RULES OF SELECTING PHONETIC UNITS IN A TEXT-TO-SPEECH SYNTHESIS SYSTEM

Christos Boubousis

A thesis submitted in partial fulfillment of the requirements for the degree of

Diploma in Electronics and Computer Engineering

Technical University of Crete

Chania 2008

Abstract

Rules of Unit Selection in a Text-To-Speech Synthesis System

by Christos Boubousis

Chairperson of the Supervisory Committee:
Professor Vassilios Digalakis

Department of Electronics and Computer Engineering

Text-To-Speech technology refers to the ability of a machine/computer to convert given text into speech. The machine should be able to carry out conversion of text of any format into speech output that can be understood by the listener. There are various techniques to accomplish Text-To-speech Synthesis. In this thesis we tried to understand the rules of selecting the phonetic units of a Text-To-Speech Synthesis System by creating a Weather Report Synthesizer, a Text-To-Speech Synthesis System for weather forecasts in Greek. In order to develop the Weather Report Synthesizer, we used the Festvox Tool of Festival Speech Synthesis System. It is a concatenative synthesis system which uses the vocal tract model as well as unit-selection in order to synthesize speech. Since there are in general multiple instances of each concatenative unit, the system performs dynamic unit selection. We used two implementation methods, the Residual-Excited LPC synthesis and the Cluster Unit Selection. The voice database amounted approximately ten hours of speech recordings and was constructed from read text taken from the weather forecasts of the National Meteorological Agency of Greece (EMY). Ultimately, the voice consists of a diphone database (LPC Synthesis) or a unit CART tree (Cluster Unit Selection Synthesis), a lexicon and a number of skeleton files that offer the complete voice. Only that set of files is required when people other than the developer of the voice wish to use our newly developed voice and needs to be distributed. We have distributed for the LPC method diphone group files, a single file holding the index diphone data itself and a set of Scheme files that describe the voice, while for Cluster Unit Selection a set of utterances, the CART tree with our units, a lexicon as well as a set of Scheme files which describe the voice.

Acknowledgements

I would like to express my sincere appreciation to Professor Vassilis Digalakis for his guidance during the design and implementation of this application and his assistance in the preparation of this manuscript. In addition, special thanks to Associate Professor Alexandros Potamianos and Assistant Professor George Karystinos for the evaluation of this work and participation of the Supervisory Committee, as well as,Vassilis Diakoloukas and Chris Vosnidis for their important help and guidance during the time of the preparation of this thesis.

I would also like to thank my friends Ntousias Alexandros and Stylianakis George for their valuable help in the subject of text processing. Furthermore, all my friends Michael, Aggelos, Despoina, Tolis, Tasos, John etc (you know who you are!) for your assistance and your mental support all these years of my studies in Chania.

Finally, I would lik e to thank my family that supported and continues to support me not only in my studies, but in every aspect of my life.

Table of Contents

CHAPTER 1 .... *10*
INTRODUCTION 10
1.1. History of Speech Synthesis 10
1.2. Structure of the thesis 11

CHAPTER 2 .... *15*
TEXT-TO-SPEECH SYNTHESIS 15
2.1. What is Text-To-Speech Synthesis? 15
2.1.1 Types of input 16
2.1.2 Basic Methods 16
2.2. Practical Applications 17
2.3. General Anatomy of a Synthesizer 20
2.3.1. The NLP Component 22
2.3.1.1 Text Preprocessing 23
2.3.1.2 Word Pronunciation 24
2.3.1.3 Prosody Generation 26
2.3.1.4 Accentuation 27
2.3.1.5 Intonational Phrasing 28
2.3.1.6 Segmental Duration 29
2.3.1.7 Sentence Intonation 30
2.3.2. The DSP Component 31
2.3.2.1 Rule-based Synthesis 31
2.3.2.2 Synthesis by Concatenation 34
2.4. Speech Synthesis 37
2.5. Segmental Quality 37

CHAPTER 3 .... *40*
RESIDUAL-EXCITED LPC DIPHONE SYNTHESIS 40
3.1. Introduction 40
3.2. Definition of the Greek Phoneset 42
3.3. Definition of the Diphone List 46
3.4. Corpus Design 48
3.4.1 Selection of Sentences 48
3.4.2 Corpus Characteristics 49

3.4.3 Word Selection........49
3.4.4 Final Corpus........50

3.5. The Recording Phase........50

3.6. Labeling........51

3.7. Text Analysis........55

3.8. Segmentation........58

3.9. Energy Normalization........59

3.10. Pitchmark Extraction........59

3.11. Building LPC Parameters........63
3.11.1 Digital Vocal Tract Model........63
3.11.2 LPC Coefficients Calculation........69
3.11.2.1 Recursive Levidson Durbin Algorithm........73
3.11.2.2 Gain Calculation........75
3.11.2.3 Some Practical Issues........76
3.11.2.4 Voiced/Unvoiced Detection........76

3.12. Building Prosodic Models........78
3.12.1 Phrasing........78
3.12.2 Duration Models........79
3.12.3 Intonation........81

CHAPTER 4........*83*
LIMITED-DOMAIN CLUSTER UNIT-SELECTION SYNTHESIS........83

4.1. Introduction........83

4.2. Cluster Unit-Selection........85
4.2.1 Building Utterance Structures........86
4.2.2 Cepstrum Parameter Files Calculation........88
4.2.3 Building the Clusters........89

4.3. Classification and Regression Trees (CART)........97
4.3.1 Advantages of CART Methods........99
4.3.2 CARTS in Festival........100
4.3.2.1 Impurity........101
4.3.2.2 Question Forming........102
4.3.2.3 Tree Building Criteria........102
4.3.2.4 Tree Format........104

4.4. Wagon CART-Building Program........105
4.4.1 Feature Vectors........105
4.4.2 Data Descriptions........106

CHAPTER 5........*109*
EVALUATION OF THE GREEK WEATHER FORECAST SYNTHESIZER
........109

5.1. Introduction....................109
5.2. Taxonomy of the Evaluation Tasks & Techniques....................110
5.2.1 Black Box vs Glass Box....................111
5.2.2 Laboratory vs Filed....................112
5.2.3 Acoustic vs Linguistic....................112
5.2.4 Subjective vs Objective....................113
5.2.5 Judgement vs Functional....................114
5.2.6 Global vs Analytic....................114
5.3. Evaluation of the Greek Weather Forecast Synthesizer....................115
5.3.1 Segments....................115
5.3.2 Prosody....................115
5.3.3 Voice Quality....................116
5.3.4 Overall Voice Quality....................116
5.4. Test Method....................117
5.5. Results....................118
CHAPTER 6....................*124*
CONCLUSIONS & FUTURE WORK....................124
6.1. Conclusions....................124
6.2. Future Work....................125
BIBLIOGRAPHY....................127

