COPYRIGHT AND CITATION CONSIDERATIONS FOR THIS THESIS/ DISSERTATION

- Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

- NonCommercial — You may not use the material for commercial purposes.

- ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.

How to cite this thesis
Voice Classification Using A Unique Key Signature

by

Johannes Hendrik Nysschen

Dissertation

submitted for the fulfillment of the requirements of the degree

Master in Commerce

in

Informatics

in the

Faculty of Economic and Business Sciences

at the

Rand Afrikaans University

Supervisor : Prof. E.M Ehlers

September 1992
I would like to express my sincere gratitude to the following persons for their incalculable assistance in the creation of this dissertation. These persons are: (in no specific order)

Gerry Cassidy (First National Bank), for the inspiration he gave me to follow this subject and for his assistance in obtaining the necessary hardware.

Gary Rolfe (First National Bank), for explaining the hardware and for his assistance in creating the software.

Lance Ayrest (First National Bank), for finding the software when I needed it the most.

Prof. Elize Ehlers (Randse Afrikaanse Universiteit), vir haar geduld en leiding deur twee jaar.

Nicholas Tombs, for his patience and for teaching me the C programming language.

Anchen Witthuhn, for understanding the pressures on my time schedule and for her ideas on improving this dissertation.
Abstract

The dissertation to follow will examine the knowledge domain of voice and speech recognition. We will further explore the field of voice classification systems, that is the study field that tries to classify the human voice into a fixed number of groups. We intend to define a model that would implement such a system. We also further intend to prove this model correct.

We will commence this dissertation with section one which is an introduction into the study field of voice and speech recognition.

Section two will introduce the reader to voice classification systems. We will also introduce the reader to our voice classification model called the Template Order Model. The remainder of section two will be devoted to a discussion of this model.

Section three intends to prove the Template Order Model. In this section we will explain the practical implementation of our model as well as a possible performance enhancer called the TOM Kernel.

We conclude this dissertation with an excerpt of the source code that is used to implement the model practically.
Opsomming

Die doelwit van hierdie verhandeling is om die kennisveld van rekenaar spraakherkenning te ondersoek met spesifieke verwysing na spraak-klassifiserings stelsels d.w.s 'n rekenaarstelsel wat menslike spraak klassifiseer in 'n beperkte aantal groepe. Ons beoog verder om ons eie spraak-klassifiserings model aan die leser bekend te stel en om hierdie model as korrek te bewys.

In die eerste afdeling van hierdie verhandeling stel ons die leser bekend aan die onderwerp van spraakherkenning. Hierdie afdeling moet deur die leser as 'n inleiding tot die kennisveld beskou word.

In die tweede afdeling stel ons die leser bekend aan ons spraak-klassifiserings model. Hierdie model staan bekend as die "Template Order Model" of TOM. Hierdie model word in afdeling twee bespreek.

In afdeling drie sal ons die leser bekend stel aan 'n prestasie verhoger wat bekend staan as die TOM "kernel". Ons sal verder die praktiese implementasie van ons model bespreek in hierdie afdeling.

Ons sluit hierdie verhandeling af met 'n uitreksel van die bron kode wat gebruik is om die model prakties te implementeer.
### SECTON I

**CHAPTER 1 : Voice And Speech Recognition**

1. The Future and Voice Recognition  
   1-1

2 Voice recognition  
   1-3

1.3 Speech Recognition  
   1-5

2. About this dissertation  
   1-6

**CHAPTER 2 : Articulary Phonetics And Phonemics**

1 Introduction into the field of phonetics  
   2-1

2 Articulary Phonetics  
   2-1

   2.1 Phonetic alphabets  
      2-1

   2.1.1. Categories  
      2-4

   2.1.2 Contoids and Consonants  
      2-5

   2.1.3 Point of articulation  
      2-5

   2.1.4 Manner or articulation  
      2-7

   2.1.5 Voicing  
      2-7

   2.1.6 Vocoids and Vowels  
      2-8

   2.1.7 Nasalization  
      2-8

   2.1.8 Diphthongs  
      2-9

   2.1.9 Coarticulation  
      2-9
2.2 Phonemics 2-10
  
2.2.1 Phonemes 2-10

2.2.2. Allophones 2-10

3 Acoustic Phonetics 2-11

  3.1 The sound spectrograph 2-12

  3.2 Acoustics of the vocal tract 2-14

  3.3 Analysis of the acoustics of vocoids 2-14

  3.4 Properties of vowel waveforms 2-15

  3.5 Acoustical characteristics of stops and fricatives 2-15

    3.5.1 Noise 2-16

    3.5.2 Formant transitions 2-16

4. Conclusion 2-17

CHAPTER 3 : The Digitized Human Voice Print

1. Digitization - An Introduction 3-1

2. Sampling 3-1

3 Quantization 3-4

  3.1 Pulse-Code Modulation (PCM) 3-4

  3.2 Delta Modulation 3-5

4 Coding 3-5

  4.1 Irreversible coding methods 3-8
5. Conclusion

CHAPTER 4 : Linear Predictive Coding (LPC)

1. Introduction

2. The basic synthesis model

  2.1 The All-pole model

  2.2 The source model

3 Direct-form analysis

  3.1 Data windowing

  3.2 Residual windowing

  3.3 Computation of predictor coefficients

4 Lattice analysis and synthesis

5 Linear prediction in speech recognition

6. Conclusions

CHAPTER 5 : Speech Synthesis

1. The principles of speech synthesis

2. Synthesis based on waveform coding

3. Synthesis based on the Analysis-synthesis method

4. Synthesis based on the speech production mechanism

5. Synthesis by Rule

  5.1 Conversion of phonemes to speech
5.2 Conversion of unrestricted text to phonemes (Grapheme-to-phoneme translation) 5-6

6. Conclusion 5-7

CHAPTER 6: Voice and Speech Recognition - Revisited

1. Background 6-1
2. Feature extraction 6-2
3. Template matching 6-4
4. Continuous-speech recognition 6-5
   - Segmentation 6-5
   - Phonetic variability 6-5
6. Conclusion 6-7

SECTION II

CHAPTER 7: Voice Classification Systems

1. What is a voice classification system 7-1
   - Search Time 7-1
   - Hardware requirements 7-2
   - Accuracy 7-2
2. Model definition 7-5
3. The origins of TOM 7-9
4. Conclusion 7-10
CHAPTER 8: Template Creation and Sound Peak Filters

1. Introduction 8-1
2. Phase 1: Template Creation 8-2
3. TOM Phase II - Peak Filters 8-4
   3.1 Peak filter design 8-6
   3.1.1 Filter One 8-6
   3.1.2 Filter Two 8-8
   3.1.3 Filter Three 8-8
   3.1.4 Filter Four 8-10
   3.1.5 Filter Five 8-10
   3.1.6 Filter 6 8-13
   3.1.7 Filter 7 8-15
3.2 Filter banks 8-17
4. Conclusions 8-17

CHAPTER 9: Start Detection and Garbage Removal

1. Start detection - The basics 9-1
2. TOM Phase Three: Start Point Detection 9-4
3. Phase Four: Garbage Removal 9-6
4. Conclusions 9-9
CHAPTER 10: Zero-crossings

1. The TOM Zero-crossing method. 10-1

2. Measuring zero-crossings 10-3

2.1 The Zero-crossings Detector (ZCD) 10-4

2.2 The Sample Distance Measurer (SDM) 10-4

3. The Zero-crossings Formatter (ZCF) 10-7

3.1 The Binary Coding Method 10-9

3.2 A practical example 10-10

4. TOM - Practical Implementation 10-15

5. Conclusions 10-16

CHAPTER 11: The TOM Kernel

1. Introduction 11-1

2. The Kernel 11-1

2.1 All three primary groups agree 11-3

2.2 Two out of Three primary descriptors agree 11-5

2.3 No agreement between the three primary descriptors 11-7

3. Conclusions 11-10
SECTION III

CHAPTER 12 : TOM - Practical Implementation

1. Introduction 12-1

2. TOM Class Definition 12-2
   - Initialize the voice card 12-2
   - Set the voice gain 12-2
   - Create a LPC coded voice template 12-2
   - Compare a new template with one saved in memory 12-2
   - Shut down the voice card. 12-2

2.1 The TOM Model Class 12-3

2.2 The TOM Zero Crossings Formatter 12-3

2.3 The TOM Kernel 12-4

2.4 The TOM System class 12-4

3. The Windows Application 12-6

3.1 The Main Menu 12-6

3.2 The FILE Menu 12-8

3.3 The Template Menu 12-11

3.4 The ANALYZE Menu 12-13

3.5 The HELP Menu 12-15

4. System Specifications 12-16
5. Conclusion

The Epilogue

Appendix A

Appendix B

Bibliography

Index
CHAPTER 1: Voice And Speech Recognition
"Due to the extreme sensitivity and delicate sensibilities of machines and to safeguard against blowing fuses, it shall be mandatory that robotbrains in such machinery, on hearing any cursing or lewd words, substitute for such word or sound 'BLEEP'. No Machine, even if pounded upon, may reproduce swearing or lewdness in any other way than 'BLEEP' and if further efforts are made to get the machine to do anything else, the machine has permission to pretend to pack up. This bylaw is made necessary by the in-built mission of all machines to protect biological systems from themselves."

(The machine Purity League, Bylaw AD-991002012-FDE) "[1]

1. The Future and Voice Recognition

The obsession creators of science fiction have with the ultimate computer, a complete human replica that is capable to perform any human function, with the same or even better performance, has influenced our time and the way we perceive the future.

This human replica could be the perfect companion, friend or servant. It could be asked to execute tasks which humans find dangerous or demeaning and with the problems the human race created for itself such as pollution that could eventually kill everything on this planet, the human race has to look at the rest of the universe for some place to go. Space exploration will be extremely demanding on the human body and soul. Even with vastly greater technological powers, it would still take considerable time for us to travel to our nearest neighboring planet. Even then we cannot envisage what it would take to travel out of our own galaxy to the next.

In these circumstances a human replica could be the ideal astronaut. It would not suffer any human biological problems, such as nourishment and the need for other human company.

This concept has been tried before, if we think of the Voyager space plan, where a "drone" was sent to our neighboring planets to collect information about them and then relayed this information back to earth. An interesting concept employed was to build into this "drone" information regarding earth, everything from animal sounds to drawings of the human body. The plan was to send the "drone" beyond our galaxy after its mission was completed, never to return to earth again.

If it was technologically possible to stay in communication with earth, this "drone" could have contained a human replica, able to relay the sights it sees in human speech,
a replica that can communicate and understand whatever it may encounter in deep space.

In order to facilitate this wonderful relationship, the replica must be able to communicate with its human masters using human communication systems, or more simply one or more human languages.

As this replica is a computer, we could expect it to understand and speak a vast number of human languages and it should be able to "learn" any form of communication within seconds, such as the C3PO character in Star Wars is able to do.

Even the fact that we presume that this replica will be a binary computer, is once again an indication of how we perceive the future. Personally, we believe that binary computers as we know them today will not be able to perform the extensive task required from such a replica. The future will need something in the form of a "mix". With this we perceive a computer that is part biological part electronical.

After completing the study that we undertook concerning this subject, it became clear to us that the technology to implement this concept is not yet with us. Keeping this in mind we can take as an example the Japanese Fifth Generation.[5]

To return to the subject domain of this publication, that is human language interpretation, synthesis and implementation by a computer, we found that there were a few negative factors that delayed significant progress.

The first negative factor is that there seems to be no agreement or standards regarding the subjects of voice and speech recognition. Most experts in this field do not even agree on a subject definition.[5,9,13]

The second negative factor is the human subjects that played their part in delaying progress in this study field. The human problems encountered in voice and speech recognition can be briefly summarized as follows:[4]

- **Speaker variations**

  No two people sound alike. The speech signal contains talker-dependent variables as well as phonetic information and it is, as we found, not always easy to separate them.

- **Ambiguity**
The phonetic sounds do not always agree with the acoustical sounds. This is most commonly encountered when one person asks the other for the correct spelling of a word. ("Is that N for November or M for Mike?")

- Variations in individual speech.

People do not talk any more distinctly than is absolutely necessary. This is related to the worst trait of the human species, unproductiveness. Therefore short words are frequently reduced to grunts and syllable breaks are often obscured.

- Noise and interference[9]

Humans can recognize speech at poor signal-to-noise ratios in the presence of interfering speech and or sound.

A large part of this capability depends on our ability to filter sound input, a process which is still not fully understood.

The presence of noise degrades the performance of any speech or voice recognizer and the way the human brain handles noise can not, at this stage, be fully mimicked.

Despite these negative influences, there have been significant advances in this study field. We will discuss these advances as well as the basic methods employed in converting the analog speech signal into a digital, computer understood, format.

But first we need to clarify the composition of the subject to the less informed reader.

2 Voice recognition

We will start this section with a classification of Voice recognition. Thereby trying to explain in detail, our understanding, of the composition of the subject.

We would like to introduce the terms that will be used further on in this document as well. We accept that these terms, do not always agree with other references, but we ask the reader to bear with us.

We understand Voice recognition to be composed of two distinct subjects, these are Speech recognition and Speaker recognition. Speech recognition can be further classified into isolated word recognition and continuous speech recognition.
We will discuss these subjects in full detail later on in this section. But let us first define the term Voice Recognition as the study of a digital or analogue signal, the interpretation of this signal and finally the reproduction of this signal in a synthesized format.

With this definition we seem to find that there is a process that is meticulously followed, that is, the conversion of the analogue speech signal into a digital format, the interpretation of this digital signal by electronic means and finally the reproduction of the analogue signal using non-human methods.

Both the sub domains of Speech recognition, that is where human speech is understood and reproduced by an electronic mechanism, as well as Speaker recognition, where the human uttering the sound is uniquely identified, are contained in this definition.

We will discuss the process which is followed in voice recognition in the following chapters. After which we will explain the problem domain and the research done on this subject in the latter chapters. We decided upon this format, as we would like to give the reader a knowledgeable understanding of the underlying concepts of voice recognition before we move on to our own problem.
1.3 Speech Recognition

There is an enormous body of literature on the subject of speech recognition and we cannot hope to give it all adequate coverage here. In this section, we will describe the particular problem areas in each type of speech recognizer and describe some of the more important techniques.

Speech recognition is the process of automatically extracting and determining linguistic information conveyed by the speech signal using computers made of electronic circuits.[2]

In the broadest sense of the word, speech recognition includes or encompasses speaker recognition which involves extracting individual information indicating who is speaking. We have seen it fit to separate these two subjects from each other, however many of the techniques used in Speech recognition is also employed by Speaker recognition.

Automatic speech recognition methods have been investigated for many years, aimed principally at realizing typewriters and robots capable of recognizing speech.

The first technical paper to appear on speech recognition was published in 1952. It described Bell Laboratories' spoken digit recognizer Audrey[2]. Research on speech recognition has since intensified, and speech recognizers for communicating with machines through speech have recently been constructed, although they remain only of limited use.

Speech communication with machines feature four specific advantages [4]:

- Speech input is easy to perform because it does not require a specialized skill as does typing or push-button operations

- Speech can be used to input information three or four times faster than typewriters and eight to ten times faster than handwriting.

- Information can be input even when the user is moving or doing other activities involving the hands, legs or eyes.

- Since a microphone or telephone can be used as an input terminal, inputting information is economical, with remote inputting capable of being accomplished over existing telephone networks.
Regardless of these positive points, however, speech recognition also has disadvantages. These include the fact that noise cancelling is necessary when used in a noisy environment.

The process followed in typical speech recognition systems is where inputted speech is compared with stored reference templates and the most similar reference templates are selected as candidates. Since speech wave forms are too complicated to compare, it is desirable to remove the phase components, such as vowels or phrases, from the speech wave before the recognition phase is executed.

Before any concepts in voice and speech recognition are further discussed the reader should be informed as to the biological functions employed by humans to create voiced sounds. Therefore in the next chapter we will discuss these biological functions under the heading of articulatory phonetics and phonemics.

2. About this dissertation.

This document titled "Voice classification - using a unique key signature" consists of three distinct sections. The first section explains the terms and procedures followed in the subject domains of voice recognition and synthesis. It should be seen by the reader as a study of current literature on these subjects. As this is a document for a commercial degree, we will limit the detail technical discussions, such as mathematical formulae, as much as we are allowed to do.

Section two is concerned with the problem of voice classification systems. In this section we will define and explain our own classification model called the Template Order Model (TOM). Each chapter in this section will discuss a different layer in this model and it will be accompanied by a source code example.

Section three consists of an explanation of the practical implementation of the Template Order Model that accompanies this document.
CHAPTER 2: Articulatory Phonetics And Phonemics
In this chapter we will supply the reader with an introduction into the world of phonetics and phonemics. We will further discuss the way in which these sounds can be captured using the sound spectrograph.

1 Introduction into the field of phonetics

The term phonetics is used for the study of speech sounds as they are perceived and thought of by speakers of a particular language.[4]

Phonetics can be divided into two distinct subjects, that is, articualr phonetics and acoustical phonetics.[4,5] The first subject considers how any given speech sound is produced with particular emphasis on anatomical detail. In the latter subject the emphasis is on observable, measurable characteristics in the wave form of speech sounds, especially those which enable them to be distinguished from one another. Acoustical phonetics thus provides theoretical and experimental background for speech recognition and synthesis by electronic hardware.

In this chapter we will consider articular phonetics in section 2.2 which will serve as a anatomical and linguistic groundwork for acoustic phonetics in section 2.3

2 Articulary Phonetics

The first task of articulary phonetics is to describe speech sounds in terms of the positions of the vocal organs when producing any given sound. An important goal is to provide a common notation and frame of reference so one linguist can understand another and reproduce accurately any unknown utterance which has been written down in "close phonetic transcription."[4]

2.1 Phonetic alphabets

We shall see that there is a wealth of different speech sounds, more than can be encompassed in any traditional alphabet. Hence phoneticians have had to devise their own system of notation. The oldest and most widely accepted notation is the international phonetics alphabet (IPA)[9]. It dates from a time when type was hand-set and derives many of its symbols by printing Roman characters upside down or by borrowing from the alphabets of other languages. These symbols clearly cannot readily be reproduced on most computer printers, hence in recent years a substitute has been developed, called the Arpabet by the Advanced Research Projects Agency (ARPA) of the American department of Defence.[9]
Fig 2.1 The human lungs, diaphragm and trachea
The Arpabet comes in two different versions, the first is a single-character format which uses lower-case characters for some sounds and a two-character version to accommodate printers which have no lower-case characters.

The corresponding Arpabet symbols are listed in table 2.1.[4]

We will use the IPA notation in this chapter for both phonetics and phonemics. The convention is to enclose the character in square brackets, [b], in phonetic notation and phonemes are written between solidi, /b/.

