parts of the complex number associated with the frequency bin \( f \), and \( N \) is the number of points (or number of frequency bins) in the Fourier Transform. A log magnitude spectral plot of a voiced section of a speech signal is shown in Figure D.2.

D.2 From the Fourier Transform to the Spectrograph

A spectrograph can be thought of as a way to view multiple frequency spectrums over time. Instead of viewing frequency spectrums for a 32 ms section of speech
one at a time, the spectrograph provides a way to view them all at the same time. Consider producing N frequency spectrums and consolidating them to obtain a three-dimensional plot of magnitude vs. frequency vs. plot number. Alternatively, consider printing each plot on a piece of cardboard and cutting off the cardboard above the plotted line. Imagine stacking each plot side by side, like a deck of cards, and looking down onto the landscape of peaks and valleys. What you are looking at is essentially a spectrograph. On a computer display we don’t have the freedom of three dimensions and simulated 3D plots are sometimes undesirable. The spectrograph represents the height and depth of peaks and valleys with colour or intensity, much like a topographical map showing height above sea level. Very large heights are represented by “hot” colours (reds, oranges, and yellows) and low values are represented by “cooler” colours (greens, blues, and black). For a non colour display, the heights are represented as intensities on a gray scale, with darker grays representing high peaks and light grays representing low peaks.

Recall that adjacent Fourier Transforms are typically overlapped by 50% to minimise the effect of the attenuation caused by the Hamming window. A one second section of speech would need approximately 31 non-overlapped 256 point spectral plots for complete coverage of the duration. A 50% overlap necessitates 62 spectral plots. No special averaging between adjacent spectra is performed, the time scale is
simply stretched by a factor of two.

On the screen each spectral plot is represented by a single vertical line 128 pixels high and 1 pixel wide. Each pixel on the line corresponds to a single frequency bin from that spectral plot. For wider bandwidth spectrographs, a single frequency bin will occupy more than a single pixel giving the spectrograph a somewhat blocky appearance. This is not inappropriate however since the low pixel definition on the screen corresponds to the low frequency definition of the spectrum.

The intensities can be adjusted by a threshold factor. Consider again a topographical map and imagine adjusting the height of the sea level to cover or uncover some peaks. Obviously raising the sea level would obscure some low hills, whereas lowering it below normal might uncover some sub-aquatic hills and mountains. This same principle is followed with the user-adjustable threshold factor for the spectrograph. Look again at Figure D.2, and consider the threshold line lying at \( y = 0 \) (where \( y \) is the vertical axis). Every value below the threshold is not assigned a colour—these are represented as black dots—only values above the threshold are assigned a colour within the blue-red range. The threshold may be adjusted between -64 and 64 using one of the spectrograph controls. Increasing the threshold value actually decreases the threshold below the zero line showing more detail. Decreasing the threshold has the opposite effect. In this way the threshold level is more like a sensitivity setting, and we refer to increasing and decreasing the sensitivity without reference to the threshold level.

While increasing the sensitivity allows us to see more of the hidden detail, it also increases the noise level. Small fluctuations in the spectral envelope show up on the spectrograph as random noise. This is particularly obvious on the spectrograph with bandwidth equal to 250 Hz and threshold equal to 35, see Figure D.4.
D.2.1 Spectrograph examples:

Shown below are some example spectrographs of a sung ‘la’ sound. The spectrographs are shown with bandwidths of 31.25 Hz, 62.5 Hz, 125 Hz, and 250 Hz. They are shown with thresholds of 15 (Figure D.3) and 35 (Figure D.4).

Figure D.3: Spectrographs of a ‘la’ sound with threshold = 15

Figure D.4: Spectrographs of a ‘la’ sound with threshold = 35
Appendix E

Capture Scripts

The format for the capture scripts used to automatically extract phonetic elements and an example script are shown below. Semicolons before the first letter of the phonetic element name serve as a comment marker and cause the capture algorithm to ignore that line of the capture script. Figure E.1 shows the capture script format. Figure E.2 shows an example capture script.

```
<filename>
phonetic element name start offset end offset
phonetic element name start offset end offset
...
phonetic element name start offset end offset
...
filename
phonetic element name start offset end offset
...
phonetic element name start offset end offset
...
```