List of Figures

Figure 1 : Block Diagram of a TTS Synthesizer....21
Figure 2 : Block Diagram of the NLP Component....23
Figure 3 : Different kind of information provided by intonation (lines indicate the pitch movement while the solid lines indicate the stress) a. Focus or giving new information. b. Relationship between words (I-early, wake-early). c. Finality or continuation. d. Segmentation of the sentence into groups of syllables 26
Figure 4 : Block Diagram of the DSP module of a Rule-Based Synthesizer....33
Figure 5 : Block Diagram of the DSP module of a Concatenative Synthesizer....34
Figure 6 : The Markov Generation Model....54
Figure 7 : Pitchmarks in a waveform signal....60
Figure 8 : Bad Pitchmarks in a waveform signal....60
Figure 9 : Close-up of pitchmarks in a waveform signal 61
Figure 10 : Close-up of pitchmarks in a waveform signal (2) 61
Figure 11 : Close-up of pitchmarks in a waveform signal (3) 62
Figure 12 : Simplified view of the vocal tract....64
Figure 13 : Two kinds of input to generate sound (T_0 : pitch period) 65
Figure 13a : Glottal Pulse Excitation for voiced sounds 64
Figure 13b : Hiss(White Noise) input for unvoiced sounds....65
Figure 14a : Simplified Model of speech production 66
Figure 14b : Detailed LPC Speech Production Model....66
Figure 15 : Linear Prediction of a filter 67
Figure 16 : Speech Analysis Filter 68
Figure 17 : Speech Synthesis Filter....68
Figure 18 : Relationships among dimensions involved in Taxonomy of speech output evaluation methods....111
Figure 19 : Evaluation Results (Voice Quality) 119
Figure 20 : Evaluation Results (Voice Acceptance)....119
Figure 21 : Evaluation Results (Intonation)....120
Figure 22 : Evaluation Results (Phoneme Junctures)....120
Figure 23 : Evaluation Results (Context Apprehension) 121
Figure 24 : Evaluation Results (Voice Quality) 122
Figure 25 : Evaluation Results (Voice Acceptance)....122
Figure 26 : Evaluation Results (Intonation)....123
Figure 27 : Evaluation Results (Phoneme Junctures)....123
Figure 28 : Evaluation Results (Context Apprehension) 124

List of Tables

Table 1: Example of Homographs in English...25
Table 2: Word distinction according to accentuation..27
Table 3: Example of implementation of *chink n' chunk* algorithm.....................................29
Table 4: Greek Phoneset in Festival...43
Table 5: Phone Feature Names and Values..44
Table 6: Greek Phoneset with SAMPA equivalent,examples and Hellenic and English transcription..45
Table 7: Function generating the vowel-vowel diphones...46
Table 8: Statistical Analysis of the Original Data Set...49
Table 9: Statistical Analysis of the final Corpus...50
Table 10: Token-to-Word mapping function..57
Table 11: Diphone Index File Sample..58
Table 12: Levidson Durbin Recursive Algorithm...73
Table 13: Phrase-Breaking...79
Table 14: Duration Prediction Tree...80
Table 15: Greek Phones Average Durations...80
Table 16: Accent Prediction CART tree..81
Table 17: Pitch range of the speaker..81
Table 18: Initial Parameter set..90
*Table*19: Function calls for loading the acousting parameters and calculation of their acoustic distance..91
Table 20 Acoustic parameters setting...92
Table 21: Function for dumping features..93
Table 22: Cluster Building function...94
Table 23: Cluster Building parameters..95
Table 24: Function for the creation of the unit catalogue...95
Table 25: Calculation of the entropy...101
Table 26: Syntax of a CART...104
Table 27: Example of a feature vector..106
Table 28: Example of a description file...107
Table 29: Example of our Evaluation form..118

Chapter 1

Introduction

1.1. History of Speech Synthesis

The idea that a machine could generate speech has been with us for some time, but the realization of such machines has been practical within the last fifty years. Even more recently, it’s in the last twenty years that we have seen practical examples of Text-to-Speech systems that can say any text they are given. The creation of synthetic speech covers a whole range of processes, and though often they are all lumped under the general term Text-to-Speech, a good deal of work has gone into generating speech from sequences of speech sounds; this would be a speech-sound(phoneme) to audio waveform synthesis, rather than going from text to phonemes and then to sound.

One of the first practical application of speech synthesis was in 1936 when the UK telephone company introduced a speeking clock. It used optical storage for the phrases, words and parts of speech ("noun" "verb" and so on) which were appropriately concatenated to form complete sentences.

With the rise of digital representations of speech, digital signal processing and the proliferation of cheap, general purpose computer hardware, more work has been done in concatenation of natural recorded speech. *Diphones* appeared; that is two adjacent half- phones (context dependent phoneme realizations), cut in the middle, joined into one unit. The justification was that phone boundaries are much more dynamic than stable, interior parts of phones, and therefore mid-phone is a better place to concatenate units, as the stable points, have by definition, little rapid change, whereas there are rapid changes at the boundaries that depend upon the previous and next unit.

The rise of concatenative synthesis began in the 70s, and has largely become practical as large scale electronic storage has become cheap and robust. When a megabyte of memory was a significant part of researcher's salary, less resource-intensive techniques were worth their… weight in saved cycles in gold, to use an odd metaphor. Of course, formant synthesis can still require significant computational power, even if it requires less storage.

Techniques were developed to compress (code) speech in a way that it could be more easily used in applications. The Texas Instruments Speak 'n Spell toy, released in the late 70s, was one of the early examples of mass production of speech synthesis. The quality was poor, by modern standards, but for the time it was really impressive. Speech was basically encoded using LPC (Linear Prediction Coding) and it mostly used isolated words, though there were also a few phrases formed by concatenation. Simple Text-to-Speech (TTS) engines based on specialized chips became popular on home computers such as the BBC Micro in the UK and Apple II.

Before 1980, research in speech synthesis was limited in large laboratories that could afford to invest the time and money for hardware. By the mid-80s, more labs and universities started to join in as the cost of hardware dropped. By the late-80s, purely software synthesizers were feasible. The speech quality was still decidedly inhuman (and largely still is), but it could be generated real-time. Of course, with faster machines and large disk space people started to look to improving synthesis by using larger and more varied inventories for concatenative speech. Yoshinory Sagisaka at Advanced Telecommunications Recearch (ATR) developed nuu-talk in the late 80s and early 90s [11]. It introduced a much larger inventory of concatenative units; thus, instead of one example of each diphone unit there could be many, and an automatic acoustically based distance function was used to find the best selection of sub-word units from a fairly broad database of general speech. This work was done in Japanese, which has a much simpler phonetic structure than English, making it possible to get high quality with relatively small databases.

With the demonstration of general unit-selection synthesis in English in Rob Donovan's PhD work and ATR's CHATR system ([12] and [3]) by the end of 90s, unit selection had become a hot topic in speech synthesis research. However, despite examples of it working excellently, generalized unit selection is known for producing very bad synthesis from time to time.

Of course, the development of speech synthesis is not isolated from other developments in speech technology. Speech recognition, which has also benefited from the reduction in cost of computational power and increased availability of general computing into the populace, informs the work on speech synthesis and vice versa. There are now many more people who have the computational resources and interest in running speech applications, and this ability to run speech applications puts the demand on the technology to deliver both working recognition and acceptable quality speech synthesis. The availability of free and semi-free synthesis systems, such as the Festival Speech Synthesis System [2] and MBROLA [14] Project, makes the cost of entering the field of speech synthesis much lower, and many more groups have now joined in the development.

However, although we are now at the stage where talking computers are with us, there is still much work to be done. We can now build synthesizers of any language that can produce recognisable speech, with a sufficient amount of work; but if we are to use speech to receive information as easily when we are talking with computers as we do in everyday conversation, synthesized speech must be natural, controllable and efficient both in the rendering and in the building of a new voice.

1.2. Structure of the thesis

Chapter 1 provides a historical retrospection of Speech Synthesis from the early years of last century until nowadays.