Table 2.1 The Arpabet Phonetic Alphabet

<table>
<thead>
<tr>
<th>Arpabet</th>
<th>Example</th>
<th>Arpabet</th>
<th>Example</th>
<th>Arpabet</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</td>
</tr>
<tr>
<td>AA</td>
<td>hod</td>
<td>K</td>
<td>kick</td>
<td>EM</td>
<td>bottom</td>
</tr>
<tr>
<td>AO</td>
<td>hawed</td>
<td>G</td>
<td>gig</td>
<td>EN</td>
<td>button</td>
</tr>
<tr>
<td>OW</td>
<td>hoed</td>
<td>F</td>
<td>fife</td>
<td>DX</td>
<td>batter</td>
</tr>
<tr>
<td>UH</td>
<td>hood</td>
<td>V</td>
<td>verve</td>
<td>Q</td>
<td>*</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>----</td>
<td>-------</td>
<td>----</td>
<td>-------</td>
<td>---</td>
<td>-----</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>pizazz</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>

*Glottal Stop*

2.1.1. Categories

Speech sounds are conventionally divided into vowels and consonants. Although these terms are also used in phonetics, it is difficult to define them precisely. Attempts at definition usually involve questions on their function within specific languages and thus encroach on the territory of phonemics.[4]

For the purpose of this document we will follow most writers in using the terms vocoid and contoid.

Vocoids are characterized by phonation and a relatively unobstructed vocal tract, and their most important feature is the tone color imparted to the sound by resonances in the vocal tract.[4,9]

Contoids are characterized by obstruction of the vocal tract. Here phonation is of secondary interest and the most important feature is audible turbulence or other interruption of the speech stream.[9]
The reason for the vocoid/contoid problem is the fact that in many cases the same sound can function as either a vowel or a consonant. An example is in the case of "you" and "eat". Both begin with very nearly the same speech sound, but in "eat" it is a vowel, followed by the consonant /t/, while in "you" the sound is a consonant followed by the vowel /u/.

In this example the sounds are unmistakably vocoids, and their vocalic or consonantal roles are simply a question of application in a particular context.

2.1.2 Contoids and Consonants

Consonants are relatively easy to define in anatomical terms. Most consonants can be described by a few well-recognized features, these are:

- Point of articulation

- Manner or articulation

- Voicing (phonation)

2.1.3 Point of articulation

This is the location constrictions in the vocal tract, defined in terms of participating organs. Table 2.2 is a list of the principal points of articulation and the names given to the corresponding consonants.
<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bilabial</td>
<td>Between lips</td>
</tr>
<tr>
<td>Labiodental</td>
<td>Between lower lip and upper teeth</td>
</tr>
<tr>
<td>Apicodental</td>
<td>Tip of tongue on teeth</td>
</tr>
<tr>
<td>Apicogingival</td>
<td>Tip of tongue on lips</td>
</tr>
<tr>
<td>Apicoalveolar</td>
<td>Tip of tongue on alveolar ridge</td>
</tr>
<tr>
<td>Apicodomal</td>
<td>Tip of tongue on hard palate</td>
</tr>
<tr>
<td>Laminoalveolar</td>
<td>Blade of tongue on alveolar ridge</td>
</tr>
<tr>
<td>Laminodomal</td>
<td>Blade of tongue on hard palate</td>
</tr>
<tr>
<td>Centrodomal</td>
<td>Middle of tongue on back of hard palate</td>
</tr>
<tr>
<td>Dorsovelar</td>
<td>Back of tongue on velum</td>
</tr>
<tr>
<td>Pharyngeal</td>
<td>Root of tongue constricting pharynx</td>
</tr>
<tr>
<td>Glottal</td>
<td>Between vocal cords</td>
</tr>
</tbody>
</table>
2.1.4 Manner or articulation

This is principally the degree of constriction at the point of articulation and the manner of release into the following sound. Table 2.3 consists of a list of these releases.

Table 2.3 The principle categories of articulation[5]

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plosive</td>
<td>Vocal tract shuts off at point of articulation</td>
</tr>
<tr>
<td>Aspirated</td>
<td>Vocal tract initially shuts. Release marked by puff of air.</td>
</tr>
<tr>
<td>Affricative</td>
<td>Initial closure of vocal tract followed by gradual release producing turbulence</td>
</tr>
<tr>
<td>Fricative</td>
<td>Vocal tract partially open at point of articulation.</td>
</tr>
<tr>
<td>Lateral</td>
<td>Vocal tract closed at point of articulation but open at the sides</td>
</tr>
<tr>
<td>Semivowel</td>
<td>Vocal tract partially open at point of articulation without turbulence</td>
</tr>
<tr>
<td>Nasal</td>
<td>Vocal tract closed at point of articulation but velum open.</td>
</tr>
<tr>
<td>Trill</td>
<td>Oscillatory opening and closure at point of articulation</td>
</tr>
</tbody>
</table>

2.1.5 Voicing

This indicates the presence or absence of phonation. Even during a stop, it is possible to force air through the vocal cords for a short time. Consonants accompanied by voicing are called voiced, the others unvoiced. There are fine points here which allow for semi-voiced sounds as well.
The perception of voicing is influenced by other factors in addition to phonation, which we will not discuss further.[5]

These three features, voicing, point of articulation and manner of articulation, provide a terminology by which we can define any contoid. The description of a consonant can therefore be a virtual formula for producing the sound.

2.1.6 Vocoids and Vowels

The first thing to note about vowels is that they are much less well defined than consonants. This is because the tongue typically never touches another organ when making the sound. Therefore there are no specific landmarks corresponding to the point of articulation, and vowels are somewhat vaguely defined in terms of position, in other words the distances from other parts of the mouth.

Vowels are described by the following variables:[5]

- Tongue high or low
- Tongue front or back
- Lips rounded or unrounded
- Nasalized or unnasalized.

These parameters are mostly self-explanatory. However the terms, concerned with vowels, that do merit further discussion are the following:

- Nasalization
- Diphthongs
- Coarticulation

2.1.7 Nasalization

In the pronunciation of any vowel, the velum (the portion of the top jaw located at the back of the throat) can be open or closed. If it is closed, the nasal cavity is for all practical purposes disconnected from the system and the vowel sound is determined only by
the position of the tongue and lips. If however the velum is open, the sound passes through the nasal cavity as well. This cavity has its own acoustics and give the vowel a characteristic color. Such sounds are said to be nasalized.[5]

In most languages, no distinction is made between nasalized and unnasalized vowels. The speaker is free to nasalize vowels or not as idiosyncrasy or dialect dictates.[4]

Nasalization is a feature that most computer scientists would like to avoid, it is therefore not surprising that some electronic hardware devices that transform the analogue sound into a digital format tries to filter out the effects of nasalization such as they would sound interference.[9]

2.1.8 Diphthongs

It is possible to combine two vowels sounds in a single syllable by moving the tongue from one position to another. Such a combination is called a diphthong.[4,5]

An example of this in English are the vowels in "boy", "few", "now".

2.1.9 Coarticulation

Everything we have said about phonetics so far is misleading in one respect. It suggests that every phone, that is the smallest individual sound in phonetics, is executed perfectly and uniformly and in a manner independent of context.

Language learning, and the synthesis and recognition of speech, would be a simple task if this were so. In fact, no speech sound is produced accurately in the context of other sounds. Instead, each phone can be considered as a target at which the vocal organs aim but which they never reach.

As soon as the target has been approached close enough to be intelligible to the listener, the organs change their destinations and start to head for a new target. This is done to minimize the effort expended in speaking and makes for greater fluency.
2.2 Phonemics

Whereas phonetics is a view of speech sounds considered in isolation from any language, phonemics is the view from within some specific language[5]. It would be only a slight overstatement to say that phonetics is not really a division of linguistics but rather a resource which linguistics draws upon. Phonemics, on the other hand, is decidedly a branch of descriptive linguistics.

2.2.1 Phonemes

In phonetics, an individual sound is a phone, in phonemics, the smallest unit is the phoneme.

We will take the following as a working definition of the phoneme, to be expanded presently:

A phoneme is the smallest sound unit in a given language that is sufficient to differentiate one word from another[6].

If changing a particular phone in an utterance changes the utterance's meaning, then the changed phone is also a phoneme. Similarly, if changing a phonetic variable changes the meaning of a word, then the variable marks a distinction between two phonemes. If however it fails to change the meaning of the word, then we say that feature is not phonemic in that language.

2.2.2 Allophones.

We can relate phonemes to phonetics by observing that a phoneme is actually a set of phonetically similar sounds which are accepted by the speakers of the language as being the same sound.

Members of the set are called allophones.

One of the reasons we speak foreign languages with an accent is the fact that we unconsciously impose the phonemic organization of our native language, complete with its allophones, on the other language.[4]
The study of acoustic phonetics dates back to the eighteenth century, when von Kempelen devised the first known speech synthesizer,[5] a bellows-driven device in which the vocal tract was modeled by a flexible leather tube whose shape was controlled by the operator's hand.

The rapid growth of modern acoustic phonetics, however, dates roughly from the invention of the sound spectrograph at Bell Telephone Laboratories in 1941[10]. The importance of this invention is hard to overestimate[5].

Fig. 2.2 The sound spectrograph [10]
Newer techniques have supplemented the sound spectrograph, but not supplanted it. We will therefore introduce this section with a brief description of this device. [10]

3.1 The sound spectrograph

In its simplest form, the sound spectrograph consists of a loop of magnetic tape, attached to the rim of a large disk with a tunable analyzing filter and a drum around which a sheet of recording paper is wrapped [4] (see Fig. 2.2).

In the "record" mode, speech is recorded onto the tape. In the "analysis" mode, the speech is played back repetitively at high speed and analyzed by the filter.

On each repetition, the filter tuning is changed slightly so a different band of frequencies is analyzed. The output of the filter is sent to a stylus which darkens the paper proportionately to the amplitude of the filter output. The filter tuning and the stylus positions are linked together so that each new analysis appears in a new position on the paper.

The result of the analysis is a graphic representation of the frequency content of the speech versus time, with frequency on the vertical axis and time on the horizontal axis, as with musical notation. The intensity of any given frequency component at any time is indicated by the darkness of the corresponding spot on the paper.

The spectrograph offers the user a choice of two bandwidths, the first called narrowband at a frequency of 45 Hz and the second is called wideband at 300 Hz [4,10]. Since essentially all voices have fundamental frequencies greater than 45 Hz [4], narrowband spectrograms show the pitch and its harmonics as horizontal lines. Since most speakers' pitches are lower than 300 Hz, however, the pitch harmonics are not resolved in the wideband mode. On the other hand, individual glottal pulses are visible and the formants (the resonances of the vocal tract) show up as dark bars.

To appreciate the importance of this invention, we must consider it in the light of technology previously available.

High-speed spectrum analysis was not available until the 1950s and was not available cheaply until the introduction of the fast Fourier transform [4,5,6,9] in 1965. Previously only hand computation or filterbank analysis was available. Hand computation was prohibitively laborious and filter banks did not give satisfactory resolution.
Fig. 2.3 The Human vocal cords and their supporting cartilages [4]
3.2 Acoustics of the vocal tract [4]

Acoustically, the vocal tract is a tube of nonuniform cross sections, approximately 17 cm long in adult males, usually open at one end and nearly closed at the other.

Such a tube is a distributed-parameter structure, meaning that there are a great number of frequencies possible. If the vocal tract was made of uniform cross sections, the frequencies would occur in groups of 500 Hz.

In the real world these sections are not uniform. As a result the resonances are not equally spaced, but the average density of the vocal tract resonances is still approximately 1 per kilohertz of bandwidth.

These resonances are known as formants and are the most important acoustical characteristics of the vocal tract.

Formants thus provide the listeners with a primary source of information about the position of the speaker's vocal organs.[4,5]

Formants are identified by number in order of increasing frequency, usually named F1, F2 and so forth. In classical acoustic phonetics, usually only F1 and F2 are considered. For recognition at least three are meaningful and for synthesis, five are recommended for greatest realism.[5]

3.3 Analysis of the acoustics of vocoids.

Because of the importance of formants, much effort has been devoted to analyzing vocal tract acoustics, particularly with a view to relating specific vocal tract shapes to the corresponding formant frequencies.

The problem is complicated by the fact that the area of the vocal tract varies irregularly along its length. Exact analysis of nonuniform acoustic tubes is possible only in a few special cases, therefore it is necessary to resort to some simplification.

The most popular model is called the piecewise-cylindrical model[3,4,5,9], in use from the 1950s. Its main advantage is that a cylindrical tube made of uniform sections is easy to understand, model and analyze.
A tube of uniform cross sections is the acoustical analogue of an electrical transmission line. As with this type of line, waves can travel in either direction in the tube, and these waves are governed by WAVE EQUATIONS [4].

The difference is that waves in an acoustical tube are waves of pressure and velocity, instead of voltage and current [4,5].

3.4 Properties of vowel waveforms

Vowel waveforms constitute two distinct characteristics, the first is Time-domain [4,5,9] characteristics and the second is Frequency-Domain [4,5,9] properties.

Time-domain characteristics consist of physical boundaries between glottal pulses, it is therefore possible to predict the appearance of the next pulse.

Frequency-domain properties include a widely used technique, called the spectral-envelope [3,4,5,6,7,9]. This envelope is extracted from the pulse wave by connecting the pulse peaks. Recovering the spectral envelope is a major part of many speech-processing applications, since it is the main source of articulatory information. Much of the importance of linear prediction [4,5,9], discussed in latter chapters, arises from the fact that it provides a fast, accurate and theoretically justifiable means of recovering this envelope.

3.5 Acoustical characteristics of stops and fricatives.

Acoustically, a fricative appears as a stretch of more-or-less wideband noise. A stop appears as a short period of silence followed by an abrupt release. The release is often not clean and usually shows up as a very short burst of noise with an abrupt onset.

If voicing is present during a stop or fricative, then a formant-like structure may also be visible in the spectrum of sound. Acoustical characteristics of stops may be divided into two groups [4]

- The first is the spectral makeup of noise
- The second is formant transitions
3.5.1 Noise

Various researchers have investigated the acoustical characteristics of fricative noise. Their findings are not entirely unanimous, probably because of differences in language, experimental methods and context. The aspects they do agree upon can be summarized as follows [4,5,9]:

- Dental noise

  In sounds like [t,d,s] with high frequency and high energy

- Labial noise

  This type of noise is found in sound such as [p,b] and is generally low frequency and low energy.

- Velar noise

  Such as found in [k,x]. It is of medium frequency and medium energy.

3.5.2 Formant transitions

In early experiments with sound spectrograms, it was noticed that formant tracks were curved in the neighborhood of consonants. These curves are called formant transitions and they were extensively investigated in a series of classic studies at Haskins Laboratories in the 1950s. Their experimental methods were to synthesize speech from drawings of speech spectrograms using a device called the pattern playback.[4]

The pattern playback function is an inverse sound spectrograph. This device permitted researchers to make speech to order with any desired spectral content.

They could test hypotheses by creating speech as desired and having it evaluated by listeners. Using this technique they reached the following conclusions[4]:

- The perceived stop was strongly effected by transitions in formants.

- The formant transition for each point of articulation was characterized by a target frequency largely independent of the following vowel.
Formant transitions generally present a more orderly and detailed pattern than do the noise bursts. It is clear that they are caused by the movement of speech organs from the preceding vowel position to the consonant position and from the consonant position to the following vowel.

This leads to the fact that because of formant transitions, more detailed articular information is to be found in the adjacent vowels than in the consonants themselves.

4. Conclusion

This then concludes our journey into the world of acoustics. We will consider the ways in which the analogue voice signal can best be interpreted by a computer in the next chapter called the Digitized Human Voice Print.
CHAPTER 3: The Digitized Human Voice Print.
In this chapter we will discuss the ways in which the analog voice signal is converted into a digitized format. This transformation is known as Analogue-to-digital conversion or A/D conversion. We will see that the whole process of A/D conversion, or digitization, consists of three distinct processes, namely: sampling, quantization and coding.

1. Digitization - An Introduction

The speech signal, or speech wave, can be changed into a computer manageable object by converting it into an electrical signal using a microphone.[4]

The electrical signal is usually transformed from an analog into a digital signal prior to almost all speech processing for the following two reasons:[9]

- Digital techniques facilitate highly sophisticated signal processing which cannot otherwise be realized by analog techniques.

- Digital processing is far more reliable and can be accomplished by using a computer.

Analog-to-digital conversion, commonly referred to as digitization, consists of sampling, quantizing and coding processes.[5]

Sampling is the process of depicting a continuously varying signal as a periodic sequence of values. Quantization involves approximately representing a waveform value by one of a finite set of values. Coding concerns assigning an actual number to each value.[4,5,6,9,13]

These processes enable a continuous analog signal to be converted into a sequence of codes selected from a finite set. The combination of these processes form digitization.

The processes of Sampling, Quantization and Coding will be discussed in more detail in this chapter, however all the mathematical equations discussed in this chapter will be assumed correct and no mathematical proofs will be supplied. The reader is referred to our list of publications at the end of this dissertation.

2. Sampling

In the sampling process, an analog signal $X(t)$ is converted into a sampled sequence of values $\{X_i\} = \{X_i(T)\}$ at a periodic time interval $t_i = iT$ (i is an integer).
Here T \([s]\) is called the sampling period, in milliseconds, and its reciprocal, \(S=\frac{1}{T}[Hz]\), is termed the sampling frequency. If the sampling period is too large, the original signal cannot be reproduced from the sampled sequence. The same is also true if \(T\) is too small, unless samples from the original signal reproduction are included in the sampled sequence. [6,13]

These two problems can be overcome by the use of the Sampling theorem, due to Shannon and Weaver [6,14] that states that any continuous-time function is exactly determined by equally spaced samples provided that the sampling rate is at least two times as high as the highest frequency component present in the function. This minimum acceptable sampling rate [4,6,14,15] is commonly referred to as the Nyquist rate. (The rate was determined by Nyquist in 1928, although the general proof was published by Shannon 1948 [16])

Fig 3.1 explains these concepts graphically. [6]

Functions sampled at a frequency less than the Nyquist rate are said to be under-sampled. In an under-sampled signal, any frequency component at a frequency larger than the sampling rate divided by two will appear to be at a lower frequency. This phenomenon is known as aliasing. [4,5,6,14,16]

If a signal is sampled at a rate well above the Nyquist rate, it is said to be over-sampled. If the sampling rate is some multiple of the Nyquist rate, the rate can be reduced by discarding the redundant samples. For example, if the original rate is three times the Nyquist rate, then every third sample can be retained and the rest discarded. This process is known as down-sampling. [4,5,6,14,16]

When data is sampled and digitized for computer processing or analysis, it is a standard precaution to precede the sampling device with an anti-aliasing filter. This filter rejects any frequency components higher than the Nyquist Rate. It is the responsibility of the user, however, to make sure that the sampling frequency is high enough so that no essential information will be destroyed by the anti-aliasing filter.
Fig. 3.1 Sampling in the time domain. From Fig 4.1 [6]
Quantization consists of an analog-to-digital conversion. In this process an analog pulse is converted into a digitized format so that it can be readily processed by a computer.[4,5,6]

The two possible methods of quantization are called Pulse-code modulation and Delta modulation. In Pulse-code modulation[4,6], analog signals are quantized in homogeneous steps, a process similar to the Analog-to-digital conversion. With Delta modulation, an extreme method of differential quantization, the sampling frequency is raised so high that the difference between adjacent samples can be approximated by a 1-bit representation[5].