Figure E.1: Format for an automatic phonetic element capture script
Eleanor2.au

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>_E</td>
<td>1258</td>
</tr>
<tr>
<td>EL</td>
<td>2652</td>
</tr>
<tr>
<td>LN</td>
<td>4216</td>
</tr>
<tr>
<td>NO</td>
<td>5933</td>
</tr>
<tr>
<td>OR</td>
<td>7480</td>
</tr>
<tr>
<td>RI</td>
<td>8687</td>
</tr>
<tr>
<td>IG</td>
<td>9486</td>
</tr>
<tr>
<td>BE</td>
<td>12818</td>
</tr>
<tr>
<td>EE</td>
<td>13838</td>
</tr>
<tr>
<td>EEₕ</td>
<td>16976</td>
</tr>
<tr>
<td>PI</td>
<td>19023</td>
</tr>
<tr>
<td>IX</td>
<td>19669</td>
</tr>
<tr>
<td>UP</td>
<td>22100</td>
</tr>
<tr>
<td>;THUH</td>
<td>22695</td>
</tr>
<tr>
<td>RA</td>
<td>24846</td>
</tr>
<tr>
<td>Al</td>
<td>26486</td>
</tr>
<tr>
<td>AIS</td>
<td>27642</td>
</tr>
<tr>
<td>IN</td>
<td>30396</td>
</tr>
<tr>
<td>CH</td>
<td>33711</td>
</tr>
<tr>
<td>EU</td>
<td>34578</td>
</tr>
<tr>
<td>ECH</td>
<td>36618</td>
</tr>
<tr>
<td>WE</td>
<td>38448</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>ED</td>
<td>44946</td>
</tr>
<tr>
<td>IV</td>
<td>68497</td>
</tr>
<tr>
<td>S</td>
<td>69850</td>
</tr>
<tr>
<td>;UH</td>
<td>72435</td>
</tr>
<tr>
<td>;UH₀</td>
<td>73293</td>
</tr>
<tr>
<td>DREE</td>
<td>73740</td>
</tr>
<tr>
<td>EEM₀</td>
<td>80730</td>
</tr>
</tbody>
</table>

Eleanor2a.2.au

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>THUH</td>
<td>5340</td>
</tr>
<tr>
<td>_UH</td>
<td>14640</td>
</tr>
<tr>
<td>UH</td>
<td>16140</td>
</tr>
<tr>
<td>UH₀</td>
<td>19500</td>
</tr>
<tr>
<td>UHP</td>
<td>24780</td>
</tr>
</tbody>
</table>

Figure E.2: Example of a capture script
# Appendix F

## Phonetic Symbols - The Arpabet

<table>
<thead>
<tr>
<th>Phonetic Symbol</th>
<th>Example</th>
<th>Phonetic Symbol</th>
<th>Example</th>
<th>Phonetic Symbol</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>IY</td>
<td>heed</td>
<td>IX</td>
<td>roses</td>
<td>M</td>
<td>mom</td>
</tr>
<tr>
<td>IH</td>
<td>hid</td>
<td>P</td>
<td>pop</td>
<td>N</td>
<td>noon</td>
</tr>
<tr>
<td>EY</td>
<td>hayed</td>
<td>B</td>
<td>bob</td>
<td>NX</td>
<td>ringing</td>
</tr>
<tr>
<td>EH</td>
<td>head</td>
<td>T</td>
<td>tug</td>
<td>L</td>
<td>lulu</td>
</tr>
<tr>
<td>AE</td>
<td>had</td>
<td>D</td>
<td>dug</td>
<td>EL</td>
<td>battle (1)</td>
</tr>
<tr>
<td>AA</td>
<td>hod</td>
<td>K</td>
<td>kick</td>
<td>EM</td>
<td>bottom (1)</td>
</tr>
<tr>
<td>AO</td>
<td>hawed</td>
<td>G</td>
<td>gig</td>
<td>EN</td>
<td>button (1)</td>
</tr>
<tr>
<td>OW</td>
<td>hoed</td>
<td>F</td>
<td>fife</td>
<td>DX</td>
<td>batter (2)</td>
</tr>
<tr>
<td>UH</td>
<td>hood</td>
<td>V</td>
<td>verve</td>
<td>Q</td>
<td>(3)</td>
</tr>
<tr>
<td>UW</td>
<td>who’d</td>
<td>TH</td>
<td>thick</td>
<td>W</td>
<td>wow</td>
</tr>
<tr>
<td>ER</td>
<td>heard</td>
<td>DH</td>
<td>those</td>
<td>Y</td>
<td>yoyo</td>
</tr>
<tr>
<td>AX</td>
<td>ahead</td>
<td>S</td>
<td>cease</td>
<td>R</td>
<td>roar</td>
</tr>
<tr>
<td>AH</td>
<td>bud</td>
<td>Z</td>
<td>pizzaz</td>
<td>CH</td>
<td>church</td>
</tr>
<tr>
<td>AY</td>
<td>hide</td>
<td>SH</td>
<td>mesh</td>
<td>JH</td>
<td>judge</td>
</tr>
<tr>
<td>AW</td>
<td>how’d</td>
<td>ZH</td>
<td>measure</td>
<td>WH</td>
<td>where</td>
</tr>
<tr>
<td>OY</td>
<td>boy</td>
<td>HH</td>
<td>heat</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table F.1: Phonetic symbols developed by US Dept. of Defence

Table F shows the phonetic symbols developed under sponsorship of the US Department of Defence Advanced Research Projects Agency. The table was taken from [28]. The annotations 1-3 are (1) Vocalic l,m,n. (2) Flapped t. (3) Glottal Stop.

113
Bibliography