Chapter 2 introduces us to the Text-To-Speech Synthesis, categorizing the types of inputs to a speech synthesizer, as well as the basic methods used for a synthesis procedure, some practical applications and the general anatomy of a synthesizer.

Chapter 3 describes the process of creating our Greek Weather Forecast Synthesizer with the vocal tract model providing explanation on every step of the process. All the technical issues that have arisen during the implementation are analyzed in every detail.

Chapter 4 describes step-by-step the implementation process of our Greek Weather Forecast Synthesizer using this time the unit-selection model providing every technical detail required.

Chapter 5 outlines our two approaches towards the definition of an evaluation procedure for the results of our application and presents our findings.

Chapter 6 provides a conclusion of our work and some propositions for future work.

Chapter 2

TEXT-TO-SPEECH SYNTHESIS

2.1. What is Text-to-Speech Synthesis?

Firstly, let's try to understand what is meant by term Text-to-Speech Synthesis. Obviously, this term refers to the creation by the computer of human-like speech. A Text-to-Speech Synthesizer (TTS) should be able to read *any* text aloud. Synthesized speech output may come from a wide range of processes that differ enormously in the nature of their inputs and the nature of their internal structures and calculations.

2.1.1 Types of Input

The input to a speech synthesizer may be

- an uninterpreted reference to a previously recorded utterance
- a message drawn from a small finite class of texts, such as telephone numbers
- a message drawn from a larger, but still restricted, class of texts such as names and addresses.
- a message drawn from unrestricted digital text, including anything from electronic mail to online newspapers to patent or larger texts, novels or cookbooks
- a message drawn from non-textual computer data structures
- a specification of the phonological content of a message, which for most applications must be produced from one of these types of input given previously

Most commercial applications so far have been of the first or the second type. Classical Text-to-Speech systems are of the fourth and sixth type, while ultimate human computer interaction is likely to be of the fifth type. A large number of people involved in applying speech synthesis technology think that the most promising current opportunity is the third type. Note that limited domain applications have been crucial to the success of computer speech recognition. Most practical speech synthesis implementations, including our application, belong to this category.

2.1.2 Basic Methods

The system internal structures and processes of speech synthesis involve

- reproduction of digital stored human voice, perhaps with compression or expansion
- construction of messages by concatenation of digitally stored voice segments
- construction of messages by concatenation of digitally stored voice segments, probably with modifications of the original time and pitch.

- construction of messages by concatenation of digitally stored voice segments, with rule-generated synthetic speech contours and rule generated segmented timing values
- construction of messages using rule-generated synthetic time-functions of acoustic parameters.
- construction of messages using rule-generated controls for the kinematics of the simplified analogs of the vocal tract.
- construction of messages by realistic modeling of the physiological and physical processes of human speech production including dynamic control of articulation and models of the airflow dynamics of the vocal tract.

The largest scale of commercial activity has been of the first and second type which are called stored voice. Much classical speech synthesis research has been of the fifth and sixth type the so-called *formant synthesis*, although the best current systems and the most active areas of research are of the third and fourth type techniques that are called *concatenative synthesis*.

2.2 Practical Applications

Every synthesizer is actually the imitation of the human reading capability, submitted to certain technological constraints that are characteristic of the time of its creation. *High Quality TTS synthesis* can have numerous practical applications such as:

- Telecommunication services

  TTS systems enable you to access textual information over the telephone. Knowing that about 70% of the telephone calls actually require very little interactivity, such a prospect is worth to be considered. Text might range from simple messages to huge databases which can hardly be stored as digitised speech. Queries to suchinformation retrieval systems could be put through the user’s voice with the help of a speech recognition system, or through the telephone keypad with DTMF systems.

VoiceXML is a programming language designed for creating applications that enable access over the phone to information, already available through a classical web browser. Using a server resident Voice browser, the telephone keypad as the method of the data input and server side speech synthesis as the method of data output, information services already offered on the Web could easily be modified to support mobile users. Given the continuously expanding number of mobile phones, and the Personal Digital Assistants, it seems that voice could be the means to address the needs of this new type of information services' users.

- Aid to handicapped persons

Voice handicaps originate from mental or sensation disorders. Machines can be an invaluable help especially to the latter case. With the help of an especially designed keyboard and a fast sentence assembling, synthetic speech can be produced very rapidly to overcome these impediments. A characteristic example of this situation, is the Astrophysicist Stephen Hawking. He suffers from a very rare disease called amyotrophic lateral sclerosis. Due to his illness, he cannot speak. However, thanks to a TTS system not only he can speak, but he gives lectures constantly. Blind people can also benefit from the TTS Systems when coupled with Optical Recognition Systems (ORS) which give them access to written information.

- Language education.

High Quality TTS Systems combined with a Computer Aided Learning tool, can be used as a method of learning a new language. This has not been done yet, however due to the constantly improving quality of commercial systems it is only a matter of time.

- Talking books and toys.

The toy market has been affected by the boost of speech synthesis. Many speaking toys have appeared in the market under the impulse of the innovative "Magic Spell" from Texas Instruments. High Quality Synthesis at affordable price, can also assist the corporations of the toy market to expand the capabilities of these products to the educational area.

- Vocal monitoring.

  In some cases, oral information is more efficient than written messages. The appeal is stronger, while the attention may still focus on other visual sources of information. As a result, many corporations orientated to develop speech synthesizers for measurement or control systems.

- Fundamental and applied research.

  TTS synthesizers have a rather bizarre feature which make them an excellent laboratory tool for linguists: they are completely under control. So, the repeated experiences provide identical results, allowing the investigation of the efficiency of intonative and rythmic models. A particular type of TTS systems, which are based on a description of the vocal tract through its reasonant frequencies *(formants)* and known as *formant synthesizers*, have been extensively used by phoneticians to study speech in terms of acoustical rules. In this manner, for example, articulatory constraints have been enlightened and formally described.

2.3 General anatomy of a Synthesizer

Anyone could think that the problem of converting the written text into speech could be “the problem of speech recognition in reverse”. Nevertheless, it’s a little more complicated. Speech recognition systems convert the speech input into a sequence of words recorded by the speaker. As a result, anyone would think that a Text-To-Speech Synthesizer just takes a sequence of words, converts every word into speech and concatenates the result.

However, this is a very simplistic approach of the problem. When you read a text in order to sound natural and as if you understand what you are reading, you must emphasize *(accent)* some words and de-emphasize others, you must chunk the sentence into meaningful (*intonational)* phrases; you must pick an appropriate F0 contour *(fundamental frequency)*; you must control certain aspects of your voice quality; you must
know that if a word appears in specific spots in a sentence you must pronounce it longer while in other spots shorter, since *segmental durations* by various factors one of them being phrasal positions.

Hence, the task of a TTS system is very complex, as it has to mimick what human readers do. In addition, TTS systems have another significant flaw. They cannot understand what they are reading as they have very little grammatical knowledge of a language. As a result, TTS algorithms have to do their “best”, using whatever grammatical information in order to decide on such things as *accent, intonation, phrasing* so they can produce the ideal result.

We can identify two basic parts in a Text-To-Speech Synthesizer. The first one is the Natural Languge Processing Component *(NLP)* and the second is the Digital Signal Proseccing Component. The former can be divided into two sub-parts. The first one includes the conversion of raw text to identified words and basic utterances and the second one does the linguistic analysis finding pronunciation of words and assigning prosodic structure to them
including phonemes to be produced and their duration, the duration and location of the pauses and an F0 contour to be used. The latter, is the actual speech synthesis part which takes the information and from a fully specified form *(pronunciation and prosody)* generates the waveform.