3.1 Pulse-Code Modulation (PCM)

In this method the amplitude range of the sampled data is divided into a finite number of discrete levels. The amplitude of a given pulse is referred to the nearest level and a digital code is generated.

The analog-to-digital conversion introduces an irremovable error known as quantization noise which depends upon the number of the levels and hence upon the number of digits in the code. The larger the number of levels the smaller the noise.

The effect of this error can be reduced by using nonuniform spacing of the levels to suit the statistical properties of the signal.

An adoption of this method can be used to improve the signal-to-noise ratio. It is known as the Adaptive Pulse-code Modulation method (APCM).[6]

With this method the speech signal is considered as stationary for a short time period, thus the step size can be varied relatively slowly.

There are two methods for varying the step size. In the first method, which is called the forward or feedforward adaptation method, the step size is changed at every block. In the second method, known as the backward or feedback adaptation method, the step size is changed on a sample-by-sample basis according to the decoded samples.
3.2 Delta Modulation

This method is advantageous in its simple structure, is based on the fact that the correlation between adjacent samples increases as a function of the sampling frequency, except for uncorrelated signals. As the correlation increases, the prediction residual decreases. Therefore, a coarse quantization can be used when signals are sampled at a high frequency. [6]

A high prediction gain can thus be obtained by such a differential coding structure. In the decoding of a delta-modulated signal, a value (D) is simply added to or subtracted from the previous sample according to the positive or negative signal.

The method in which (D) is fixed is called Linear Delta Modulation (LDM) [6]. (See fig. 3.2) In this method, when the speech amplitude becomes too large or changes too rapidly, the reconstructed sample does not exactly follow the original signal. Distortion in this case is referred to as Slope Overload Distortion. On the other hand, when there is no speech, that is, during a period of silence, or when the speech wave changes only slightly and very slowly, the quantization output alternates between zero (0) and one (1).

The encoded waveform thus indicates an alternating increase and decrease with the stepping of delta (D).

This type of noise or distortion is referred to as granular noise. [4, 5, 6]

To compensate for this problem a more effective quantization method called Adaptive Delta Modulation (ADM) [6] was introduced. In this method the value of delta (D) is changed with respect to the input speech waveform. Most of the various ADM methods are based on the backward or feedback techniques, in which the minimum step size is adjusted according to the output code sequence.

Since the step size is varied in Adaptive Delta Modulation (ADM), the granular noise in ADM is smaller than those in LDM. [6]

4 Coding

There exists a fine line between coding and quantization. In the literature these two processes are seen as one [4] or as two [5, 9] distinct subjects. For the purpose of this do-
Slope overload distortion

Step size (D)

Sampling period

(A) Linear Delta Modulation

(B) Adaptive Delta Modulation

Fig. 3.2 Illustration of Delta Modulation. From Fig. 6.5 [6]
cument we shall make a distinction between these processes by implying that coding is a refinement process of quantization.

Coding can be defined as the allocation of an actual number to an approximate representation of a digitized speech waveform.[5,6]

Principle coding techniques can be classified into reversible coding which is not accompanied by information loss and irreversible coding which is so accompanied.

Reversible coding is based on Shannon's information sources theory[6,14,15], which implies that the coding efficiency is limited by the occurrence of the information source. This means that when the occurrence probability of each code is not homogeneous, the bit rate can be reduced by variable length coding.

Although no information is lost with reversible coding, a certain amount of distortion is usually permitted in speech coding as long as auditory comprehensibility is not impaired.

Irreversible coding is based on the Rate distortion theory of Jayrant and Noll[17]. When a certain information source is coded so that the distortion is less than a certain value $D_s$, the average code length $L$ for each information source has the lower limit of $L = R(D_s)$. On the other hand if the information rate $R$ is given, the lower limit of quantization distortion $D_s(R)$ exists. The lower limits of $R(D_s)$ and $D_s(R)$ are referred to as the rate distortion function and distortion rate function, respectively.[17]

The rate distortion theory has no practical application, for the following two reasons:[6]

- $R(D_s)$ and $D_s(R)$ are very difficult to calculate except for very simple cases.

- The actual coding methods cannot be derived from this theory.
4.1 Irreversible coding methods

The basic techniques for irreversible coding methods can be summarized by the following:[6]

- **Nonlinear quantization**

  The amplitude of the speech signal is compressed by nonlinear transformation or logarithmic transformations. These techniques are based on the statistical characteristics of the speech amplitude.

- **Adaptive quantization**

  This method concentrates on the amplitude dynamics of the speech signal.

- **Predictive coding**

  In this method the transmission bit rate can be compressed by utilizing the correlation between adjacent samples as well as distant samples in a speech wave.

- **Time and frequency division**

  Speech information is divided into several time periods or several frequency bands. This method makes use of this speech characteristic.

- **Transform coding**

  A speech wave in the time domain is converted to a frequency domain and then it is encoded with this method.

- **Vector quantization**

  Instead of coding individual samples, the information source sample sequence or group or block is coded all at once as a vector.

Each of the above mentioned techniques will not be explained further, it is sufficient for our purposes only to know that these methods exist.
5. Conclusion

This then concludes our chapter on digitization. This chapter was intended to be an introduction into this science. It would be impossible for us to cover the subject in its entirety. Therefore we tried to explain some of the basic concepts, so as to give an understanding of relevant concepts and terminology.

In chapter 4 we will discuss a very important coding method called Linear Predictive Coding or LPC, as it forms part of most voice systems in use today.
CHAPTER 4: Linear Predictive Coding (LPC)
In this chapter we will discuss a very important coding method called Linear Predictive Coding (LPC). This method is used extensively in almost all voice recognition systems. It is not implemented in our model, which we will define in chapters 7-11. We however felt that LPC should be explained in detail to the readership, as no document on voice recognition can be complete without a reference to this coding method.

1. Introduction

In the early 1970s, within just a few years, linear predictive coding became by far the most popular method for the digital analysis and synthesis of speech signals. In this chapter, we examine the basic methods and properties of linear predictive analysis and synthesis. The discussion will hopefully give the reader an appreciation of the time-domain and the frequency-domain aspects of linear prediction and some insights into its workings.

The chapter begins in section 2 with a description of the basic all-pole speech synthesis model upon which Linear Predictive (LP) analysis is based. Although the most general LP model contains zeros as well as poles, we shall restrict our attention to the all-pole model since it is, practically speaking, the only model in serious use. In section 3 we discuss various direct-form LP analysis methods and present the fundamental properties of linear prediction. Section 4 is devoted to the lattice analysis and synthesis methods of linear prediction. Finally in section 5 we discuss the use of LPC in speech recognition.

The basic idea behind linear predictive coding is that the speech sample \( s(n) \) is predicted from the sum of \( p \) linearly weighted previous samples:

\[
s'(n) = a_1 s(n-1) + a_2 s(n-2) + \ldots + a_p s(n-p)
\]

where the set of \( a_k \) are the prediction coefficients \([4.0] [13]\) and \( 1 \leq k \leq p \).

2. The basic synthesis model.

Fig. 4.1 shows a schematic diagram of the basic model used in LP synthesis. The model has two major components: a flat-spectrum excitation source [4,5,13,18,19] and a spectral shaping [4,5,13,18,19] filter \( H(z) \). The excitation source generates a signal \( u(n) \) with
a flat spectral envelope, which is used to drive the filter $H(z)$ resulting in the synthetic speech signal $s'(n)$. Because the input to the filter $H(z)$ has a flat spectrum, the spectral envelope of the output signal $s'(n)$ will have the same shape as the spectrum of the filter $H(z)$. Therefore, for speech synthesis, we endeavor to set the parameters of the $H(z)$ filter on a time-varying basis such that its short-term spectrum is the same as the short-term speech spectral envelope we desire.

Given a particular speech signal $s(n)$, one can obtain its short-term spectral envelope by appropriate inverse filtering, as shown in fig. 4.2.[15] The parameters of the inverse filter $A(z)$ are adjusted, as will be described in section 3, such that the residual signal
$e(n)$ has a flat spectral envelope\cite{18,19,20}. In essence, $A(z)$ is a time-varying spectral whitening filter. If the excitation $u(n)$ in fig. 4.1 is set equal to the residual $e(n)$ and $H(z)$ is set equal to the inverse of $A(z)$ then the synthetic signal $s'(n)$ will be equal to the original $s(n)$.

![Diagram of spectral whitening of the speech signal by an inverse filter.](image-url)

Fig. 4.2 Spectral whitening of the speech signal by an inverse filter
2.1 The All-pole model

In LP analysis, $A(z)$ is given by equation 4.1,

$$A(z) = 1 + \sum_{k=1}^{p} a(k)z^{-k} \quad \ldots \ldots \quad (4.1) [19]$$

where the $a(k)$, $1 \leq k \leq p$ are known as the predictor coefficients, and $z^{-1}$ is the unit delay operator. The residual $e(n)$ is then given by

$$e(n) = s(n) + \sum_{k=1}^{p} a(k)s(n-k) \quad \ldots \ldots (4.2)[19]$$

Equation 4.2 can be transformed into a synthesis equation by rewriting it as

$$s(n) = -\sum_{k=1}^{p} a(k)s(n-k) + e(n) \quad \ldots \ldots (4.3)[19]$$

if we set

$$s'(n) = -\sum_{k=1}^{p} a(k)s(n-k) \quad \ldots \ldots (4.4)[19]$$

as the predicted value of $s(n)$ from the previous $p$ samples, then

$$e(n) = s(n) - s'(n) \quad \ldots \ldots (4.5)[19]$$

can be viewed as the *prediction error*.

Analogous to (4.3), the synthetic signal $s'(n)$ can be computed from equation (4.6)
\[ s'(n) = \sum_{k=1}^{p} \alpha(k)s(n-k) + Gu(n) \quad \text{(4.6)[19]} \]

where \( G \) is a gain factor [18,19]. It is clear from (4.3) and (4.6) that if \( Gu(n) \) is set equal to \( e(n) \), the synthetic signal \( s'(n) \) will be identical to the speech signal \( s(n) \). The synthetic equation (4.6) is known as direct-form synthesis, following digital signal processing terminology. In section 4, we will discuss a different form of synthesis: lattice synthesis [13,19].

In the \( z \) domain, the transfer representation by (4.6) is given by equation (4.7)[19]

\[ H(z) = \frac{S'(z)}{U(z)} = \frac{G}{A(z)} = \frac{G}{1 + \sum_{k=1}^{p} a(k)z^{-k}} \quad \text{.........(4.7)} \]

where \( S'(z) \) and \( U(z) \) are the \( z \) transforms of \( s'(n) \) and \( u'(n) \), respectively. Of interest below will be the autocorrelation \( R'(i) \) of the impulse response of \( H(z) \) which can be shown to obey the following equations

\[ R'(0) = -\sum_{k=1}^{p} \alpha(k)R'(k) + G^2 \quad \text{.........(4.8)} \]

\[ R'(i) = -\sum_{k=1}^{p} \alpha(k)R'(i-k), \quad 1 \leq |i| \leq \infty \quad \text{.........(4.9)} \]

The same equations apply if the input to the \( H(z) \) is a white noise source with a unit power instead of a unit impulse [13,18,19].

2.2 The source model

The major problem in using \( e(n) \) as the excitation signal in practise is the large number of bits required to store it. For example, at a sampling rate of 10 KHZ, if one quantized each sample to one bit only, the required storage would be 10 000 bits per second,
which for many commercial applications would be prohibitive. One effective solution to this problem has been to model [19] the excitation as coming from one of two sources

- A pulse source (buzz) [18, 19]
- A noise source (hiss) [18, 19]

The most effective form of a pulse source is the impulse response of an all-pass filter [19]. The simplest and most popular form is the single impulse. When the pulse source produces a sequence of pulses separated by a pitch period, it is known as a buzz source, and is used to synthesize voiced sounds [18].

The noise or hiss source is a white noise source that can be simply a random number generator producing random sample values with a flat spectral envelope. A noise source is used to synthesize unvoiced or fricated sounds [13]. While most sounds can be generated by either a pulse source or a noise source, there are some sounds, such as voiced fricatives /z/ and /v/, that are best synthesized by a combination of the two sources. If such a mixed-source model is used, the speech is found to be more natural sounding.

Coding the parameters of the source model [4, 6, 13] described above requires in the order of a few hundred bits per second only. The resulting speech quality is not quite as natural as using the residual signal for the excitation, but the vast reduction in bit rate more than offsets the loss in speech quality for many applications.

3 Direct-form analysis

In direct-form analysis, the predictor coefficients a(k) are computed as the result of the minimization of the energy in the residual e(n) [18]. This computation must be performed on a short-term basis so as to follow the speech dynamics. Below we describe two methods of effecting the short-term aspect of the computation: by windowing either the speech signal or the residual signal.
3.1 Data windowing

In this method[18,19], the speech signal $s(n)$ is first multiplied by a data window $w(n)$ before it is inversely filtered. The most popular data windows are finite in extent. Thus we form

$$x(n) = w(n)s(n), \quad 0 \leq n \leq N-1 \quad \text{(4.10)}$$

where we assume that the window is zero outside the interval $0 \leq n \leq N-1$. The window width $N$ is usually set to correspond to 20-30 ms for short-term analysis. The most popular data windows used in speech analysis are the Hamming and Hanning windows[13,19]. Both raised cosine types of windows.

The residual signal whose energy is to be minimized is obtained by passing $x(n)$ through the filter $A(z)$. The residual energy given by

$$E = \sum_{n=-\infty}^{\infty} e_x(n) = \sum_{n=-\infty}^{\infty} [x(n) + \sum_{k=1}^{p} a(k)x(n-k)]^2 \quad \text{(4.11)}$$

where $e_x(n)$ is the residual corresponding to the windowed signal $x(n)$. $E$ is also the total squared error in predicting $x(n)$ from its past. The coefficients $a(k)$ that minimizes $E$ are obtained by setting the partial derivative of $E$ with respect to each of the coefficients $a(k)$ to zero. The result is the normal equation (4.12), where equation (4.13) is the autocorrelation of the windows signal $x(n)$. Equations (4.12) are $p$ linear equations in $p$ unknowns, which can be solved to obtain the desired predictor coefficients.

$$\sum_{k=1}^{p} a(k)R(i-k) = -R(i), \text{ where } 1 \leq i \leq p \quad \text{(4.12) [19]}$$

$$R(i) = \sum_{n=-\infty}^{\infty} x(n)x(n-i) = \sum_{n=i}^{N-1} x(n)x(n-i), \text{ where } 0 \leq i \leq p \quad \text{(4.13)}$$

This method is often called the autocorrelation method because the coefficients $R(i-k)$ in (4.12) are autocorrelation coefficients of the signal.[19] The minimum residual energy $E_p$ is obtained by substituting (4.12) in (4.11)
\[ E_p = R(0) = \sum_{k=1}^{p} a(k)R(k) \quad \text{...... (4.14)} \]

\( E_p \) is also known as the minimum total squared error, or simply the minimum prediction error.[4,5,6,13,18,19] For synthesis, if we set the gain of the filter \( H(z) \) such that

\[ G^2 = E_p \quad \text{...... (4.15)} \quad \text{[19]} \]

one can show by comparing (4.14) to (4.8) and (4.12) to (4.9) that

\[ R' = R(i), \text{ where } 0 \leq i \leq p \quad \text{...... (4.16)}\quad \text{[19]} \]

or the first \( p+1 \) autocorrelation coefficients of the windowed signal and the synthetic signal are equal. In particular, having \( R'(0) = R(0) \) means the energy in the synthetic signal is equal to the energy in the original windowed signal.

### 3.2 Residual windowing

In this case, the energy to be minimized is given by

\[ E = \sum_{n=-\infty}^{\infty} W_e(n)e^2(n) \quad \text{...... (4.17)} \quad \text{[19]} \]

where \( e(n) \) is the residual in (4.2) and \( W_e(n) \) is a window that weights the residual. Minimizing (4.17) with respect to \( \{a(k)\} \) results in the normal equations (4.18),

\[ \sum_{k=1}^{p} a(k)R(i,k) = -R(0,i), \text{ where } 1 \leq i \leq p \quad \text{...... (4.18)} \quad \text{[19]} \]

where

\[ R(i,k) = \sum_{n=-\infty}^{\infty} W_e(n)s(n-i)s(n-k) \]

The most popular residual window is the rectangular window.
We(n)={1, p≤n≤N−1 0 otherwise  ...... (4.19) [19]

in which case (4.1) reduces to equation (4.21)

$$R(i,k)=\sum_{n=p}^{N-1}s(n-1)s(n-k) \quad \text{where } 0\leq i,k \leq p \ldots \quad (4.21)$$ [19]

Since $R(i,k)$ provides an estimate of the covariance of the signal $s(n)$, this method is sometimes called the covariance method.[5,6,13,19,20] Equations (4.18) are $p$ linear equations in $p$ unknowns, which can be solved for the predictor coefficients.

Substituting (4.18) in (4.17), we obtain the minimum residual energy or the minimum prediction error

$$E_p=R(0,0)+\sum_{k=1}^{p}a(k)R(0,k) \quad \ldots \quad (4.22)$$

In (4.21), note that $s(n)$ must be known in the range $0\leq n \leq N-1$, just like in the autocorrelation method, but the equations to be solved in the two methods are different.

3.3 Computation of predictor coefficients

The solution of (4.12) or (4.18) for the predictor coefficients can be obtained using any of the familiar methods of solving a set of linear equations. Because the matrix of coefficient values in both sets of equations are symmetric and usually positive definite, the equations can be solved efficiently by the squared-root or Cholesky[19] decomposition method. The computation of the autocorrelation or covariance coefficients requires on the order of $pN$ operations, which can dominate the total computation time if $N >> p$, as is often the case.