The block diagram of a very general Text-To-Speech synthesizer is the following

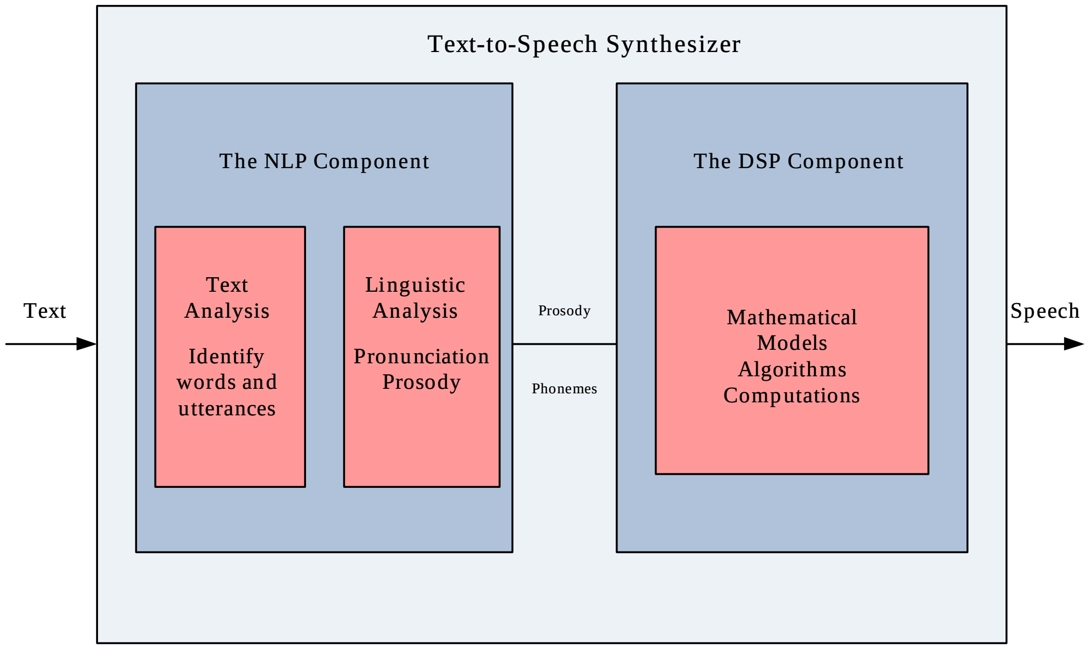


Figure 1. Block Diagram of a TTS Synthesizer

These partitions are not absolute but it is a very good way to chunk the problem. Of course different waveform generation techniques may need different types of information. *Pronunciation* may not always use standard phones and intonation may not necessarily mean an F0 contour. However the main path of the above diagram is absolutely right.

2.3.1 The NLP Component

Although text analysis is often considered as a trivial problem, anyone who has listened to general Text-To-Speech systems quickly realises it is not as easy to pronounce text as it first appears. Numbers, symbols, acronyms, abbreviations, appear to various degrees and in different types of text like news, novels and do not have a simple pronunciation that can be found merely by looking up the token in a lexicon or using letter-to-sound rules. In any language and in any limited domain (time, weather etc) that you wish to convert text to speech building an appropriate text analysis module is necessary. The Natural Language Processing Component of the TTS synthesizer is used to perform text and linguistic analysis on the input text and can be chunked to the following sub-parts:

- *Text preprocessing* : including end of sentence detection, “text normalization” (expansion of numerals and abbreviations) and little grammatical analysis, such as grammatical part of speech assignment.
- *Word Pronunciation* : including the pronunciation of every word and disambiguation of homographs.
- *Accent Assignment* : the assignment of levels of prominence to various words in the sentence.
- *Intonational Phrasing* : the chunking of long stretches of text into one or more intonational units.
- *Segmental Durations* : the determination of appropriate durations for the phonemes in the input on the basis of linguistic information computed as far.
- *F0 contour computation* : computation of the fundamental frequency.

The block diagram of the NLP module is shown in the following figure:

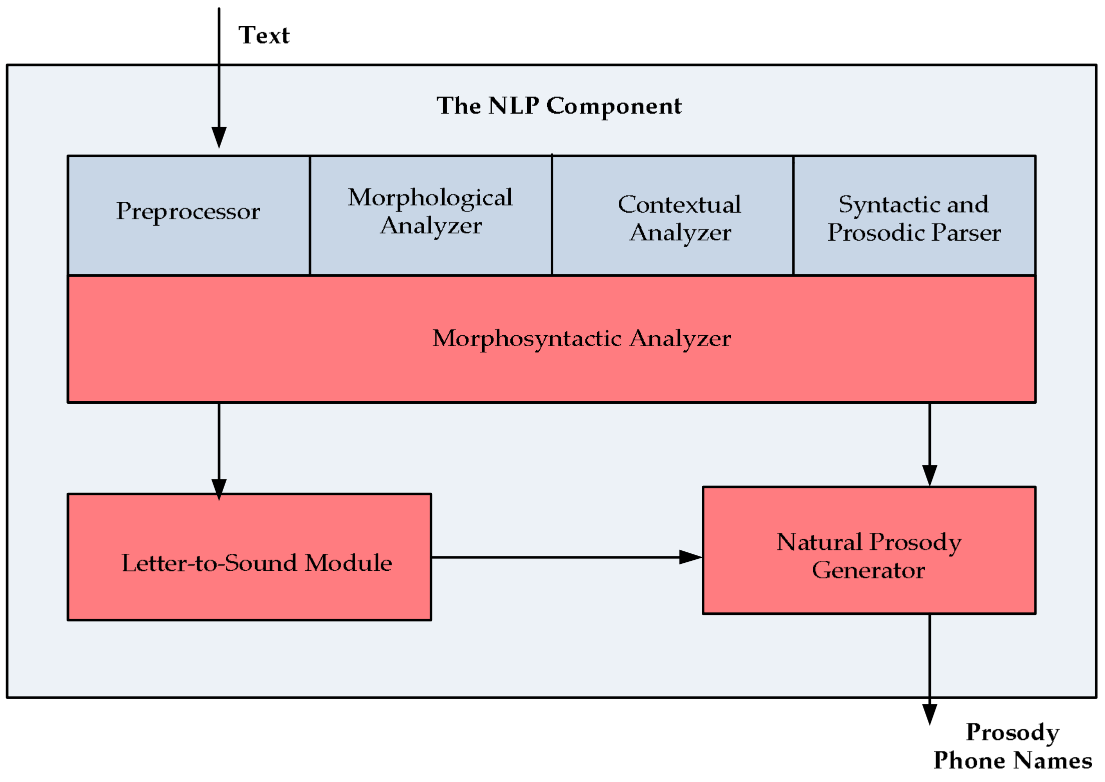


Figure 2. Block Diagram of the NLP Component

2.3.1.1 Text Preprocessing

The Text Preprocessing Module performs the following steps:

- It splits the input sentence into a manageable list of words. It identifies numbers, idioms, abbreviations, acronyms and transforms into full text.
  It performs a morphological analysis on the input text in order to propose all possible part of speech categories for each word taken individually on the basis of their spelling. Inflected, derived and compound words are decomposed into their elementary graphemic units (their *morphs)* using basic grammar rules exploiting lexicon of stems and affixes.
- Words are considered in their context, which allows for the reduction of the list of their possible part of speech categories to a very restricted number of highly probable hypotheses, given the corresponding possible parts of speech of neighboring words. This can be achieved either by *n-grams,* which local syntactic dependences in the form of probabilistic finite state automata such as Markov Models to a lesser extent with *multi-layer percetrons*, such as neural networks trained to uncover contextual rewrite rules, or with *local, non stochastic grammars* by expert linguists or automatically inferred from a training data set with *classification and regression trees (CART trees see Chapter 4 section 3 for further analysis)*.
- Finally a syntactic-prosodic parser, which examines the remaining search space and finds the text structure (i.e its organization into clause and phrase-like constituents) which more closely relates to its expected prosodic realization.