4 Lattice analysis and synthesis

While direct-form LP analysis and synthesis of speech has been very popular in computer implementation, lattice synthesis has been the most popular method of LP syn-
thesis in commercial chips.[18,19] One of the major advantages of lattice analysis is that filter stability can be guaranteed with finite word length computations.[13,19]

In both the autocorrelation and covariance methods of LPC, the processing has two stages: the calculation of the correlation matrix, and the solution of the resulting set of linear equations.[13] However, in the lattice methods, the two stages have been effectively combined into a recursive procedure for determining the LPC parameters, where not just one but p backward linear predictors are used and the parameters are calculated one stage at a time.[19]

5 Linear prediction in speech recognition.

In general, the autocorrelation method requires the least computation, and the covariance method requires slightly more. However, because of windowing, the autocorrelation method may require a larger waveform segment length, so the two methods are generally considered of equivalent complexity.[13] The lattice methods require somewhat more computation. In addition, the autocorrelation and lattice methods guarantee stable synthesis filters, while this is not guaranteed for the covariance method.[13]

LPC analysis is very useful for speech coding. Only a few parameters need to be transmitted at a rate of 1200 to 2400 bits per second to produce speech. More natural speech may be obtained by coding error sequences in this format, at rates from 4800 to 16000 bits per second. LPC analysis has also found wide use in speech recognition systems and speech synthesis systems.[19]

6. Conclusions

This chapter was meant for a reader with a sound mathematical background. Once again the information was compressed so that it may not reduce its attractiveness for those readers with a lesser mathematical history. The basic concept of this chapter is that with the help of LPC a speech signal can be effectively coded into a binary format, and that this format can then be used to "reproduce" the original speech. The way in which LPC achieves this is quite an extensive process. If the reader wishes to learn more about how LPC can be implemented in a computer language, please feel free to examine the FORTRAN examples in [22].

In the next chapter we will examine the subject of speech synthesis.
CHAPTER 5: Speech Synthesis
This chapter follows logically on the previous chapter concerning LPC. In this chapter we start our discussion of speech synthesis with the history and basic principles underlying the subject. We will then further examine this subject by analyzing the three different methods that can be employed by speech synthesis. These are synthesis based on wavefrom coding, the analysis-synthesis method and synthesis by rule.

1. The principles of speech synthesis

We decided to include this chapter to give our readership as wide as possible exposure to the subject of Speech and Voice Recognition. A further reason for this inclusion is that it logically flows from the previous chapter on LPC.

Of the two problems of speech synthesis and speech recognition, synthesis is undoubtedly the easier problem. We still have no general theory of how the brain recognizes speech or speakers, and if we did have such a theory, there is no assurance that simply aping the process on the computer would be the best way to proceed, or that it would even be feasible. [9]

Speech synthesis is a process which artificially produces speech for various applications, diminishing the dependence on using a person's recorded voice. The speech synthesis methods enable a machine to pass on instructions or information to the user through "speaking".

As already mentioned, progress in LSI technology and LPC techniques in recent years have collectively helped to advance speech synthesis research. Moreover, information supply services are now available in a wider range of application fields. Speech synthesis research is closely related to research in deriving the basic unit of information carried in speech waves and in the speech production mechanism. [9, 23]

Voice response technology designed to convey messages via synthesized speech presents several advantages to that of information transmission: [5]

- Anybody can easily understand the message without training or intense concentration.
The message can be received even when the listener is involved in other activities, such as walking, handling an object or looking at something.

The conventional telephone network can be used to realize easy, remote access to information.

This form of messaging is essentially a paper-free communication form.

The last item also means that the lack of a hard copy of the messages makes them difficult to scan. Thus, synthesized speech is sometimes inappropriate for conveying a large amount of complicated information to many people.

History's first speech synthesizer, produced by von Kempelen[9, 11], is said to have been constructed in 1779, more than 200 years ago. This synthesizer, the first of its kind capable of producing both vowels and consonants, was intended to simulate the human articulatory organs.

Sounds originated through the vibration of reeds. They were modulated by resonance of the leather tube and radiated as a speech wave. Fricative sounds were produced through the "S" and "SH" whistles. This synthesizer is reported to have been able to produce words consisting of up to 19 consonants and 5 vowels. Early mechanically structured speech synthesizers, of course, could not generate high-quality synthesized speech, since it was difficult to continuously and rapidly change the vocal tract shape.

The first synthesizer incorporating an electric structure was made in 1922 by J.Q Stewart.[5] Two coupled resonant electric circuits were excited by a current interrupted at a rate analogous to the voice pitch. By carefully tuning the circuits, sustained vowels could be produced by this synthesizer.

The first synthesizers which actually succeeded in generating continuous speech was the vocoder, constructed by H. Dudley in 1939.[9] It produced continuous speech by controlling the fundamental period and band-pass filter characteristics, respectively, using a foot pedal and 10 finger keys. The vocoder[9, 23, 24], became a principle foundation block for recent speech synthesis research. The vocoder structure, based on the linear separable equivalent circuit model, is still used in present speech synthesizers.

Present speech synthesis methods can be divided into three categories:[4, 5, 9, 23]
- Synthesis based on waveform coding.

In which speech waves of recorded human voice is stored after waveform coding or immediately after recording has taken place to produce the desired messages.

- Synthesis based on the analysis-synthesis method

In which speech waves of recorded human voice are transformed into parameter sequences by the analysis-synthesis method and stored, with a speech synthesizer being driven by concatenated parameters to produce messages.

- Synthesis by rule.

In this method speech is produced based on phonetic and linguistic rules from letter sequences of phoneme symbols and prosodic features.

2. Synthesis based on waveform coding

As mentioned, synthesis based on waveform coding is the method in which words or phrases of human voice are stored and the desired sentence speech is synthesized by reading and connecting the appropriate units.[5,23] In this method, the quality of synthesized sentence speech is generally influenced by the quality of the continuity of acoustic features at the connections between units.

Acoustic features include the spectral envelope, amplitude, fundamental frequency and speaking rate.[5] If large units such as phrases or sentences are stored and used, the quality of the synthesized speech is better, although the variety of words or sentences which can be synthesized is restricted. On the other hand, when small units such as syllables or phonemes are used, a wide range of words and sentences can be synthesized, but the speech quality is largely degraded.

In practical systems typically available at present, words and phrases are stored, and then words are inserted or connected with phrases to produce a desired sentence of speech. Since the pitch pattern of each word changes according to its position in differing sentences, it is necessary to store variations of the same words with rising, flat and falling inflections. The inflection selected also depends on whether the sentence represents a question, statement or exclamation.
The major problems of simply concatenating words to produce sentences are as follows:

- A spoken sentence is very different from a sequence of words uttered in isolation. In a sentence, words are as short as half their duration when spoken in isolation, making concatenated speech seem painfully slow.
- The sentence stress pattern, rhythm and intonation are disruptively unnatural when words are simply concatenated.

3. Synthesis based on the Analysis-synthesis method

In synthesis based on this method, words or phrases of human speech are analyzed based upon the speech production model and stored as time sequences of feature parameters.[21] Parameter sequences of appropriate units are connected and supplied to a speech synthesizer to produce the desired spoken message. Since the units are stored by source and spectral envelope parameters, the amount of information is much less than with the previous method of storing by waveform, although the naturalness of the synthesized speech is slightly degraded. Additionally, this method is advantageous in that changing the speaking rate and smoothing the pitch and spectral change at connections can be performed by controlling the parameters. Channel vocoders and speech synthesizers based on LPC analysis methods are used for this purpose.[5,6,21]

4. Synthesis based on the speech production mechanism

Two methods are capable of producing speech by electrically simulating the speech production mechanism. One is the vocal tract analog method,[5] which simulates the acoustic wave propagation in the vocal tract. The other is the terminal analog method [5] which simulates the frequency spectrum structure, that is, the resonance and anti-resonance characteristics, which again simulates articulation as a result. Although in the early years these methods were realized by analog processing using analog computers.
or variable resonance circuits, digital processing has recently become popular owing to advances in digital circuits and computers and to their ease of control.

5. Synthesis by Rule

The most demanding synthesis process starts with the written text as input and produces acceptable speech as output. The written text can be a phonemic representation of the speech or, in the most ambitious systems, conventional written language. The latter process can be broken down into two sub-processes: [4, 5, 24, 26]

- Grapheme-to-phoneme translation
- Phoneme-to-speech translation.

Solutions to both these problems rely heavily on rules found empirically, and considerable effort has gone into developing systems in which proposed rules can be easily implemented and tried out. In general, the higher the required output quality is, the more complicated the synthesis rule will be. Since phoneme-to-speech translation is the more general problem, we will discuss it first.

5.1 Conversion of phonemes to speech

This process involves the following three problems: [5, 27, 28]

- Selecting proper allophones
- Joining adjacent phones naturally
- Providing proper pitch, stress and rhythm over the sentence

Conversion of phones to allophones is a matter of rules, the most of which can be applied at the word level, since most allophone choices are determined by context within the individual word. In some very simple systems it is left to the user to specify the allophones.
Suitable joints between phonemes are necessary to produce the fluency of human speech. Clearly we must supply formant transitions adjacent to consonants in order to provide articulatory information. The concept of articulatory locus is helpful here, since the locus is in most cases a function only of the point of articulation.

An additional application, however, is the fact that in natural speech true steady states are rare. The set of formant frequencies for each vowel should actually be regarded as a target toward which the actual formants move, but which they rarely reach, since we change targets at a rate comparable to the time scale in which the shift occurs. [4, 5, 9, 19, 23, 24]

Because we are used to hearing these continually shifting formants, artificial speech which does not include them is not only not acceptable but in many cases not intelligible. On paper, a pattern may be recognizable as "we were away", but to be understandable when played through a synthesizer, it must have smooth formant tracks. We therefore have an additional set of formant transitions to consider. There are not only those transitions which convey articulatory information about the adjacent consonants but also transitions which arise simply from the continuous motion of the vocal organs from target to target.

Another approach to the transition problem has been to break the speech stream at midpoint instead of between phones. Since each unit now contains two elements, the number of descriptions to be stored is significantly larger. On the other hand, the transitions between phones are included in the description and do not have to be estimated. Furthermore, the strongest coarticulation effects are those between adjacent phones, and these two-phone units can incorporate such coarticulations as a matter of course. [9]

5.2 Conversion of unrestricted text to phonemes (Grapheme-to-phoneme translation)

Most systems for translating text to phonemes use a combination of one or more dictionaries and a set of rules. A number of such sets of rules have been developed. Two of the more widely cited ones are the rules developed at the Naval Research Laboratories (NRL) called the NRL_Rules and the text-to-speech rules used in the MITalk system. [4, 5, 25]

Such rules operate primarily (1) on words or parts of words and (2) on individual letters or small groups of letters. These two categories correspond to the two ways in
which humans can treat alphabetical text. We identify full words or dissect the word into recognizable parts and when this fails, we sound the word out letter by letter. Rules of type 1 tend to be very numerous and to give high-quality output where they are applicable. Rules of type 2 can be reduced to a relatively small set, particularly when the exceptional cases have been already covered by type 1 rules.

The detailed description of these text-to-speech conversion methods will not be explained here. We urge the reader, that might want further information on this subject to consult our references.

6. Conclusion

In the next chapter we will conclude Section I of this document by discussing voice and speech recognition in more detail.
CHAPTER 6: Voice and Speech Recognition - Revisited
In this chapter we will continue the review of voice and speech recognition. We will commence discussions with a return to the background of speech recognition. We will then further examine such aspects as feature extraction and template matching.

1. Background

"Progress in speech recognition seems disappointingly slow by comparison to speech synthesis. The performance of commercial speech recognition machines sometimes ranks closer to that of well-trained dogs than of people."[4]

This statement may seem harsh, but if you would compare the progress made by speech synthesis in the last two decades with that of speech recognition one ultimately comes to this conclusion. There are however certain products or systems available on the market that would astonish the early researchers in speech recognition. Another aspect to take into consideration is that, although speech recognition and synthesis share a common knowledge domain, these two subjects should not really be compared. Further more if we consider the inherent difficulty of speech recognition, it may be unfair, after all, to compare a machine, which typically gets less than an hour of training, with a human, who gets a lifetime of training in spelling, grammar, logic, literature and other intellectual pursuits.

Because verbal communication is inextricably intertwined with human intellect, matching human speech recognition capabilities would require duplicating the capabilities of the human brain - a near impossibility given today's technology.[23]

As we have seen in the previous chapters, extracting verbal messages from the sound stream in which they are encoded is a formidable task in itself. Speech patterns vary from speaker to speaker because of such differences as physiology, sex, age and education. A word or syllable or other speech element may vary in loudness, pronunciation and stress, depending on its function in a sentence and the speaker's physical state.[4]

In the subject domain of speaker identification, these difficulties may turn out to be more of an asset. A security system, would not want to admit a person who may be under the influence of drugs or alcohol - overly relaxed, or a person that is under enormous stress.
Furthermore quite different sound patterns may represent the same word. While different words may sound very similar, especially if the speech has been distorted by background noises and other interference sources that plague electronic equipment. Compounding these problems is the fact that individual elements, such as words or phonemes, tend to lose their identity in continuous speech, making them difficult to isolate.

Because the general problem is so difficult, most research has concentrated on solving specific tasks, such as speaker-dependent recognition of isolated words or continuous speech with a small vocabulary. Other systems have combined these two applications very successfully. Fortunately, many practical applications exist for limited recognition. Isolated word recognition, for example, is adequate for logging freight destinations in a warehouse or for interpreting a client's bank account number over a telephone link to the bank's computer.

Isolated word recognition also side-steps the problem of segmenting continuous speech. For this reason, the development of such machines has received great emphasis and has led to the emergence of low-cost commercial systems.

Although word-recognition machines vary greatly in detail, they all use the same basic recognition process. A spoken word is converted into an electrical signal by a microphone; the signal is processed to extract a set of identifying features or the signal is converted by an algorithm, such as LPC. The result is then compared to a library representing the machine's vocabulary. A word is recognized if it matches one of the templates stored in the machine's memory. Unfortunately the search for a matching template may take hours if the machine's memory contains a great number of these templates.

2. Feature extraction.

The analysis stage of speech recognition consists of extracting identifying characteristics from the electrical analog of the speech signal generated by a microphone. When viewed in the time domain, speech signals look extremely complex and reveal no readily identifiable characteristics that distinguish one speech sound from another. Speech signals are much more readily analyzed in the frequency domain, where a signal's amplitude is examined as a function of frequency. This is because each speech element has a characteristic spectral signature, as evidenced in spectrograms (See chapter 2 and 3).
A trained observer can use spectrograms to identify and distinguish sounds. A spectrogram reveals the bar patterns of pulsating sounds from the vibrating vocal cords, the blurred traces made by fricative sounds, or clearly defined areas mapped out by the vocal resonance for vowel sounds. [5, 10]

Word-recognition systems employ various strategies of extracting spectral information from speech sounds. Such strategies minimize the computer processing needed to analyze speech sounds and the computer processing needed to store sounds and reference templates. They do, however, entail a trade-off. Simplifying the sounds by discarding irrelevant information can lower a machine's cost or increase its vocabulary size and response speed. Over-simplification can lead to a loss of recognition accuracy. The trick is to find a suitable compromise for a particular application. [4, 9, 23]

The simplest and cheapest way to convert a speech signal into the frequency domain is to count the number of times per second the signal changes algebraic sign, using a circuit known as a zero-crossing detector. This gives a rough measure of the dominant frequency, or pitch, of a speech signal. However, this method is not very discriminating, since it measures only the dominant signal frequency and does so only crudely. Most researchers believe that at least three formant frequencies must be identified to discriminate vowel sounds with reasonable certainty. [4, 5, 6, 13]

A more sophisticated version of the zero-crossing technique employs filters to split the speech spectrum into three frequency bands. Then the zero-crossings are counted separately in each band to obtain rough estimates of the first three formant frequencies. The method allows fairly reliable classification of vowel sounds and is useful for word recognition with small vocabularies. [13] Though more expensive than a single zero-crossing detector, the improved technique is still much less expensive than other frequency analysis techniques.

Increasing the number of filters, typically to sixteen or thirty-two, improves precision. In fact, this filter-bank technique eliminates the need for zero-crossing detection because any energy passed by the filters must be at or near the center frequencies of the filter passbands, which are known. [23]

Digital filtering entails converting speech signals to digital form before filtering. However, this often makes sense, since most speech recognition systems eventually convert a signal to digital form anyway to allow the information to be processed by a computer. (See chapter 3)
Digital processing can be used to eliminate the need for digital filtering altogether. Because of this, Fourier spectral analysis by computers has become very popular as the result of the development of a computationally efficient version of the Fourier transform, called the Fast Fourier Transform or FFT. The FFT makes it possible to compute rapidly and directly the amount of energy present in a speech signal at each frequency and its relative phase as a function of time - in other words its spectrum.

Another approach is the use of a technique called linear predictive coding or LPC. LPC predicts the amplitude of a speech waveform at a given moment from a weighted average of its amplitudes at a small number of previous instants. LPC is especially appropriate for speech recognition because it is based on a simple acoustic model of the human vocal tract that emphasizes the formant structure of speech, which is important for recognition.

3. Template matching

After analysis, speech signals must be compared to reference templates before they can be recognized. This matching operation presents a major stumbling block to successful speech recognition. One difficulty is in isolating the speech elements to be matched. A solution to this problem is to analyze words in units smaller than words. Even when the basic unit is a complete word, the word must be subdivided into time intervals so that the changing speech sounds can be tracked throughout the word.

Another difficulty with template matching is that speech elements are rarely the same length as the templates they are supposed to match. A word can be drawled or spoken crisply, depending on the speaker and the circumstances. This is true even when the speaker is the same person who recorded the original template. Varying the pronunciation rate can also change the relative distances between identifying features in a speech element.

Low-cost word-recognition systems handle this problem by adjusting all templates and words to be matched to a standard length, typically by omitting or interpolating time segments. Systems that require greater matching accuracy use techniques collectively known as dynamic programming to warp a template so that its features are aligned as accurately as possible with those of the word to be matched.

In speaker identification, these dynamic programming techniques must be modified, as the system is supplied with a template and it has to match it with all the tem-
plates in its memory. This process can take an extensive amount of time and we therefore need some sort of classification system that defines a supplied template to be enclosed in a given possible group of templates also called a template class, thereby reducing the search time. We will return to this problem in the chapters to follow.

4. Continuous-speech recognition

Although word-recognition systems are adequate for some applications, there are many potential applications that will require continuous-speech recognition.

Continuous-speech systems share some basic recognition process of extracting spectral signatures from speech signals and comparing them to templates. In addition, most use words as their basic recognition element.

The continuous-speech recognizer has two important problems which the isolated-word recognizer does not have:

- Segmentation

It is obviously impractical to do recognition of whole phrases because the number of possible phrases is too vast. Hence the process must break the input stream into constituent parts, as people do. This means the system must be able to recognize word boundaries. This is in general extremely difficult to do, since there is no consistent clue to their location. Energy minima are occasionally acceptable, but they usually need to be backed up by phonetic information.

- Phonetic variability.

Pronunciation in connected speech is sloppier than in isolated words, and coarticulation effects are more severe. Short words are especially effected. For example, "What do you want?" might become "Waddayahwant?"

Connected-word recognition can be implemented as an extension of word spotting or it can be done by a more general extension of the time-warping algorithm. Clearly, if we have a reliable technique for detecting the occurrence of a given word in continuous speech, then we should be able to make a continuous-speech recognizer simply by enlarging the set of words to be spotted until it comprises the entire vocabulary to be recognized. The principle drawbacks to this approach are normally limited to tasks like connected-digit recognition or other limited-vocabulary tasks. The nature of the digit-
recognition problem affords other convenient simplifications besides the vocabulary limit. Digit groups are normally spoken in bursts, so the length of signal to be analyzed is limited. In a burst, digits can be searched from left to right.

Another technique is that of Myers and Rabiner called Level building[9]. It describes a level-by-level time-warping algorithm which drastically reduces the number of possible paths to be evaluated over the digit group. A level is defined by duration of a possible template as matched to the incoming signal. A Time-warping program normally proceeds from frame to frame of the unknown word, for each frame it then goes through all relevant frames of the template. Hence we can envision the process as proceeding up a series of vertical strips. The use of levels means that the matching process does not have to consider every sample as a possible starting point for a new template, but only points near a level boundary.

An alternative strategy for continuous-speech recognition is to analyze the input stream into its constituent phonemes and then to identify the words in the utterance from their phonemes. (See Chapter 2)

Analysis tends to run from left to right, since the beginning of the first word at least is not in doubt. Systems generally pursue several alternative interpretations[23], frequently in parallel if the hardware can do so in a reasonable amount of time. These systems can usually be described in terms of two parts, the front end and the word matcher.[9]

The front end includes segmentation and preliminary phonetics or phonemic decisions. A segment at this level normally corresponds to a phoneme rather than a word or syllable. Features for segmentation are normally amplitude, voicing, zero-crossing rate, pitch and changes in phonetic or spectral type.[5,6,9,23] Of these, amplitude is the most important, but is not sufficient by itself, and practical segmenters use it in combination with other cues.

Recognition of segment boundaries is itself a pattern-recognition problem and like all such problems has two types of errors[4,5,9], false alarms and missed boundaries. Decision thresholds are frequently set to favour false alarms, since for many systems, recovering from errors caused by missing boundaries is more troublesome than recovery from illegitimate boundaries.

The word matcher takes the phonetic data from the front end and tries to make words out of them on the basis of stored phonetics rules, vocabulary and syntax rules. The syn-
tax rules specify the allowable sequences of phonemes and of words, the syntax rules thus must provide a model of language to be recognized.[9]

The most commonly used language model is the generative model[9]. This model views a language as a set of strings generated by a grammar. A grammar consists of a vocabulary, a set of syntactic types and a set of generating rules, called productions, which lead from a designated starting category to the final string.[23]

In summarizing, we can safely state that continuous-speech systems have a long way to go before they become suitable for widespread commercial use. Firstly vocabularies will have to be expanded at least tenfold[9], and secondly the recognition speeds will have to be expanded at least over a hundredfold[9,23]. Also machines must handle the perplexity of natural languages and become essentially speaker-independent. Furthermore, to penetrate the really large markets, cost must drop to below R250.00, which is far below the cost of mainframe-based research systems.

Progress in computer hardware is fairly certain. The parallel architectures and associative memories needed for speech recognition are already developed.[23] So, too, are faster and denser integrated circuits. The real breakthrough, however will be when speech recognition machines will be able to learn throughout their lifetimes - just as people do.

6. Conclusion

This then concludes section I of this document. Section I was meant to be an introduction into the world of voice recognition. We intended to give the reader a basic knowledge of the subject domain so that he or she may understand the underlying concepts governing our model. In Section II we will discuss the various layers of the Template Order Model as well as the concept behind the development of such a model.

In the next chapter we will start the definition of our voice classification model, called the Template Order Model or TOM.
SECTION II
CHAPTER 7: Voice Classification Systems
In this chapter we will introduce the reader to the problem domain of voice classification systems. We will examine the factors leading to the creation of such a system as well as the implementation criteria for the classification system.

In this chapter and those to follow we will introduce the reader to a voice classification model and how this model can be implemented in large scale voice-based computer systems.

1. What is a voice classification system?

A voice classification system is a computer-based system that examines a voice print and then decides where this voice print can be placed or where a copy can be found in a predefined group of voice prints.

This process contrasts steeply with that of voice template matching. In the latter, as explained in fig 7.1, the computer has a voice print that it must compare with one or more other prints in a voice library.

In the past the system would have had to compare every print in the library with the print in hand. If the library consisted of a great number of prints, say 1 million, it would have taken the recognition system a vast amount of time to find a match, that is, if a match existed within the million prints. Even if a match was found, it may not be the correct voice print.

It can therefore be said that such a large voice print library has three distinct drawbacks, these are:

- Search Time

Time is often said to be the most valuable commodity. Just consider the amount of time needed to find a matching voice print between a million other voice prints. A further problem is that most systems that employ voice technology, must be able to return with a fast response.
Hardware requirements

This factor is often associated with that of search time. A mistake some system developers make is that in order to achieve a faster response they should use a faster machine. One of the criteria, as we shall see later, is that the voice classification system should be able to execute on any type of machine, from PC to Mainframe to Super Computer.

Accuracy

The problem with searching through a million voice prints is that more than one print may satisfy the match requirements. The basic reason for this is statistical, as the probability for more than one match occurring grows bigger with the amount of prints examined.

We therefore will try to develop a classification system:

- that will reduce the search time in a large or small voice print library.
- that can be implemented on a very wide range of machines.
- that will prove to be reliable, fast and more accurate than current methods, especially in large voice libraries.

It must however be stressed that the voice classification system is only there to find the voice print or the group to which the voice print belongs to in the library. After this has been done the voice matcher takes over and compares the allocated prints with the unknown print, as described in fig 7.1 and fig 7.2.

The classification system is therefore an aid to the voice recognition system, that seeks out a voice print from among other voice prints.
Fig. 7.1 Voice Matching Systems
Fig 7.2 Voice Classification System
2. Model definition

In adherence to the design criteria we developed a voice Template Order Model or TOM. If we examine the basic elements of our model name we find that we firstly will use a Template - An image made to reproduce someone or something. We further envisage an Order - A class that is defined by the common attributes possessed by its members.

If we combine these attributes into a single idea we are confronted with a model that is used to group objects into fixed classes that are defined by the common attributes of its members.

TOM is based upon a few assumptions,

- The first is that a template based system is used.

This means that the voice system creates holding blocks or templates for a voiced utterance. These utterances are then compared or matched.

- The second is that TOM is not concerned with the actual meaning of the utterance.

To TOM these utterances are statistical manipulative data and nothing more.

- The third and final assumption is that TOM expects that the data inside the template will try to emulate a sine wave pattern.

That is, the data can be plotted on a X, Y -axis system where X equals time and Y equals "amplitude". A further assumption in this regard is that X falls within the range 0 ≤ X ≤ ∞ and that Y ranges between 0 ≤ Y ≤ 255.

Once these assumptions are fully understood, we can proceed with the layered definition of the model.

TOM consists of six distinct phases and each phase builds upon the output of the previous phase, see Fig. 7.3.

The first phase called Template Creation consists of the birth of the voice print. Here the analogue sound is converted to a digital format, such as described in the previous chapters.
Fig. 7.3 Voice Classification Model

Blaa...Blaa...Blaa......

- Template Creation
- Sound Peak Filtering
- Start Detection
- Garbage Removal
- Wave Reduction
- Zero-crossing Formatter

GROUP XZ

Phase

I
II
III
IV
V
VI
In the second phase called Peak elimination, the digital voice print is filtered through a set of logical filters. These filters were designed to remove any unexplained sound peaks in the template.

The third phase is called start detection. In this phase the actual start position of the wave is found, and any sound before this point is removed.

In the fourth phase any white space in the wave is removed. To achieve this the wave is left-shifted every time a value (X,0) is repeated. This phase is aptly called the garbage remover.

With the execution of the fifth phase the wave is divided into either a positive value or a negative value so that only three possible y-axis values are permitted, -1, 0 and 1. This phase is called wave reduction.

In the final phase the distances between zero-crossings, that is where y = 0, are measured and divided into two or more basic groups. This process is repeated several times and coded into a group descriptor.

These six phases can then be repeated several times on different templates inorder to achieve greater accuracy. This algorithm that "decides" in which group the template should be placed is called the Template Order Model Kernel. This kernel is not essential to the model but it has proved to be a performance enhancer.

Each of these phases will be described separately in the chapters to come, and an example of how it can be implemented in a high-level language will be given.
Fig. 7.4 FNB Personal Banking Service
3. The origins of TOM.

The idea to construct such a voice classification system originated with First National Bank. This institution is currently actively pursuing voice technology in solving commercial problems.

The first of these is a system whereby the bank's client can telephone the bank computer and request personal banking information verbally.

In order to verify the caller's identity the bank's computer request the caller to supply it with the client's account number. The account number is ascertained by the use of an Independent voice recognizer. The verbal number is converted into its digital representation.

This digital number is then used to access the bank's files. If the account number is found, the caller is asked to supply a verbal password. This password is matched against the template held by the computer. If a suitable match is found the caller is allowed access to his personal banking information, see Fig. 7.4.

A further interesting use of voice technology is the use of "verbal Links". The client can use a verbal password that is interpreted by the computer to execute a set of predefined instructions. The use of the word "Escom" could instruct the computer to pay the client's electricity account electronically.

The ideal situation would be if the client can call up the computer with the following dialogue:

*Computer*: "Good day, this is the Bank Electronic Banking Service. How may we help you?"

*Client*: "Hello. It's me."

*Computer*: "Please wait. We are verifying your voice identity. (2 Seconds later) Pleased to hear from you Mr. Brown. What can we do for you?"

*Client*: "What is my latest balance on my savings account?"
With the use of TOM the verbal code of "Hallo. It's me " should, theoretically, be able to verify most client's identity even if they all answered the same. In that way eliminating the need for an Independent voice recognizer to access the client's account number. The words "Hello it's me." can be used as an index to the bank's database.

In our minds, theoretically at least, it should be possible to use TOM to access any database verbally. The only problem is that the phrase "Hello. It's me. " must thereafter always be used to access the same database slot. This is a limitation placed by ourselves, in order to create a working model. The ideal situation would be for the person to repeat any phrase and not just the one the system was trained to accept. At the current stage of development we feel that this is not entirely possible given this moment in time.

4. Conclusion

The use of verbal commands to access such a database holds a number of advantages to normal database accessing methods, such as security and user friendliness.

The main disadvantages would be speed. It will take anything from 2 seconds to 5 minutes to find a database slot using TOM, depending on the size of the database and the number of TOM groups.

TOM has its definite advantages, but it must be realized that the only real test for a model such as this is the practical implementation.

Although we found it to be a possible working solution it is still in its infancy.

In the next chapter we will discuss the first and second phases of our model, named Template creation and Sound peak filtering respectively.
CHAPTER 8: Template Creation and Sound Peak Filters
In this chapter we will discuss the first two phases in our Template Order Model (TOM), these are the template creation and the sound filter phases. Each phase will be highlighted with a high-level language code example. The source code for these examples can be found in Appendix A.

1. Introduction

In phase one of our model the analogue voice is converted into a computer understandable digital format.

This process is achieved using the basic voice processing techniques described in the earlier chapters of this document.

In our case these techniques were already implemented on a locally designed and developed PC extension board.

The card is fitted with telephone and microphone extension plugs. We however only made use of a microphone in our experiments.

The card samples the input from the microphone or telephone at a predetermined sampling rate. In our case it was set at 5 times per second. The samples received are sent through a set of sound frequency filters. If the sample values are within a defined sample frequency range the sound is digitized. In our research we used a 120Khz sample cut-off value.

The samples are collected in the PC’s memory until the end of the utterance is reached or the maximum sample value is exceeded (20 000 samples). It is then LPC coded and stays in the PC’s memory for later retrieval.

The LPC coded template consists of 10 LPC samples per actual speech sample. We used only the 1st LPC sample, as it tended to be the most accurate representation of the speech wave.
The software algorithm to code the speech wave into its LPC representation was supplied to us by the developers of the A\D extension card. We therefore assume that the code is correct as there was no way for us to determine otherwise.

2. Phase 1: Template Creation

The beginning of this phase is reached as soon as the speech wave is LPC coded.

TOM then retrieves the LPC template from memory and formats the LPC data into a more user-friendly format for use by the phases to follow. (Example 8.1 - 8.2 in Appendix A page A-1)

Phase One is now completed.
Fig. 8.1 TOM Template Creation (Phase I)
In phase two the unexpected sound peaks are filtered through a set of seven filters.

We however first had to define an unexpected sound peak, so that we might not mistakenly consider a real wave peak for such a phenomena.

A sound peak, see Fig 8.2 - Fig. 8.3, usually occurs for 1 to 4 template time intervals, it also disrupts the sine wave, so that where we expect a positive wave a short, explosive, negative wave appears. The same is also true for the reverse situation, where we expect a negative wave, it suddenly turns positive for a short duration and then returns to negative.

The easiest way to identify a sound peak is to examine its duration. It is seldom larger than 5 samples, and it almost always looks out of place in the sine wave.

The implications of not filtering out these peaks could lead to impaired results, as they are totally random in nature. It might be there the first time when the system is trained, and never appear again in the test templates to follow.

These sound peaks can not actually be seen as noise, although it is highly likely that they are caused by noise or unvoiced plosive sounds. We therefore tried not to refer to these unexpected template samples as noise, but rather as sound peaks, a term that might best describe them.

We think that, at this time, we might possibly explain the concept of a template and a wave, as we visualize them. In our minds a sound wave is a set of digitized voice sounds. These samples form a sine wave pattern if they are plotted on a graph. Therefore the term sound wave. A template however is the collection of these samples into a container, which could be a computer memory location or a computer disk file.

We can therefore state that a template holds a sound wave, where a sound wave consists of a group of digitized sound samples.
Fig 8.2 Sound Peak when a positive wave is expected

Fig 8.3 Sound Peak when a negative wave is expected
3.1 Peak filter design

Each filter is developed for a specific scenario, and they are independently executed in a sequential format. That is to say that filter 2 follows filter 1 and filter 4 follows filter 3 and so forth.

As each filter only examines the template for its case scenario, it could be possible for one filter to create a scenario for another. The filter alters the y-value to zero if its case scenario is discovered. This action can lead to the effect that filter 4 created a scenario for filter 2, which filter 2 did not find at the time it was executed.

We therefore execute the filter bank algorithm more than once. So that the second filter operation would execute on the output of the previous filter operation. We would take template X and receive template Y after the first complete filter operation. We would then take template Y and receive template Z at the end of the second complete filter operation.

Please note that the filter only examines the samples’ y-value. The x-value’s only use is to determine time, that is to denote the previous sample, the current sample and the future sample. Thus each filter will examine a 3 sample window and then move the window one sample into the future. This windowing concept is repeated until the last 3 samples in the template are reached.

3.1.1 Filter One

Filter 1 was designed to filter out the easiest case scenario. That is when the peak consists of only one sample, namely the current sample as explained in Fig 8.4.

Filter 1 becomes active in the following scenario:

<table>
<thead>
<tr>
<th>Previous = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current &lt;&gt; 0</td>
</tr>
<tr>
<td>Future = 0</td>
</tr>
</tbody>
</table>

Page 8-6
Fig 8.4 Filter 1. An unexpected flow of the sine wave.
3.1.2 Filter Two

This filter is an extension of filter 1. It makes provision for a peak that is characterized by a sudden polarity shift. In the example in Fig. 8.5 the polarity shift is from negative to positive and back to negative.

It is executed if the following scenario exists:

<table>
<thead>
<tr>
<th>Previous</th>
<th>Current</th>
<th>Future</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 0</td>
<td>&gt; 0</td>
<td>&lt; 0</td>
</tr>
</tbody>
</table>

3.1.3 Filter Three

Filter three is the inverse to filter 2. It makes provision for a sudden polarity shift in the wave pattern. This is characterized by a shift from positive to negative and back to positive. This scenario is illustrated in Fig. 8.6 and given by the following table:

<table>
<thead>
<tr>
<th>Previous</th>
<th>Current</th>
<th>Future</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; 0</td>
<td>&lt; 0</td>
<td>&gt; 0</td>
</tr>
</tbody>
</table>
Fig. 8.5 Filter Two. Unexpected polarity change from negative to positive.

Fig. 8.6 Filter Three. The unexpected polarity change from positive to negative.
3.1.4 Filter Four

Filter Four was designed to remove peaks that start with a positive value, switches polarity and then returns to 0. It only differs with filter 3 in that the Future value equals 0. This filter is explained in Fig. 8.7. It is executed if the following scenario exists:

<table>
<thead>
<tr>
<th>Previous  &gt; 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current  &lt; 0</td>
</tr>
<tr>
<td>Future = 0</td>
</tr>
</tbody>
</table>

3.1.5 Filter Five

Filter Five is the inverse to Filter Four. The samples start with a negative value, it then switches polarity to positive and then returns to 0. It differs from filter 2 only in the respect that with this scenario the Future value equals 0. It is executed if the following scenario exists:

<table>
<thead>
<tr>
<th>Previous  &lt; 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current  &gt; 0</td>
</tr>
<tr>
<td>Future = 0</td>
</tr>
</tbody>
</table>

This filter is illustrated in Fig. 8.8
Fig. 8.7 Filter Four. The unexpected polarity change from positive to negative to zero.
Fig. 8.8 Filter Five. Unexpected polarity change from negative to positive to zero.
3.1.6 Filter 6

In Filter 6 we concentrate on the value of the Future sample. It must be larger than 5% of the Current Value. If this filter is executed, it might create a condition for other filters, as it resets the value of Current. See Fig. 8.9 for an illustration. Filter 6 is executed if the following scenario exists:

<table>
<thead>
<tr>
<th>Previous $&lt;&gt; 0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current $&lt;&gt; 0$</td>
</tr>
<tr>
<td>$</td>
</tr>
</tbody>
</table>
Fig. 8.9 Filter Six, Previous is less than zero, Current is greater than zero and the difference between Current and Future is equal or greater than 95 % of Current's value.
3.1.7 Filter 7

Filter 7 compares the three sample values, Previous, Current and Future. It is executed if the value of Current is extensively different to that of its neighbors.

\[
\begin{array}{|c|}
\hline
|\text{Current}|-|\text{Previous}| \geq 95\% \text{ of } |\text{Current}| \\
\hline
\text{Current} < > 0 \\
\hline
|\text{Current}|-|\text{Future}| \geq 95\% \text{ of } |\text{Current}| \\
\hline
\end{array}
\]

This filter is illustrated by Fig. 8.10

- Continued on the next page -
Fig. 8.10 Filter Seven. Previous and Future is compared with Current. If Current differs extensively then this Filter is executed.
3.2 Filter banks

The above TOM filters were designed to meet the problems we encountered. It may be possible that we did not discover other cases of sound peaks.