2.3.1.2 Word pronunciation

The Word Pronunciation Module is responsible for the automatic determination of the phonetic transcription of the incoming text. Hence, anyone would think that this process is equivalent of a dictionary look-up! However, if we take a deeper examination, we quickly realize that most words appear in genuine speech with several phonetic transcriptions, many of which are not even mentioned in pronunciation dictionaries. Specifically:

- Pronunciation dictionaries refer to word roots only. They do not explicitly account for morphological variations (i.e plural, feminine, conjugations especially for highly inflected languages) which therefore have to be dealt with by a specific component of phonology, called morphophonology.
- Some words actually correspond to several entries in the dictionary, or more generally to several morphological analysis with different pronunciation. This is the case of the homographs, words that are pronounced differently although they have

  the same spelling. Their correct pronunciation generally depends on their part-of-speech and most frequently verbs and non-verbs. The following table shows some examples of homograph words in English language.

dove	/dʌv/ (noun)	/doʊv/ (verb)
live	/lɪv/ (verb)	/laɪv/ (noun)
number	/ˈnʌm.bɚ/ (noun)	/ˈnʌ.mɚ/ (adjective)
read	/ɹid/ (verb)	/ɹɛd/ (verb) past tense
wind	/waɪnd/ (verb)	/wɪnd/ (noun)

Table 1. Examples of homographs in English

- Pronunciation dictionaries merely provide something that is closer to a phonemic transcription than from a phonetic one as they refer more to phonemes than to phones. Consonants, for example, may reduce or delete in clusters, a phenomenon called consonant cluster "simplification", e.g 'softness' /s fnIs/ where /t/ fuses in a single gesture with the following /n/.
- Words embedded into sentences are not pronounced as if they were isolated. This does not only originate in variations in the word boundaries, but also on alternations based on the organization of the sentence into non-lexical units,

  that is whether into groups of words (as for phonetic lengthening) or into non-lexical parts thereof.

- Finally, not all words can be found in a phonetic dictionary: the pronunciation of new words and of many proper names has to be deduced from the one of already known words.

The first two bullets rely on preliminary morphosyntactic analysis of the sentences to read. To a lesser extent it also happens to be the case for the third bullet as well, since reduction processes are not only a matter of context-sensitive phonation, but they also rely on morphological structure and on word grouping, that is on morphosyntax. Fourth bullet puts a strong demand on sentence analysis, whether syntactic or metrical, and fifth bullet can be partially solved by addressing morphology and/or by finding graphemic analogies between words.

2.3.1.3 Prosody Generation

The term *prosody* refers to certain properties of the speech signal, which are related to audible changes in pitch, loudness and syllable length. Prosodic features have specific functions in speech communications. The most apparent effect of prosody is that of focus. For instance, there are certain pitch events which make a syllable stand out within the utterance, and indirectly the word or syntactic group it belongs to will be highlighted as an important or a new component in the meaning of that utterance. The presence of a focus marking may have various effects, such as contrast, depending on the place where it occurs, or the semantic context of the utterance.

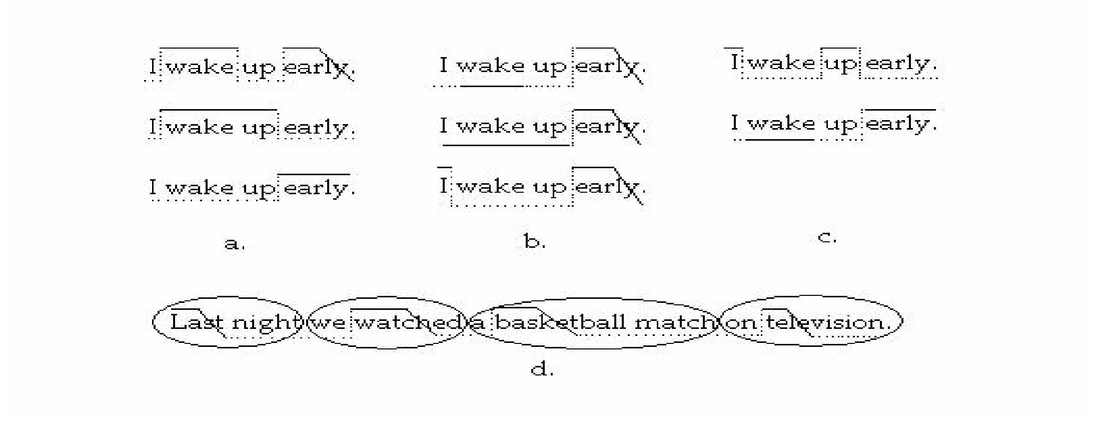


Figure 3. Different kind of information provided by intonation
(lines indicate the pitch movement while the solid lines indicate the stress)
a Focus or giving new information.
b Relationship between words (I-early, wake-early).
c. Finality or continuation.
d.Segmentation of the sentence into groups of syllables.

Prosodic features create a segmentation of the speech chain into groups of syllables, or, put the other way round, they give rise to the grouping of syllables and words into larger chunks. Moreover, there are prosodic features, which indicate relationships between such groups, indicating that two or more groups of syllables are linked in some way. This grouping effect is hierarchical, although not necessary identical to the syntactic structure of the utterance.

2.3.1.4 Accentuation

Various words in a sentence are accosiated with accents, which are usually manifested as upward or downward movements of fundamental frequency. Accentuation, along with intonational phrasing, and F0 contour are the main part of the biggest problem of prosody generation. Words are typically distinguished into three groups, with regard to their *prominence*. The two are *accented* and *unaccented* an the third is *cliticised*. *Cliticised* words are those which are *unaccented* but they have also lost their word stress, so that they tend to be short in duration: in effect they behave like unstresses affixes, even though they are seperated words. Accents are assigned primarily on the basis of broad lexical categories or parts of speech.

In the following table we can see the distinction of the words in the three categories: Content words such as nouns, verbs, adjectives in general tend to be accented; on the other hand, function words including auxiliary words and propositions tend to be unaccented

Accented	*Unaccented*	*Cliticised*
nouns	auxiliary verbs	short function words
verbs	propositions	
adjectives	adverbs	

Table 2. Word distinction according to accentuation.

However, more complex accentuation schemes based on syntactic and semantic analysis have been used providing better results.

2.3.1.5 Intonational Phrasing

Most commercially developed *TTS* systems have emphasized coverage rather than linguistic sophistication by concatenating their efforts on text analysis strategies aimed to segment the *surface structure* of incoming sentences as oposed to their syntactically, semantically and pragmatically related *deep structure*. The resulting syntactic-prosodic descriptions organize sentences in terms of prosodic groups strongly related to phrases (also termed as *minor* or *intermediate phrases*), but with a very limited amount of embedding, typically a single level of these minor phrases as parts of higher-order prosodic phrases (also termed as *major* or *intonational phrases*, which can be seen as the prosodic-syntactic equivalent for clauses) and a second one for these major phrases as parts of sentences, to the extent that the related major phrase boundaries can be safely obtained from relatively simple text analysis methods. In other words, they focus on obtaining an acceptable segmentation and translate it into the continuation or finality marks of Figure 3.c but ignore the relationships or contrastive meaning of Figure 3.a and b.