We therefore admit that there might be a need for more filters, or for more sophisticated filters. We believe that these seven filters are adequate in removing the unexpected sound peaks from any template. An example of how these filters can be implemented is given in example 8.3 in Appendix A page A-6.

The filter banks are essential in achieving the end results. A template that did not go through a filter operation may behave in an uncontrollable fashion.

4. Conclusions

The use of filters cannot be stressed enough. If the template is not filtered for sound peaks, it will disrupt the model's test results. Unfiltered templates behave in a random fashion and in our case it degraded the model's results.

This then concludes our discussion of the TOM peak filters. The next chapter will explain phase three and four of our model, that of the concept of start- and end-point detection and garbage removal respectively.
CHAPTER 9: Start Detection and Garbage Removal
In this chapter we will discuss the third and fourth phases of our Template Order Model (TOM). These phases are start detection and garbage removal respectively. Each phase will be explained with the help of a code example. The source code for these examples can be found in Appendix A.

1. Start detection - The basics

A fundamental problem in voice recognition systems is that the detailed timing of an utterance at recognition time will generally not be exactly the same as it was during the training phase. With the training phase we mean the stage when the system was given a new template for the very first time.

Not only will the two utterances have different durations, but the spacing between phonetic events will not be consistent. This means that time-dependent features may fail to match because the unknown template and the reference template are out of time registration, that is, they are actually both originating from the same speaker, but they cannot be matched because each template’s time window differs.

In such a case, the correct pattern may seem as far different from the unknown as any incorrect template. In fact, even between two training utterances the detailed timing may not agree, particularly if the various utterances have been recorded at different times.

In many cases, as with our model, the accuracy of alignment depends on the accuracy of identifying the start- and end-points of the template.

In the laboratory, recognizing start- and end-points is usually relatively easy, because speech data is collected under controlled conditions and in some cases may be hand-edited to identify these points. Furthermore, speakers recording data for laboratory work tend to speak with care. In the field, however, speech is not clean, the conditions are not controlled, and speakers may have no motivation to help the recognizer. [23]
The principal feature for identifying start and end-points is energy. The normal technique is to compare the energy with some threshold value and identify the start of the utterance as the point at which energy first exceeds the threshold and the end as the point at which energy drops below the threshold.

Such a test requires a number of safeguards. First, the threshold itself may have to be normalized to general intensity; if the speech signal is weak, a lower threshold may have to be used. Alternatively, the intensity itself may be normalized.

The process must also guard against false alarms. Random noise may cause a momentary crossing of the threshold in the absence of speech, and this must not trigger the recognizer. Hence there is usually a length requirement. An example may be if the intensity exceeds the threshold for more than five sample time frames, then the first frame will be retroactively identified as the start of the utterance.

Identifying the end of an utterance uses similar techniques. Final unvoiced plosives are an additional source of problems, because if the release of the plosive is delayed too long the recognizer may identify the beginning of the closure as the end of the word and thus miss the release.

Hence most techniques require some frames of silence to elapse before the utterance is declared dead.

In the following section we will discuss the way in which TOM handles the problem of Start Point Detection.
Fig. 9.1 The sample window starts at the beginning of the template.

Fig. 9.2 The sample window starts at the first non-zero sample value.
2. TOM Phase Three: Start Point Detection

Our model, TOM, is only interested in the start detection of the template, as we only use the first 8 zero-crossings and most templates are usually much longer than that.

The method we use to find the start of the template, is actually a less complicated algorithm than those explained in section one of this chapter.

TOM also makes use of energy or more appropriately amplitude. We first need to execute phase two of our model, that is sound peak detection, in case the first few samples of the unknown reference template contains any unexpected sound peaks.

After we have filtered the template through our set of six peak filters we are relatively sure that the first few samples of the template do not contain any sound peaks.

The process that follows examines the template’s first five samples, called the sample window. We scan through this sample window until we find a sample value that does not equal zero. Once we have found a non-zero sample we move the sample window to consist of the first non-zero sample, plus one sample, to sample five. We therefore reduced the size of this sample window.

We then examine this sample window for any zero-crossings, that is any sample where the y-value crosses the zero mark. If any were found we reset all the samples in this sample window to zero. If none were found we admit that this sample window is in fact part of our template and the start is considered to be the first non-zero sample.

Now we can already hear remarks from our readership that we may be eliminating an actual start point and shifting it fifteen samples to the right. See also Fig. 9.1 to Fig. 9.3.

This case scenario is possible, we will admit, but once again highly unlikely. The templates we tested almost never started within the first fifteen samples, these samples are usually taken up by noise or unvoiced utterances. And even if we had mistakenly eliminated part of our wave, it should not have had any great impact on the rest of the model.

Example 9.1 explains the way in which this Start Point Detection algorithm can be implemented in a computer language. It can be found in Appendix A page A-10.
Fig 9.3 TOM Phase 3: Start Detection. The new start point is found and the previous sample values are reset to zero.
3. Phase Four: Garbage Removal.

At the end of phase three, start detection, the start of our template is determined and all previous y-values are set to zero. A large amount of white space exists. White noise is defined as values with an amplitude value of zero.

TOM has no use for sample values where the value \((X,0)\) repeats itself.

![Diagram](image)

Fig 9.4 Tom Phase 4: Garbage Removal. At the beginning of this phase the whole template is left shifted until the initial white space is removed. Other forms of white space will still exist within the template as this example shows.
To combat this feature TOM will first left shift the whole template, starting at the new start point, until the y-value of our start point is not equal to zero. This operation then effectively removes any white space BEFORE the start point. We still however have a problem with the white space within the rest of the template, as explained by Fig. 9.4.

The process used to remove these duplicate sample values is called garbage removal. The now left-shifted template is examined for any duplicate samples with the value (X,0). If such a sample is found the template right from this sample is left-shifted for one sample, thereby eliminating the duplicate value (X,0). This process is repeated until the end of the template is reached and in the process the length of the template is shortened.

We therefore actually compressed the template to its real size or its normalized size. Theoretically this length should come close to or equal to the actual length of any other template created by the same person uttering exactly the same word.

At the end of this phase we have a template without any noise or duplicate zero-crossings as can be seen in Fig. 9.5. In the next chapter we will reduce the wave to three possible y-values and we will measure the lengths between zero-crossings.

Example 9.2 explains how the Garbage Removal phase should be implemented in a high-level computer language. This can be found in Appendix A page A-12.
Fig 9.5 TOM Phase Four: Garbage Removal. At the end of this phase any remaining white space is removed from the template.
4. Conclusions

The above two phases are extremely important in our model. They have the same underlying problem, what to do with nothing, or in this case the duplicate zero-crossings. Especially if we consider that the ultimate goal of this model is to examine the zero-crossings in a sound wave. If these duplicates were allowed to exist the result of our model will be totally different: the zero-crossing counter will mistakenly include duplicate values - thereby admitting fictitious zero-crossings.

In the following chapters we will explain how these zero-crossings are measured and interpreted to form the final output of our model.
CHAPTER 10: Zero-crossings
In this chapter we will explain how our Template Order Model (TOM) finds a group descriptor from an unknown, but TOM-formatted template. We shall see how the zero-crossings are measured and how they are then formatted into a group descriptor.

1. The TOM Zero-crossing method.

We should first explain to the reader the concept of a zero-crossing.

If we consider that a voice template consists of a set of (X, Y) values, where the y-value donates the voice "amplitude" and the x-values shows the time factor. Then the value (X, Y) forms a voice sample at a given interval in time.

If we then try to connect the preceding sample's y-value and the current sample's y-value with a straight line and we then repeat this process for the duration of the template, we will be left with something emulating the spectral envelope.

If we follow this envelope from the start to the end of the template, chances are that the envelope would exhibit any of three possible characteristics:

- It may change from a positive value to a negative value, by crossing the y = 0 mark.
- It may change from a negative value to a positive value by crossing the y = 0 mark again or
- It may stay constant on the zero mark for a few samples and then enter into a positive or negative phase.

All three the above cases will comply to being a zero-crossing, as they all move at one time or another through the y = 0 mark.

We could thus define a zero-crossing as any sample on a voice template that has a y-value that equals zero or any sample that has a y-value that differs in sign from the preceding or forthcoming sample y-value.

This definition is explained by the diagram in fig. 10.1
Fig. 10.1 Zero-crossings. The arrows show where zero-crossings can be found in this template.
Zero-crossings in the voice literature has a somewhat different meaning. In order to eliminate confusion we will explain the difference between our zero-crossing method and those described in the literature.

The zero-crossing analysis method, described in other literature, is used in the digital processing of the voice print in the time or frequency domain.[4]

The number of zero-crossings are usually counted in order to detect the formant frequencies or it can be used as a cheap way to convert a speech signal into the frequency domain.[29,30]

We named our own procedure the zero-crossing method, as this name explains the method most extensively. This terminology will be used throughout this dissertation.

2. Measuring zero-crossings

The previous layers of the TOM model were extremely essential for this layer to be executed correctly. Special note should be made of the noise reduction and garbage removal layers. Without them the zero-crossing detector (ZCD), the part of our model that scans the template for zero-crossings, would be prone to mistakes.

If we did not have the noise reduction layer, the ZCD might accept noise as part of the sine wave and it might even count a zero-crossing where noise has created one.

Without the garbage removal layer, zero-crossings will once again be counted where white space existed, thereby giving a false presentation of the actual wave.

It is therefore imperative that the different layers of TOM be executed, and executed in the correct order.

The zero-crossings are measured using a simple algorithm. It consists of two parts. The first is the zero-crossings detector (ZCD), as previously mentioned, and the second element is the sample distance measurer (SDM), the element in our model that measures the distance between two zero-crossings.
The zero-crossings algorithm starts executing at the start of the formatted template, as received by this layer from the garbage removal layer. It will first reset a counter called the sample distance register (SDR) to zero. This counter is used to hold the position of the previous zero-crossing. The algorithm further empties a container called the sample distance container. This container holds the distances between the zero-crossings detected by the ZCD.

The Zero-crossings Detector (ZCD) is then invoked, as mentioned earlier at position $$(X_1,0)$$. 

2.1 The Zero-crossings Detector (ZCD)

The zero-crossings detector scans the template sample by sample for the following conditions:

- The previous and current samples' y-values differ in sign

or

- The current y-value is equal to zero.

Example 10.1, in Appendix A, explains how the ZCD may be implemented.

If any of these conditions are found the sample distance measurer is activated, the actual way in which the template is scanned is by advancing a sample window two samples at a time. The two samples are compared for the above conditions in either previous and current mode or current and future mode. It actually does not matter in which way the two samples are compared as long as the method stays constant for the duration of the template.

2.2 The Sample Distance Measurer (SDM)

The sample distance measurer takes the current x-value and subtracts from it the value stored in the sample distance register (SDR). The result is then stored in the sample dis-
Fig. 10.2 The Sample Distance Measurer. It's main function is to determine the lengths between zero-crossings.
tance container. After completion of this task the sample distance register (SDR) is set equal to the current x-value.

If the end of the template is reached the sample distance container is handed over to the zero-crossing formatter, otherwise processing is turned over to the zero-crossings detector (ZCD). See Fig. 10.2

Fig. 10.3 The Zero-crossings Formatter. The first example is the primary input. The binary format method converts this into a binary string and then into a ASCII value. The output of this formatter then forms the primary group descriptor.
Once the end of the template is reached the ZCF is invoked. Its function is to examine the sample distance container and to deduce a TOM group descriptor from this information.

The ZCF examines the sample distance container in a sequential format. If this element has a value larger than 18 it is classified as a DOT (.), otherwise it is classified as a DASH (-). This classification leads to a set of characters resembling the Morse-code communication system. An example of such a group can be seen in Fig. 10.3. The Morse-code classification system[12] is described in table 10.1.
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Alpha</td>
<td>-</td>
<td>November</td>
</tr>
<tr>
<td>Bravo</td>
<td>-...</td>
<td>Oscar</td>
</tr>
<tr>
<td>Charlie</td>
<td>-...</td>
<td>Papa</td>
</tr>
<tr>
<td>Delta</td>
<td>-...</td>
<td>Quebec</td>
</tr>
<tr>
<td>Echo</td>
<td>.</td>
<td>Romeo</td>
</tr>
<tr>
<td>Foxtrot</td>
<td>...</td>
<td>Sierra</td>
</tr>
<tr>
<td>Golf</td>
<td>-...</td>
<td>Tango</td>
</tr>
<tr>
<td>Hotel</td>
<td>......</td>
<td>Uniform</td>
</tr>
<tr>
<td>India</td>
<td>..</td>
<td>Victor</td>
</tr>
<tr>
<td>Juliet</td>
<td>---...</td>
<td>Whiskey</td>
</tr>
<tr>
<td>Kilo</td>
<td>-.-</td>
<td>X-Ray</td>
</tr>
<tr>
<td>Lima</td>
<td>---...</td>
<td>Yankee</td>
</tr>
<tr>
<td>Mike</td>
<td>-</td>
<td>Zulu</td>
</tr>
</tbody>
</table>

Table 10.1 The Morse-Code Communication System
The value of 18 was determined by us as the most effective cut-off point. If a value greater than 18 is used the DOTS become excessive and the same is true for the DASHES if the value is less than 18.

These DOTS and DASHES can effectively be binary coded, by assigning a 0 to a DOT and a 1 to a DASH.

The way the rest of the formatter operates is entirely dependent on the users of the model. In our minds the Template Order Model ends here, but we suggest the following formatting method:

3.1 The Binary Coding Method

This method suggests that the first eight zero-crossings (0,1) are used to form an ASCII number. This number, from 0 to 255, can then be used as the primary group descriptor.

If only one of these codes is the wrong way around, meaning it should be DOT when it is a DASH or it should be a DASH when it’s really a DOT, the group descriptor will assign the wrong group. We therefore suggest that a set of 8 secondary group descriptors be defined by inverting each element in the primary group code, turning an element into a zero instead of a one and the other way round.

This method will then supply the matcher with a primary group. If the template is not found in that group, the matcher will search in the eight secondary groups. If the template is once again not found, then it is probably not contained in the library.

The binary coding method is only successful in a small library, where the number of templates per group is still relatively small.

Variations on this method is unlimited. The number of codes can be expanded from 8 to 32. We would not suggest a code with more than 32 elements, as the number of groups, using 32 codes, are 65536 and the model may become ineffective. Example 10.2 explains how the Binary Coding Method could be implemented in a high-level computer language. It can be found in Appendix A on page A-15.
3.2 A practical example

To illustrate the model, we included a practical example. This example takes the format of a slide presentation, illustrated by figures 10.4 - 10.7.

Each illustration shows the end result of the specific TOM phases.

The example consists of a template created by a male voice, uttering the phrase "Father".

Fig. 10.4 forms the first slide in our presentation. It consists of the LPC coded utterance as supplied to us by the hardware. In fig. 10.5 the sound filters were executed twice, and this was the result we received. Fig. 10.6 contains the output of the Start detection and Garbage removal phases.

The final slide in Fig. 10.7 shows the template after the Wave reduction phase. The output from this phase is sent to the Zero-crossings formatter. The Sample Distance Register (SDR) was set to 8 samples, that effectively means that if the distance between two zero-crossings was less than 8 samples, it was seen as a DOT(0). Or if the distance was larger than 8 samples it was seen as a DASH(1). The primary group for this example was calculated at 76 (DOT,DASH,DOT,DOT,DASH,DASH,0,0). As there were only 6 zero-crossings the primary group was padded with two zeros. The eight secondary groups were calculated by inverting each bit of the primary descriptor (76). An example of this might be 204 (DASH,DASH,DOT,DOT,DASH,DASH,0,0). The final output of this example is:

<table>
<thead>
<tr>
<th>Group Type</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary Group Descriptor</td>
<td>76</td>
</tr>
<tr>
<td>Secondary Group Descriptors</td>
<td>204,12,108,92,68,72,78,77</td>
</tr>
</tbody>
</table>
Fig. 10.4 The phrase "Father" after the Template creation phase
Fig. 10.5 The phrase "Father" after the Sound Peak Filter phase
Fig. 10.6 The phrase "Father" after the
Start Point Detection and Garbage Removal phases
Fig. 10.7 The phrase "Father" after the Wave Reduction phase. This forms the input to the Zero-crossings phase.
We conclude this chapter by divulging our test results found using TOM with the binary coding method.

We created a library that is divided into 256 groups, starting at 0 and ending at 255.

We selected 20 words in the English language. We then asked our test subjects, one male and one female, to create 40 templates each by repeating each word twice.

We then created a library of these 20 words by training the system using the Template Order Model. The library was set up to hold 9 identical copies of each word, by placing one copy into the primary group and one copy in each of the 8 secondary groups.

We then asked our test subjects to test the system by repeating a word out of the 20 selected words and afterwards to express a word that we knew was not contained in the library.

In most of the cases the word was found in the library using the primary group only. When the secondary groups were employed in the search a large percentage of the templates were found. The system got stuck in only a small percentage of the cases, when it could not locate an existing template in the library using the primary and secondary groups. In such cases the process was repeated in order to get a "fresh" template. This repetition was only done once, in order to be fair to the real user of the system, who will not know if the word is contained in the library or not.

In some cases we found that TOM allocated the same primary group to both the male and female subject's templates, when they expressed the same word. This we do not see as a problem as the matcher will eliminate any discrepancies, by finding the right match between different voices.

The use of secondary groups expands the library eight-fold, which is a waste of storage space. We would suggest that a random secondary group or groups be selected and that only in them copies of the template be placed, thereby saving storage space but reducing the system's efficiency.
If the system only used the primary group, the efficiency is lower, but the storage space is equal to the number of templates contained in the library. The user does however run the risk of not finding a template in the library, when it is actually contained in it.

5. Conclusions.

TOM is a workable solution to a very difficult problem. As with any workable model, it can with time and innovation be improved.