Liberman and Church [13], for instance, have reported on such a very crude algorithm, termed as *chinks n' chunks* algorithm, in which prosodic phrases (which they call *f-group)* are accounted by a simple regular rule:

a (minor) prosodic phrase = a sequence of chinks followed by a sequence of chunks.

in which chinks and chunks belong to sets of words which basically correspond to function and content words, with the difference that objective pronouns (like "*him*" or "*them*") are seen as *chunks* and tensed verb forms are considered as *chinks*. They show that this approach produces efficient grouping in most cases, slightly better than the simple decomposition into sequences of function and content words as shown in the example below:

function words/content words	chinks n chunks
I asked	*I asked them*
them when was	*when was the match on television*
the match	*and they said*
on television	*at nine*
and they said	
at nine	

Table 3. Example of implementation of *chink n' chunk* algorithm.

Other, more sophisticated approaches include syntax-based expert systems and automatic corpus based methods as with the *classification and regression tree* (*CART*) techniques [9].

2.3.1.6 Segmental Durations

Once the phonemes to be produced by the synthesizer have been computed, it is necessary to decide how long to make each one. What duration to assign to a phonemic segment depends on many factors, including:

- The identity of the segment in the question.
- The stress of the syllable of which the segment is a member.
- Whether the specific syllable bears an accent.
- The quality of the surrounding segments.
- The position of the segment in the phrase.

Some methods involve the use of *duration rules*, which are rules of the form "*if the segment is* X *and it is in the phrase-final position, then legthen* X *by* n msec". These rules can be formalized in terms of *duration models*, which are mathematical expressions prescribing how the various condition factors are to be used in computing the durations of segments. We would use exploratory data analysis, to arrive to models whose predictions show a good fit to durations from a corpus of labeled speech.

2.3.1.7 Sentence Intonation

Infromation such as:

- The syllables in the utterance to be stressed, as computed by the accentuation and the pronunciation module.
- The type of accents to be used, as well as the types of initial and final boundary tones and phrase accents.
- The duration of the segments in the utterance.

Sentence intonation is implemented by the F0 contour of the phrase. However its generation is not straightforward either. It requires formalizing a lot of phonetic or phonological knowledge, either obtained from experts or automatically acquired from data with statistical models.

2.3.2 The DSP Component

Once the text has been transformed into phonemes, and the associated durations and a fundamental frequency contour have been calculated, the system is ready to compute the speech parameters for synthesis.

The operations involved into the DSP module are the computer analogue of dynamically controlling the articulatory muscles and vibratoral frequency of the vocal folds so that the input signal matches the input requirements. As we can understand, the DSP module in order to do this properly, should take into account the articulatory constraints since it has been known for a long time that phonetic trancriptions are more important than stable states for the understanding of speech. There are two possible ways to do this:

- by storing examples of phonetic transitions and co-articulations into a speech segment database, and using them as they are, as ultimate acoustic units in place of phonemes.
- in the form of a series of rules which formally describe the influence of phonemes on one another.

As a result, two main types of TTS synthesizers have been developed from these two strategies correspondigly: *synthesis-by-concatenation* and *synthesis-by-rule*.

2.3.2.1 Rule-based Synthesis

Rule-based synthesizers are mostly in favor with phoneticians and phonologists as they constitute a cognitive and generative approach of the phonation mechanism. Rule-based synthesizers are space efficient, since they eliminate the need to store speech segments and they also make it easier to implement new speaker characteristics for different voices, as well as different phone inventories for new dialects and languages.

These systems are also restrictive regarding the choice of the parametric representation of the speech, since such schemes rely both on our understanding of the relation between the parameters and the acoustic signals they represent and on our ability to compute the dynamics of the parameters as they move from one sound to another. As a result, only articulation parameters and formants have been used in rule-based systems.

Most such systems describe the speech as the dynamic evolution of up to 60 parameters mostly related to *formant* and *anti-formant* frequencies and bandwidths together with glottal waveforms. In order to understand the term *formant frequencies* let us imagine how our vocal tract works when we speak. The vocal tract (the throat from the vocal chords to the lips) has certain major reasonant frequencies. These frequencies change as the configuration of the vocal tract changes, like when we produce different vowel sounds. These reasonant peaks in the vocal tract transfer function (or frequency response) are known as *formants.*

Clearly, the large number of parameters complicate the analysis stage and tends to produce analysis errors. Furthermore, formant frequencies and bandwidths are indeherently difficult to estimate from speech data. The need for intensive trials and errors, in order to cope with the analysis errors make them time-consuming systems to develop. Nevertheless, the synthesis quality achieved up to now reveals typical buzzyness problems, which originate from the rules themselves: introducing a a high degree of naturalness is theoretically possible, but the rules to do so are still to be discovered. Rule-based synthesizers remain however, a potentially powerful approach to speech synthesis.

They allow for instance, to study speaker-dependent voice features so that switching from one synthetic voice to another can be achieved with the help of specialised rules in the rule database. Following the same idea, synthesis-by-rule seems to be a natural way of handling the articulatory aspects of changes in speaking styles (as opposed to their prosodic counterpart which can be accounted for by the concatenation-based synthesizers)

A schematic depiction of the DSP module of a rule-based synthesizer is shown in the next figure:

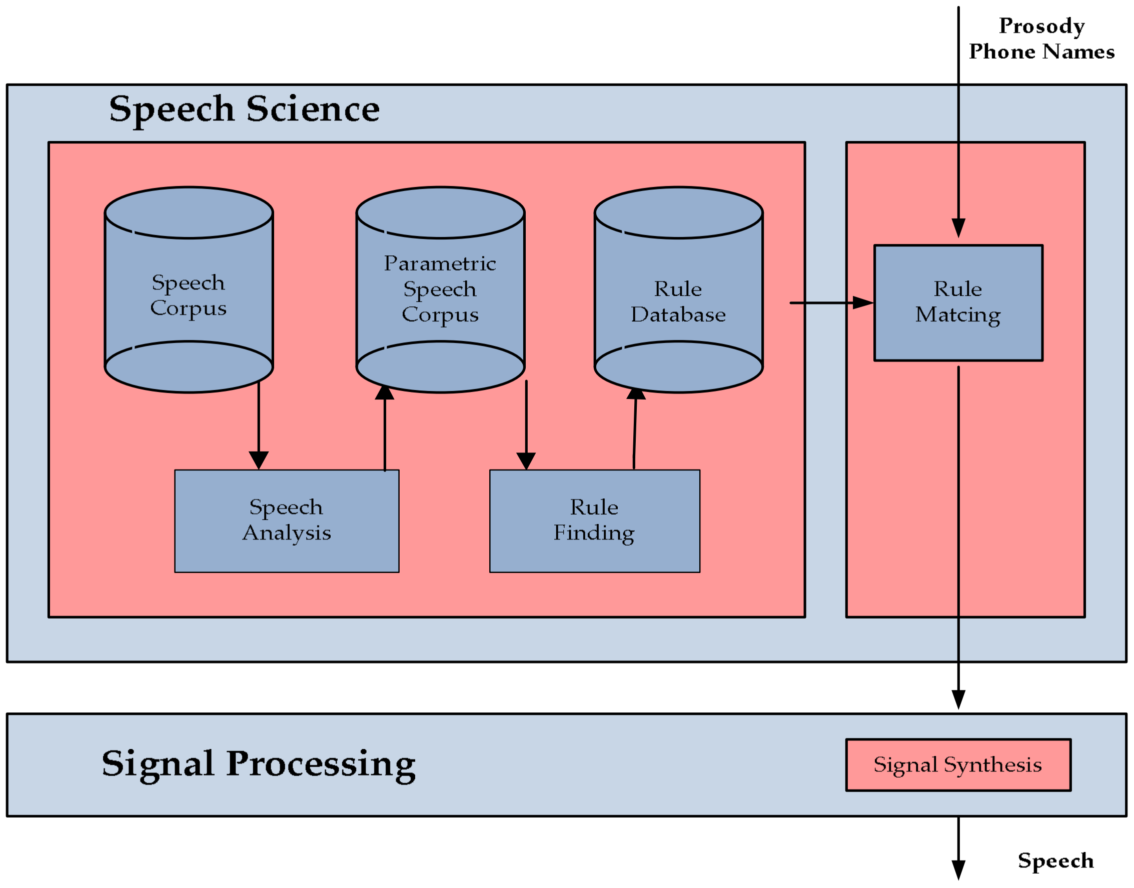


Figure 4. Block diagam of the DSP Module of a Rule-based Synthesizer.

2.3.2.2 Synthesis by Concatenation

As opposed to rule-based ones, *concatenative synthesizers* possess a very limited knowledge of the data they handle: most of it, is embedded to the segments to be chained up. If we study the block diagram of a *concatenative synthesizer*, we can realize that all the operations which could be used indifferently in the context of a music synthesizer, have been grouped into a *sound processing block*, contrary to the upper *speech processing block* whose design requires at least some understanding of phonetics. In order to clarify that, the following figure shows the block diagram of a *concatenative synthesizer:*

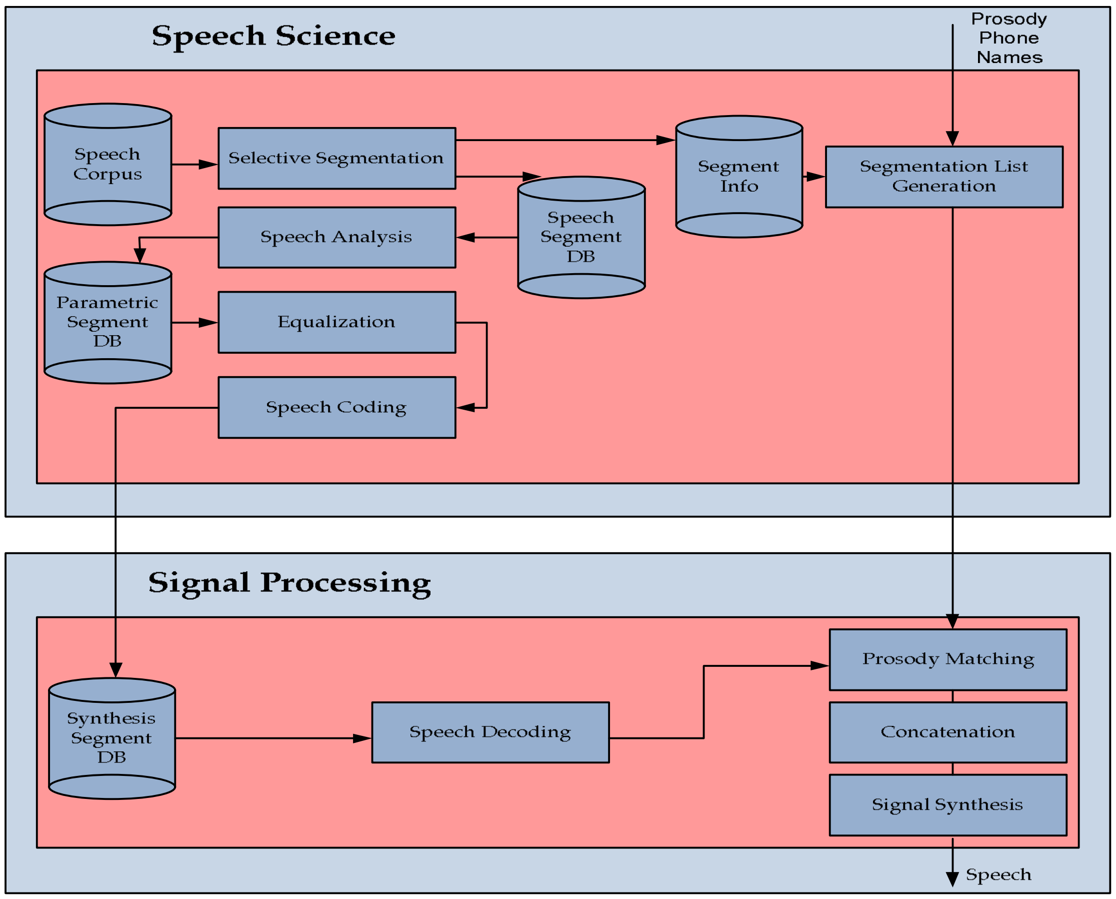


Figure 5. Block Diagram of the DSP Module of a Concatenative Synthesizer.

Database preparation

The stages that have to be fulfilled before the synthesizer produces its first utterance is the following:

1. Segments are chosen so as to minimize future concatenation problems. A combination of diphones, half-syllables or triphones are frequently chosen as speech units since they involve most of the transitions and coarticulations while requiring an affordable amount of memory.
2. When a complete list of segments has emerged, a corresponding list of words is carefully completed, in such a way that each segment appears at least once. Unfavorable positions like unstressed syllables or in strongly reduced contexts, are excluded.
3. A corpus is digitally recorded and stored and the elected segments are spotted, either manually with the help of signal visualization tools, or automatically thanks to segmentation algorithms the decisions of which are checked and corrected interactively.
4. A segment database centralizes the results in the form of the segment names, waveforms, durations and internal sub-splittings. In the case of diphones for example, the position of the border between phones should be stored, so as to be able to modify the duration of the one-half phone without affecting the length of the other one.
5. Segments are given a parametric form, in the form of a temporal sequence of vectors of parameters collected at the output of a *speech analyzer* and stored in a parametric segment database. The advantages of using a speech model originates in the fact that
    - Well chosen speech models allow data size reduction, an advantage which is hardly negligible in the context of concatenation based synthesis given the amount of data to be stored. Consequently, a parametric speech coder often follows the analyzer.

- A number of models explicitly separate the contributions of the source and the vocal tract, an operation that remains helpful for pre-synthesis operations such as prosody matching and segment concatenation.

Indeed, the actual task of the synthesizer is to produce, in real-time, an adequate sequence of concatenated segments, extracted from its parametric segment database. The prosody of these segments has been adjusted from their stored value, i.e the intonation and the duration they appeared within the original speech corpus, to the modification of the pitch, duration and spectral envelope. As a result, the respective parts played by the prosody matching and segments concatenation modules are considerably alleviated when input segments are presented in a form that allows easy

Since segments to be chained up have generally been extracted from different words they often present amplitude and timbre mismatches. Even in the case of stationary vocalic sounds, for instance, a rough sequencing of parameters typically leads to audible discontinuities. These can be coped with during the constitution of the synthesis segment database, thanks to an *equalization* in which related endings of segments are imposed similar amplitude spectra, the difference being distributed on their neighborhood. However, this operation is restricted to amplitude parameters: the equalization stage smoothly modifies the energy levels at the beginning and at the end of the segments, in such a way as to eliminate amplitudee mismatches. In contrast, timbre conflicts are coped during the run-time, by *smoothing* individual couples of segments when necessary rather than equalizing them once and for all, so that some of the phonetic variability naturally introduced by co-articulation is still maintained. In practice, amplitude equalization can be done either in the beginning or in the end of the speech analysis.