We found our test results to be adequate, but we do acknowledge that these were done under laboratory conditions. The system’s results will most probably be reduced in practical surroundings, where such aspects as user willingness, acoustics and time will be factors to be reckoned with.

We designed TOM in a modular fashion, we therefore expected the user’s of the system to make small changes on each level, to fit their own needs.

We will be the first to admit that TOM is an idea in the form of a theoretical model, and should be treated as such.

In the next chapter we will explain the Template Order Model Kernel.
CHAPTER 11: The TOM Kernel
In this chapter we discuss a performance enhancer for the Binary Coding Method. This algorithm is named simply the Kernel.

1. Introduction

The binary coding method described in the previous chapter formats the zero-crossings into a primary descriptor with eight secondary descriptors.

We found in our experiments that the primary group descriptor may not always be the same for different templates of the same utterance. In most cases however the primary descriptor for one template can be found in the secondary groups of another.

In order for us to design a performance enhancer, we obviously needed more than one set of input data, we therefore had to suggest that the user create three templates. These three templates are examined using the Template Order Model and the group descriptors of each template is compared by our algorithm.

The kernel then decides in which group this utterance will be placed.

2. The Kernel.

The kernel can therefore be best described as a semi-intelligent device, that determines the actual group into which the template will be placed. In order for it to achieve this goal, it makes use of boolean logic, that is, it derives its "decision" by following a primitive decision tree.

The Kernel algorithm can operate under three different modes, depending on the three primary group descriptors. These modes can be described by using the following scenario:

- All three the primary group descriptors have the same value.
- Two out of the three primary descriptors have the same value.
- None of the primary descriptors agree.

Thus, depending on the situation, the kernel will execute a decision in a different manner and it will return a different performance rating. The performance rating returned
Fig 11.1 The Kernel and its decision tree.
by the kernel can be seen as a score-card for TOM, with a maximum of 100 points available.

2.1 All three primary groups agree.

If all three the primary groups agree, then it is obvious that all three sets of secondary descriptors will also agree. The kernel has no other choice but to select the value of this primary descriptor as the final group. In this case the performance enhancer will return with a 100% success rate. This case scenario is rare, but it has been known to appear.

Table 11.1 Shows an example where all the primary groups agree.

<table>
<thead>
<tr>
<th>Template No.</th>
<th>Primary Group</th>
<th>Secondary Groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>128</td>
<td>(0,1,2,4,8,16,32,64)</td>
</tr>
<tr>
<td>2</td>
<td>128</td>
<td>(0,1,2,4,8,16,32,64)</td>
</tr>
<tr>
<td>3</td>
<td>128</td>
<td>(0,1,2,4,8,16,32,64)</td>
</tr>
</tbody>
</table>
Primary Groups Agree.

Performance Rating 100 %

Fig. 11.2 All three the primary groups have the same value. Performance rating 100 %
2.2 Two out of Three primary descriptors agree.

In this case, two of the three primary descriptors agree, so also will their secondary groups. In this case the kernel has achieved a basic success rate of 66.6\%.(Three groups, two agree.)

For readability we shall refer to the three primary descriptors as $P_1$, $P_2$ and $P_3$. Where $P_1 = P_2$ and $P_3 \neq P_1$. (And so by default $P_3 \neq P_2$)

The kernel searches for the value of $P_3$ in the set of secondary groups of $P_1$ and $P_2$. Three possible outcomes of this search operation is possible:

1 - The value of $P_3$ could not be located in either the secondary groups of $P_1$ or $P_2$.

In this case the kernel returns the value of $P_1$ with performance rating of 66.6\%\(\frac{\frac{2}{3} \times 100}{1}\%\)

2 - The value of $P_3$ was found in one of the secondary groups.

Here the kernel will return the value of $P_1$ with a basic performance rating plus an additional performance rating of 16.5 \%. \(\frac{33.3}{2}\) giving a performance total of 83.1 \%.

3 - The value of $P_3$ was found in both the secondary groups of $P_1$ and $P_2$.

In this case the kernel will return the value of $P_1$. The performance rating will consist of the base rating plus 33.3\% minus a 10 \% penalty giving a performance total of 90 \%.
Two Primary Groups Agree
\( (P1 = P2, P3 <> P1) \)

P3 Not Found in S1 or S2

Performance 66.6%

P3 Found in S1 and S2

Performance 90%

P3 Found in either S1 or S2

Performance 82%

Fig 11.3 Two out of the three primary groups agree.
Table 11.2 Shows an example where two out of the three primary groups have the same value.

<table>
<thead>
<tr>
<th>Template Name</th>
<th>Primary Group</th>
<th>Secondary Groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>128</td>
<td>(0,1,2,4,8,16,32,64)</td>
</tr>
<tr>
<td>$P_2$</td>
<td>129</td>
<td>(1,128,131,133,137,145,169,193)</td>
</tr>
<tr>
<td>$P_3$</td>
<td>128</td>
<td>(0,1,2,4,8,16,32,64)</td>
</tr>
</tbody>
</table>

2.3 No agreement between the three primary descriptors.

In this case the kernel searches through the three sets of secondary descriptors twice. In the first search the kernel examines the secondary groups for any of the primary group descriptors. If any primary group descriptors are found in a secondary group it becomes the final selection.

At the end of this search the kernel has two options:

1 - A primary descriptor was found in a secondary group.

The kernel then selects this value as the primary descriptor, and returns a performance result of 33.3% + 16.5% giving a performance total of 40%.

2 - No primary descriptor can be found in the set of three secondary groups.

The kernel then commences the second search as described hereafter.

If however no primary descriptor is found then each secondary descriptor is allocated his own counter. As each secondary descriptor is repeated in this set of secondary group descriptors the counter for that specific descriptor is incremented.
1- Not one secondary descriptor was repeated in the three sets.

In such a case the kernel returns a negative primary group descriptor with a 0% performance rating.

2 - The same secondary descriptor was found in more than one set.

The kernel will return the value of this secondary descriptor as the primary group of this template. The performance rating will depend on the value of the secondary descriptor's counter. An example would be if the counter equals 5. As there are 24 secondary descriptors the performance rating will be $\left(\frac{5}{24}\times\frac{100\%}{1}\right)$.

Table 11.3 Shows an example where none of the primary groups have the same value. In this case the primary group 128 would be selected as it could be found in one of the secondary groups, but a relatively low performance result will be returned.

<table>
<thead>
<tr>
<th>Template No</th>
<th>Primary Group</th>
<th>Secondary Groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>128</td>
<td>(0,1,2,4,8,16,32,64)</td>
</tr>
<tr>
<td>2</td>
<td>129</td>
<td>(1,128,131,133,137,145,169,193)</td>
</tr>
<tr>
<td>3</td>
<td>130</td>
<td>(2,132,134,135,138,146,162,194)</td>
</tr>
</tbody>
</table>
Fig. 11.4 No agreement between primary groups
3. Conclusions

The kernel improved the performance of the Template Order Model enormously. It is a definite asset for a voice classification system, which makes use of our model. The performance rating supplied by the kernel can also be used to determine if an utterance should be accepted into the system or not. We would suggest that a template should only be accepted if it reached the 50% acceptance mark. If a template does not conform to these standards then the utterance should be repeated, which inevitably means starting at phase 1 of the Template Order Model.

We now conclude the second section of this dissertation. In the third section we will explain the practical system that implements our model in the Windows (tm) operating system and after that we will list the examples in Appendix A.
SECTION III
CHAPTER 12: TOM - Practical Implementation
In this chapter we will explain the practical system, designed by us, to implement the Template Order Model. This chapter will presume that the reader has a basic knowledge of the Windows operating system and that he or she understands the Object Oriented Design approach.

1. Introduction

We decided to create a practical system that could execute and verify the Template Order Model.

Our model is, up to this stage, only a theoretical idealization, and needs to be proven practically. We therefore designed the program called Tom.exe. In order for this system to explain our model it had to be extremely visual, that meant using pictures, bit maps and graphs. In order for any practical system to use these resources it had to be implemented with a Graphics Display Interface (GDI).

In this respect the Windows™ operating system was ideally suited. It had all the facilities we needed, such as a Graphics Display Interface, tools to create graphs, bit maps and icons and most importantly a means to implement an object oriented design approach.

We first had to create the background processing, this included means to access the voice extension card and an object to implement the Template Order Model.

The software needed to access the voice extension card, was supplied to us by the developers of the A/D card. This software was designed for a DOS environment, we therefore converted it to be applicable to the Windows™ operating system. Most of the voice processing algorithms were used as supplied to us by the developer of the A/D card.

We next set out to create the C++ source code that would call the voice card, receive the template from it and execute our Model.
In the following section of this chapter we will first discuss the way in which the Tom classes were defined and thereafter we will explain the Windows system using its menu structure.

2. TOM Class Definition

The first class' goal was to execute the commands supplied to us by the developers of the A\D card. This class had to collect the template from these functions and format it so that it may be in an understandable format for the classes to follow. It was called the Template Feeder. It contained the basic member functions to:

- Initialize the voice card

The card is interrupt driven. An interrupt service routine had to be created and the card parameters had to be initialized. These included setting the voice time-out and filter frequencies.

- Set the voice gain

This member function had to find the gain or "volume" setting for the user's voice.

- Create a LPC coded voice template

The voice samples had to be converted from analogue to digital and then be LPC coded.

- Compare a new template with one saved in memory

It was possible to compare two utterances. This member function then returned a result stating the accuracy of the match.

- Shut down the voice card.

Here functions had to be called that removes the interrupt service routine.

This class formed the base upon which the rest of the system would be constructed.
2.1 The TOM Model Class

This class implemented the Template Order Model. It has no physical ties to the Template Feeder. The class communicates to the rest of the world through its defined interface. It uses the template formatted by the Template Feeder and it returns the formatted output.

The template is executed following the model phases, described in chapter 7. These phases are:

- Sound Peak Filtering
- Start point detection
- Garbage removal
- Wave compression without creating group descriptors.

The class is designed in such a fashion so as to be able to execute the complete model in one swoop or to step through the model, pausing after each phase.

2.2 The TOM Zero Crossings Formatter

This class implements the zero-crossing formatter and returns as output the primary group descriptor with its eight secondary descriptors. The class communicates only through its interface and inherits no other classes. It has basically one member function, the zero-crossing formatter and this function returns a structure containing the group descriptors.
2.3 The TOM Kernel

The kernel, as described in the previous chapter, collects the output from the Zero-crossing formatter. Once no more output is received from that object it executes. This class implements the semi-intelligent algorithm known as the Kernel Algorithm. It searches through a basic decision tree to select the most appropriate primary group descriptor. It then returns a performance rating on this decision.

2.4 The TOM System class

This class inherits the four previous classes and thereby implements the Template Order Model. Its main function is to implement a TOM nerve center, that creates, calls and destroys the objects as they are needed.
Fig 12.1 The Class Definitions
3. The Windows Application

This program forms the kernel for use by the Windows operating system. The Windows operating system executes by message passing. The Windows application kernel, processes the messages it wishes to and discards the rest. It further passes messages to the TOM system class, which then implements the model depending on the messages received.

The Windows Application Kernel is also responsible for the management of internal resources such as icons, bit maps, graphs, accelerators, menus, dialogue and message boxes.

It is thus responsible for the whole Windows application.

We will now explain our system by using its menu structure.

3.1 The Main Menu

The main menu consists of four items, these are in order of appearance,

- The File Menu

This menu contains all the options needed to access template files as well as a means to exit the system, other than using the Window's system menu.

- The Template Menu

This menu contains the options that access the voice extension card.

- The Analyze Menu

The analyze menu holds the functions that implement the Template Order Model

- The Help Menu

The help menu displays what further information can be ascertained.
| File | Template | Analyse | Help |

Fig. 12.2 The TOM main menu
3.2 The FILE Menu

The file menu holds the option that accesses template files. These include means to Open or Save a template file. It further contains an option called About and the Exit system command.

3.2.1 The OPEN File Option

This menu option creates a dialogue box that holds the following information:

- The default file extension (*.vtp)
- The default file path (C:\Voice\Templates)
- Two list boxes that display possible paths and filenames for selection
- An Accept and Cancel Icon.

Any changes to the default file extension will be reflected in the path and file name list boxes. Any file extension can be used but we would advise the user to use "*.vtp". Only once the ACCEPT Icon had been selected would the file be opened and the contents read into memory.

This menu option was intended to be used to load voice templates from files, thereby bypassing the need to recreate a template through the TEMPLATE Menu.

3.2.2 The SAVE and SAVE AS menu options

These menu options are used to save a template created by the Template Menu to a disk file. The SAVE AS menu option allows the user to save the same template under a different file name. When any one of these options are selected a dialogue box is created containing the following information:

- The default file extension (*.vtp)
- The default file path (C:\Voice\Templates)
- Two list boxes that display possible paths and filenames for selection
- An Accept and Cancel Icon.

Any changes to the default file extension will be reflected in the path and file name list boxes. Any file extension can be used but we would advise the user to use "*.vtp". Only once the ACCEPT Icon had been selected would the file be created.

3.2.3 The ABOUT option

Keeping to other Windows applications we created this menu option. It's main purpose is to display a copyright notice.

3.2.4 The EXIT option

This option exits the system and destroys the window. Before doing so it however asks the user for confirmation by creating a dialogue box with two icons named YES and NO. If the YES icon is selected the system posts a quit message to the windows operating system and then shuts down the application.
Template Order Model

File    Template    Analyse    Help
Open    F3           
Save    F2           
Save As Shift F2    
About    
Exit    Alt X       

Fig. 12.3 The File menu
3.3 The Template Menu

The purpose of this menu is to call the voice card routines. It allows the user to initialize or reset the voice card and to create a voice template.

3.3.1 The INITIALIZE CARD option

This option sends a reset message to the Template Feeder object. It commands the object to reset the voice card. A result is returned that informs the user if this operation executed correctly.

3.3.2 The CREATE option

The create option instructs the card to sample the analogue input for a template. The user will be asked to repeat the utterance for a minimum of three times. This is done so that the kernel object has more than one template to compare before making its judgement.

This option will create a dialogue box with instructions for the user as well as a CANCEL icon. If this icon is selected the process of creating a template is aborted.
Fig. 12.4 The Template menu
3.4 The ANALYZE Menu

This menu holds the options that implement the Template Order Model on the template received by the TEMPLATE menu or the OPEN FILE option.

The ANALYZE menu holds two options, EXECUTE MODEL and a check option called STEP THROUGH.

This menu creates a dialogue box that holds a graphical representation of the template before and after the execution of the model. These graphs depict the waves on a X,Y axis system.

3.4.1 The EXECUTE MODEL option

This option executes the model depending on the check option STEP THROUGH. If STEP THROUGH is set then the TOM model object will pause after each phase in the model, thereby giving the user a look "inside" the model via the graphs.

If however STEP THROUGH is not set the model executes without pausing, giving the user a before and after graphical representation only.
### Template Order Model

<table>
<thead>
<tr>
<th>File</th>
<th>Template</th>
<th>Analyse</th>
<th>Help</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Execute Model</td>
<td>F9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>✓ Step Through</td>
<td></td>
</tr>
</tbody>
</table>

**Fig. 12.5 The Analyse menu**
The HELP menu supplies the user with a compressed version of this dissertation. It also supplies the user with further information on the Accelerators and the different icons available.

The help system makes use of the Windows Help facility.

---

**Fig. 12.6 The Help menu**

<table>
<thead>
<tr>
<th>File</th>
<th>Template</th>
<th>Analyse</th>
<th>Help</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Template Order Model Shift F1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Index</td>
<td></td>
<td>F1</td>
</tr>
</tbody>
</table>
The TOM Windows Application was designed to execute on Windows version 3.1. It is still compatible with Windows version 3.0 except that we do not guarantee that the application will execute under Real mode. Under Windows version 3.1 the application executes in standard and extended mode. The system makes use of a set of keyboard accelerators or hot-keys. We listed these below.

<table>
<thead>
<tr>
<th>Menu Option</th>
<th>Accelerator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open File</td>
<td>F3</td>
</tr>
<tr>
<td>Save File</td>
<td>F2</td>
</tr>
<tr>
<td>Save As File</td>
<td>Shift F2</td>
</tr>
<tr>
<td>Exit</td>
<td>Alt X</td>
</tr>
<tr>
<td>Initialise Voice Card</td>
<td>F4</td>
</tr>
<tr>
<td>Create Template</td>
<td>F5</td>
</tr>
<tr>
<td>Execute Model</td>
<td>F9</td>
</tr>
<tr>
<td>TOM Help</td>
<td>F1</td>
</tr>
<tr>
<td>Help Index</td>
<td>Shift F1</td>
</tr>
</tbody>
</table>
The source code for this application was not published with this dissertation, but part of the model classes can be found in Appendix A, as they were used as examples.

5. Conclusion

In this chapter we concluded section III of this dissertation by allowing insight into the practical system designed to prove the merit of the Template Order Model.
The Epilogue
Our aim with this dissertation was threefold, the first was to document the theoretical background of voice and speech recognition, the second was to create a model that could classify a human voice utterance and the final objective was to prove such a model.

These objectives were dealt with individually in separate sections of this dissertation. Section I explained the theoretical background. Section II defined the Template Order Model, a voice classification model. Section III explained the practical system designed to prove the Template Order Model.

We are satisfied that we have met or even surpassed our original goals.

When we first commenced the research into the subject of voice classification systems, we found that it was not going to be an easy problem to solve.

In the beginning we had a range of possible ideas, some plausible others ridiculous, but these ideas transformed in time to the Template Order Model.

We will admit that the model is by no means perfect. We therefore suggest the following areas for possible research:

- Greater accuracy, with the elimination of the need for secondary groups.

- Implementation of the model without it being dependent on the same utterance. That is to create a model where the human voice and not the human utterance is classified into groups.

- Voice-based database indexing models. This suggests a system where a database can be accessed by human voice alone.

We express the hope that further research in this study field will one day create these systems, and we further hope that other researchers will not be daunted by the expression: "Impossible".
In this appendix we have placed all source code examples referenced in chapters 8, 9, and 10. All the examples were implemented in the C++ language. The examples were extracted from our own code implementation and they follow the class definition explained in chapter 12.

1. Chapter 8.

Chapter 8 was concerned with the TOM phases - Template creation and Sound Peak filtering. Examples 8.1 and 8.2 address the phase of Template creation and example 8.3 Sound Peak filtering.

1.1 Example 8.1 - 8.2

Examples 8.1 and 8.2 call the voice card dependent functions and implement them in the Template_Feeder class.

C++ Header File : TOMINIT.HPP

```cpp
#ifndef TOMINIT_HPP
#define TOMINIT_HPP
#include "model.hpp"
extern "C" {
    #include "voice.h"
}
```

Appendix A-1
class Template_Feeder

Begin

byte * ClassTemplate;
byte IntrNo;
int TimeCut;
byte Gain;
int CardResult;
BOOLEAN CardInitialiseFlag;

Voice_Sample * Samples;

public :

int Template_Size;

Template_Feeder(int IntrruptNo,int TimeOut);

~Template_Feeder();

int InitialiseVoiceCard();

void FocusCard();

byte * CreateTemplate(int *TemplateSize);

int RecognizeTemplate();

Voice_Sample * MakeTemplate(byte TemplateNo, int * NoOfSamples);

Appendix A-2
C++ Source Code File: TOMINIT.CPP

#include "TOMInit.hpp"

Template_Feeder :: Template_Feeder(int InterruptNo, int TimeOut)

Begin

   IntrNo=InterruptNo;
   TimeCut=TimeOut;
   Samples=(Voice_Sample *) malloc(sizeof(Voice_Sample));
   CardInitialiseFlag=FALSE;

End

Template_Feeder :: ~Template_Feeder()

Begin

   free(Samples);
   if (CardInitialiseFlag EQ TRUE)
      FreeTemplate(ClassTemplate);

End
int Template_Feeder::InitialiseVoiceCard()
Begin
 Gain=0;
 CardResult=InitVoice(IntrNo);
 switch (CardResult)
 Begin
 case -1 :
 return(1); // Interrupt In Use at the Moment;
 case -2 :
 return(2); // No Hardware Detected;
 case 0 :
 return(0); // All is OK
 End
End

void Template_Feeder::FocusCard()
Begin
 Gain=SetGain(TimeCut);
End

byte * Template_Feeder::CreateTemplate(int *TemplateSize)
Begin

Appendix A-4
Template_Size=0;
ClassTemplate=VoiceTrain (TimeCut, &Template_Size,10);
*TemplateSize=Template_Size;
CardInitialiseFlag=TRUE;
return(ClassTemplate);
End

int Template_Feeder::RecognizeTemplate()
Begin
return(VoiceRecog (TimeCut, ClassTemplate));
End

Voice_Sample * Template_Feeder :: MakeTomTemplate(byte TemplateNo, int * NoOfSamples)
Begin

union byte_to_int
Begin
char byte_sample[2];
int int_sample;
End Template_sample;

int a;
for (a=0; a<2000; a++)
    Samples-sample[a]=0;

*NoOfSamples=(Template_Size-20)/ 18;
ClassTemplate+=20+(2*TemplateNo)-1;
for (a=0; a*NoOfSamples; a++)
    Begin
        Template_sample.byte_sample[0]=ClassTemplate[0];
        Template_sample.byte_sample[1]=ClassTemplate[1];
        Samples-sample[a]=Template_sample.int_sample;
        ClassTemplate+=18;
    End;
return(Samples);
End

1.2 Example 8.3

This example shows an extraction from the TOM model class. It shows the
initial filter bank operation in the member function Remove_Noise_Peaks() 
and the filter repeat operation in Execute_Filter_Banks(). In order to make
this example more understandable we included the class header file for 
this class.

Appendix A-6
C++ Header File: MODEL.HPP

#ifndef MODEL_HPP
#define MODEL_HPP

#include "struct.h"
#include "alloc.h"
#include "math.h"

const
MAXSIZE = 2000;

typedef struct
Begin
    int sample[MAXSIZE];
End Voice_Sample;

class Template_Order_Model
Begin
    Voice_Sample * Tom_Template;

Appendix A-7
int NoOfSamples;
void Load_Voice(Voice_Sample * Template);
void Remove_Noise_Peaks();

public :

Template_Order_Model (Voice_Sample * Template,int template_size);
~Template_Order_Model();
void Return_Voice (Voice_Sample * Template);
void Execute_Filter_Banks();
void Start_Point_Detection();
void Remove_White_Space ();
void Compress_Wave();
int Execute_TOM (Voice_Sample * Template);

End;
#endif

C++ Extraction from the source code file: MODEL.CPP

void Template_Order_Model :: Remove_Noise_Peaks()
Begin
int a;
double Previous, Current, Future;

double Difference;

for (a=1; a<NoOfSamples-1; a=a+1)

Begin

Previous = Tom_Template-sample[a-1]; // Previous Sample
Current = Tom_Template-sample[a]; // Current Sample
Future = Tom_Template-sample[a+1]; // Future Sample

Difference = ceil (Current * 0.95);

// Filter 1
if ((Previous EQ 0) AND (Current 1= 0) AND (Future EQ 0))

    Tom_Template-sample[a]=0;

// Filter 2
if ((Previous 0) AND (Current 0) AND (Future ))

    Tom_Template-sample[a]=0;

// Filter 3
if ((Previous 0) AND (Current 0) AND (Future 0))

    Tom_Template-sample[a]=0;

// Filter 4
if ((Previous 0) AND (Current 0) AND (Future EQ 0))

    Tom_Template-sample[a]=0;

// Filter 5
if ((Previous 0) AND (Current 0) AND (Future EQ 0))

    Tom_Template-sample[a]=0;
2. Chapter 9

In chapter 9 we examined the TOM phases - Start Detection and Garbage Removal. Examples 9.1-9.2 show another extraction from the TOM model class. Please note that the C++ Header File is shown in section 1 of this appendix.

2.1 Example 9.1

In this example we implemented the TOM phase of start detection. In the member function Start_Point_Detection(), we searched through the first 5 samples of the template for a sample value with a Y = 0 value. If this sample is found the template is zeroed from the beginning, up to the X-value of this sample.

Appendix A-10
void Template_Order_Model :: Start_Point_Detection()

Begin

int a,b,Point;

for (a=0;a<a++)

Begin

if (Tom_Template-sample[a] EQ 0)

Begin

Point=0;

NoOfSamples-=a+1;

for (b=a+1;b<=NoOfSamples;b++)

Begin

Tom_Template-sample[Point]=Tom_Template-sample[b];

Point++;

End;

a=5;

End

End

End
2.2 Example 9.2

This example shows how to implement the TOM Garbage Removal phase. It is implemented with the use of a member function named Remove_White_Space(). This function scans the entire template for samples with $Y = 0$ values. If such a sample is found, the template is left-shifted, so that these samples are effectively removed.

C++ Source Code Extraction From File: MODEL.CPP

```cpp
#include <iostream>
#include <vector>

void Template_Order_Model :: Remove_White_Space()
{
    int a, c;
    int Current, Future, PastFuture, FirstZero;
    int Scan, HoldSamples;

    for (a = 0; a < NoOfSamples - 1; a++)
    {
        Current = Tom_Template-sample[a];
        Future = Tom_Template-sample[a+1];
        if ((Current EQ 0) AND (Future EQ 0))
        {
            PastFuture = a + 2;
            FirstZero = a;
        }
    }
}
```

Appendix A-12
do
    Begin
        Scan=Tom_Template-sample[PastFuture];
        PastFuture++;
    End while ((Scan EQ 0) AND (PastFuture NoOfSamples));
    if (Scan != 0)
        Begin
            PastFuture--;
            HoldSamples=NoOfSamples-(PastFuture-FirstZero-1);
            for (c=FirstZero+1;c<=NoOfSamples-FirstZero;c++)
                Begin
                    Tom_Template-sample[c]=Tom_Template-sample[Past Future];
                    PastFuture++;
                End
            NoOfSamples=HoldSamples;
        End
    End
End
End

Appendix A-13
Chapter 10 was concerned with the implementation of the Zero-crossing Formatter. In example 10.1 we show how to compress the template, so that the Y-sample may only have three possible values (-1, 0, 1). In example 10.2 we implement the Binary Coding Method.

### 3.1 Example 10.1

In this example we created a member function called Compress_Wave(). Its only function is to compress positive Y-values into Y = 1 and negative values into Y = -1.

```cpp
void Template_Order_Model :: Compress_Wave()

Begin

int a;

for (a=0; a<=NoOfSamples; a++)

Begin

if (Tom_Template-sample[a] > 0)

Tom_Template-sample[a]=1;

```

Appendix A-14
if (Tom_Template-sample[a]<0)

    Tom_Template-sample[a]=-1;

End

End

3.2 Example 10.2

Example 10.2 is an extraction from the TOM Zero-crossing class. It is implemented with the use of a function called Calc_ZCR, which returns the structure ZeroCrossing-Register. In this structure the Primary group descriptor and its eight secondary descriptors can be found.

C++ Header File : TOMBIN.HPP

#ifndef TOMBIN_HPP
#define TOMBIN_HPP
#include "model.hpp"
typedef struct
Begin
    int Primary_Group;
    int Secondary_Groups[8];
End ZeroCrossingRegister;
class TOM_ZC_Method
Begin
int NoOfSamples;
int ZCM;
Voice_Sample * Template;
unsigned char ZCR_Hold[28];
char HoldFlag[3];
int Line;

TOM_ZC_Method(Voice_Sample * VoiceTemplate,
               int Template_size,
               int zero_crossing_mark);
~TOM_ZC_Method();
void Calc_ZCR (ZeroCrossingRegister * ZCR);
End;
#endif

C++ Source Code File : TOMBIN.CPP

#include "tombin.hpp"

TOM_ZC_Method::TOM_ZC_Method(Voice_Sample * VoiceTemplate,
                             int Template_size,int zero_crossing_mark)
Begin

ZCM=zero_crossing_mark;
NoOfSamples=Template_size;
Template=VoiceTemplate;
Line=0;
End

void TOM_ZC_Method :: Calc_ZCR (ZeroCrossingRegister * ZCR)

Begin

int Mark,Pos,Value,Scan,ZCLength,a;
int Primary,Mask;

Mark=0;
Mask=128;
Primary=0;
Pos=0;
do
Begin
Scan=Template-sample[Pos];
Pos++;
End while ((Scan EQ 0) AND (Pos <NoOfsamples)); // Scan for Non Zero
Pos--; 

while ((Pos NoOfSamples) AND (Mask 0)) 

Begin 

if (Scan != 0) 

Begin 

Mark=Pos; 

Value=Scan; 

do 

Begin  // Start Counter 

Scan=Template-sample[Pos]; 

Pos++; 

End while ((Scan EQ Value) AND (Pos < NoOfSamples)); 

if (Scan != Value)  // Crossing found 

Begin 

ZCLength=(Pos-1)-Mark; 

if (ZCLength ZCM) 

Begin 

Primary=(Primary | Mask); 

End 

Mask/=2; 

End 

End 

else Begin 


Appendix A-18
4. Conclusions

We hope that these examples enlightened the reader as to how the Template Order Model was implemented.
Appendix B
1. System requirements

The following constitutes the system requirements to execute the Template Order Model:

- 80286, 80386 or 80486 personal computer.
- Herc, Mono, CGA, EGA, VGA or Super VGA graphics display devices.
- Windows (tm) 3.0 or 3.1 operating system.
- 2 mb RAM
- 1mb secondary storage.
- A/D PC extention card. (First National Bank)
- An Industry quality microphone.
"Profundity is seldom achieved by misquoting the opinions of those who cannot return to defend themselves."

Bryce Courtenay. From Tandia. Mandarin Press. 1992
References


Mission earth vol 1 - The Invaders Plan
New Era Publications 1985


Computation of Language
Springer-Verlag 1987


Natural Language Generation
Springer-Verlag 1987

[4] Parsons T.

Voice and Speech Processing
McGraw-Hill. 1986


Marcel Dekker Inc. 1989.

[6] Bolec L.

Speech Communication with Computers
MacMillan 1978
[7] Sklansky, Dowden
   Pattern Recognition
   Hutchinson & Ross 1973

[8] Samet H
   The Design and Analysis of Spatial Data Structures
   Addison-Wesley 1990

[9] Geoff Bristow
   Electronic Speech Synthesis.

[10] Koenig W.
    The sound spectrograph.

    The speaking machine of Wolfgang von Kempelen.
    JASA vol 27 no. 4 pp 151-166. March 1950.

    Aeronautical RadioTelephony Procedures.

    Marcel Dekker Inc 1988
[14] Shannon C.E, Weaver W.

The mathematical theory of communication.

University of Illinois Press.

[15] Shannon C.E.

Communication in the presence of noise.


[16] Nyquist, H

[ NOT USED]

[17] Jayrant N.S, Noll P.

Digital Coding of Waveforms.

Prentice-Hall. 1984

[18] Makhoul J.

Linear Prediction : A tutorial view.


[19] Makhoul J.

Linear Predictive Coding. Chapter 5. Electronic Speech Synthesis


[20] Harris F.J.

On the use of windows for harmonic analysis with the discrete Fourier Transform.


Linear prediction of speech.

Springer-verlag. 1976
[22] Bednar J.B.

Spatial recursive filter design via rational Chebyshev approximations.

[23] Doddington G, Schalk T.

Speech Recognition: Turning theory into practice.


Machine-to-man communication by speech. Part II: Synthesis of prosodic features of speech by rule.


MITalk-79: Conversion of Unrestricted English Text to Speech.
MIT Press. 1979

[26] Chomsky, N Halle M.

The sound pattern of English.
Harper and Row. 1968

[27] Harris C.M.

A study of the building blocks of speech
JASA vol 25. Page 962-969. 1953


Natural communications with computers: speech compression research at BBN

Bib-4


Voice classification using a unique key signature. Chapter 5 Speech Recognition Revisited.


[31] Petzold, Charles

Programming Windows

Microsoft Press 1990
Index

A

Acoustics
  Fricative, stops, 2-15
Articulation, 2-5

C

C
  functional, 12-1
C++, 12-1
Coding, 3-5
  Adaptive quantization, 3-8
  Irreversible, Non-reversible, 3-8
  Non-linear, 3-8
  Predictive, 3-8
  Rate-distortion theory, 3-7
  Reversible, Non-reversible, 3-7
  Transform, Vector Quantization, 3-8

D

DOS, 12-1
Dudley, H, 5-2

F

Fast Fourrier Transform (FFT), 6-4
First National Bank, 7-9
  Verbal Links, 7-9
FORTRAN, 4-10

G

Graphics Display Interface (GDI), 12-1

IDX-1
H

Hamming and Hanning, 4-7
Hardware
  Voice Card, 12-2
Haskins Laboratories, 2-16

L

Linear Analyses
  LPC, 12-2
Linear Analysis, 3-8, 5-1, 6-2, 6-4
  All-pole speech synthesis model, 4-1
  Autocorrelation method, 4-7
  Data windowing, 4-7
  Direct-form analysis, 4-6
  Direct-form synthesis, 4-5
  Flat-spectrum excitation source, 4-1
Inverse Filter, 4-2
Lattice analysis, 4-9
Lattice Synthesis, 4-5
Linear Predictive Coding (LPC), 3-9, 4-1
Prediction, 2-15
Predictor Coefficient, 4-9
Predictor Coefficients, 4-4
Pulse and Noise sources, 4-6
Residual signal, 4-7
Residual window, 4-8
Spectral Envelope, 4-2
Spectral shaping filter, 4-1
Squared error, 4-7
The Gain Factor, 4-5

M

Myers and Rabiner
  Level building, 6-6
N

Noise, 1-3, 4-6
   Granular, 3-5
   Hiss, 4-6
   Random, 9-2
   White, 4-5
Nyquist
   Filter, 3-2
   Rate, 3-2

O

Object Oriented Design, 12-1

P

Parallel Architecture, 6-7
Phonetic, 9-1
   Allophones, 2-10
   Formant Transitions, 2-16 - 2-17
   International phonetics alphabet (IPA), 2-1
   Phonemes, 2-10
   Resonances, 2-14
   Vowel Waveforms, 2-15
Phonetics, 2-1
   Acoustic, 2-11
   ARPA, 2-1
   Articular, acoustical, 2-1
   Contoids, Vocoids, 2-5
   Point and Manner of Articulation, 2-8
   Transcription, 2-1
   Voicing, 2-8

Q

Quantising
   Delta Modulation, 3-4 - 3-5
Pulse Code Modulation (PC), 3-4
Quantizing, 3-4

R

Recognition, 7-1
Continuous, 6-2, 6-5
Digital filtering, 6-3
Feature extraction, 6-2
Generative model, 6-7
Phonetic variability, 6-5
Segmentation, 6-5
Template length, 6-4
Template matching, 6-4
Time-warping algorithm, 6-5
Zero-crossing technique, 6-3

Recognition
Isolated word, 6-2
Speaker-dependent, 6-2

S

Sampling, 3-1
Aliasing, 3-2
Over-sampling, down-sampling, 3-2
Shannon, 3-7
Speaker Identification, 6-1, 6-4
Spectral Envelope, 5-3
Spectrogram, 2-16, 6-3
Spectrograph, 2-12
Spectrum Analysis, 2-12
Speech, 1-4
Biological function, 1-6
Communication, 1-5
Definition, 1-5
Disadvantages, 1-6
Recognition and synthesis, 6-1
Signal, Wave, 3-1
Synthesis, 2-1, 5-1
Variation, 1-2

Speech Organs
Acoustics, 2-12
Standards, 1-2
Stewart, J.Q, 5-2

Synthesis
Analysis-synthesis method, 5-3 - 5-4
by Rule, 5-3, 5-5
Grapheme, Phoneme, 5-5
Inflections, 5-3
Information transmission, 5-1
LSI Technology, 5-1
MITalk system, 5-6
Naval Research Laboratories (NRL), 5-6
Production mechanism, 5-4
Vocoder, 5-2
Waveform coding, 5-3

Template, 1-6
Template Order Model, 6-7
Assumptions, 7-5
Binary Coding Method, 10-9, 11-1
DOT, DASH, 10-9
Filter Two, 8-8
Garbage remover, 7-7, 9-6, 10-3
Kernel, 11-1, 12-4
Model Class, 12-3
Performance Rating, 11-5
Primary, Secondary Groups, 10-9
Primary Descriptors, 11-5
Sample distance measurer (SDM), 10-3 - 10-4
Sample distance register (SDR), 10-6
Start detection, 7-7, 9-1, 9-4
System Class, 12-4
Template creation, 7-5, 8-2
Tom.exe, 12-1
Zero-crossing Formatter, 12-3
Zero-crossings, 7-7, 9-4, 9-7, 9-9
Zero-crossings detector (ZCD), 10-3 - 10-4
Zero-crossings formatter (ZCF), 10-7
Zero-crossings method, 10-1
Template Order Model (TOM), 1-6

V
Vocal organs, 2-1
Vocoid, 2-5
Voice classification
  Goals, 7-2
  System, 7-1
  Template Order Model, 7-5
Voice Recognition, 1-3
Von Kempelen, 5-2

W
Wave Equations, 2-15
Windows
  Application, 12-6
  Operating System, 11-10, 12-1
  Version 3.0 and 3.1, 12-16