6. Once the parametric segment database is completed, the synthesis itself can begin.

2.4 Speech Synthesis

A sequence of segments is first deduced from the phonemic input of the synthesizer in a block termed as *segment list generation* which interfaces the NLP and DSP modules. Once prosodic events have been correctly assigned to individual segments, *the prosody matching module* queries the synthesis segment database for the actual parameters, adequately encoded, of the elementary sounds to be used and adapts them one by one to the required prosody. The segment concatenation block is then in charge of dynamically matching segments to one another, by smoothing discontinuities. Here again, an adequate modeling of speech is highly profitable, provided simple interpolation schemes performed on its parameters approximately correspond to smooth acoustical transitions between sounds. The resulting steam of parameters is finally presented at the input of a synthesis block, the exact counterpart of the analysis one. Its task is to produce speech.

2.5 Segmental quality

The factors that determine the efficiency of high quality speech synthesizers are the following:

1. The types of the segments chosen. Segments should obviously exhibit some basic properties:

    - They should account for as many co-articulatory effects as possible.
    - Given the restricted smoothing capabilities of the concatenation block, they should be easily connectable.
    - Their number and length should be kept as small as possible.
    - On the other hand, longer units decrease the density of concatenation points, therefore providing better speech quality. Similarly, an obvious way of

accounting for articulatory phenomena is to provide many variants for each phoneme. This is clearly in contradiction with the limited memory constraint. Some trade-off is necessary. Diphones are often chosen. They are not too numerous and they incorporate most of phonetic transitions. For example, there are about 1050 diphones in Greek including lots of phoneme sequences that are only encountered at word boundaries. No wonder then why they have been extensively used. They imply however, a high density of concatenation points, which reinforces the importance of an efficient concatenation algorithm. Besides, they can only partially account for the many co-articulatory effects of a spoken language, since they often affect a whole phone rather than just its right or left halves independently. Such effects are specially patent when somewhat transient phones, such as liquids or semi-vowels have to be connected to each other. Hence, the use of larger units as well, such as triphones.

2. The model of speech signal to which the analysis and synthesis algorithms refer.

The models used in the context of concatenative synthesis can be roughly classified into two groups, depending on their relationship with the actual phonation process. *Production models*, provide mathematical substitutes for the part respectively played by vocal folds, nasal and vocal tracts and by the lips radiation. Their most represantive members are the Linear Prediction Coding (LPC) synthesizers and the *formant* synthesizers. On the contrary, *phenomenological models* intetionally discard any reference to the human production mechanism. Among these pure digital signal processing tools, spectral and time-domain approaches are increasingly encountered in TTS systems. Three such leading models exist: the hybrid Harmonic/Stochastic of [Abrantes] and the Time-Domain Pitch-Synchronous-OverLap-Add [TD-PSOLA] of [Moulines and Capentier] and the MBROLA algorithm [Dutoit]. The latter is a time-domain algorithm: it virtually uses no speech explicit speech model. It exhibits very interesting practical features: a very high speech quality combined with a very low computational cost (7 operations per sample on the average). The hybrid

Harmonic/Stochastic model is more powerful than the TD-PSOLA one, but it is also about ten times more computationally intensive. PSOLA synthesizers are widely used in the speech synthesis community. Nevertheless, the MBROLA algorithm is the best because provides a time-domain algorithm which exhibits the very efficient smoothing capabilities of the H/S model as well as its very high compression ratios while keeping the computational complexity of PSOLA.

Chapter 3

RESIDUAL-EXCITED LINEAR PREDICTION CODING (LPC) DIPHONE SYNTHESIS

3.1 Introduction

The basic idea behind diphone synthesis is to list all possible phone-to-phone transitions in a language. This makes the incorrect but practical and simplifying assumption that co-articulatory never go over more than two phones. The exact definition of *phone* is generally nontrivial, and what a standard phoneset should be is not uncontroversial (various allophonic variations must also be included). Unlike generalised unit-selection, where multiple occurences of phones may exist with various distinguishing features, in a diphone database only one occurrence of each diphone is recorded. This makes selection much easier.

In general, the number of diphones in a language is the square of the number of phones. However, in natural human languages, there are phonotactic constraints, some phone-phone pairs, even whole classes of phone-phone combinations, may not occur at all. These gaps are common in the world's languages. The exact definition of *never exists* is also problematic. Humans can often generate those so-called non-existant diphones if they try and one must always think about phone pairs that cross over word boundaries as well, but even then certain combinations cannot exist; for example the diphone λ-ϱ in the Greek language is impossible.

Diphone synthesis, and generally any concatenative synthesis method, makes an absolutely fixed choice about which units exist, and in circumstances when something else is required a mapping is necessary. When humans are given a context where an unusual phone is desired, for example in a foreign language, they will try to pronounce it although does not belong to their phonetic vocabulary. The articulatory system is flexible enough to produce (or try to produce) unfamiliar phones, as we all share the same underlynig physical phonetic structure. Concatenative synthesizers however, have a fixed inventory, and cannot reasonably be made to produce anything above their pre-defined vocabulary. That is the advantage of formant and articulatory synthesizers. This a basic trade-off, concatenative synthesizers typically produce much more natural synthesis than the formant synthesizers but at the cost of being only able to produce those phones defined within their inventory.

Since we wish to build a new voice, we must include any premisible or not phone combination in the Greek language by doing some mapping typically at the lexical level. As a result, we ensure that all the required diphones lie within the recorded inventory. In addition to the base phones, various allophonic variations may be considered. For example, the pronunciation of /ϰ/ is different in the word ϰαιϱός and in the word ϰαταιγίδα. Ideally, all possible variations must be included in the diphone list, but the more variations you include, the larger the diphone set will be. This will affect recording time, labeling time and ultimately the database size.

Although generalised unit-selection synthesis can produce *much* better synthesis than diphone synthesis techniques, using bigger units makes selecting the appropriate ones much more difficult. With a harder selection task, it is more likely that mistakes will be made, which in unit-selection can give some selections which are much, much worse than diphones even though other examples are better.

3.2 Definition of the Greek Phoneset.

The first and most basic task for creating our Greek voice is the definition of its phoneset. A phoneset is a set of phones which may be further defined in terms of features, such as vowels/consonants, place of articulation for consonants, type of vowel etc. The set of features and their values must be defined with the phoneset. The notion of phoneset is important to a number of different subsystems in Festival [2]. Festival also supports multiple phonesets simultaneously and allows mapping between sets when necessary. The lexicons, letter-to-sound rules, waveform synthesizer etc, all require the definition of a phoneset before they will operate. A phoneset definition has the following form:

```
(defPhoneSet
    NAME
   FEATUREDEFS
   PHONEDEFS)
```

The NAME is any unique symbol for example our phoneset is named tuc_gr, the FEAUREDEFS is a list of phone features with their values and the PHONEDEFS is the list of phones with their feature values. Our phoneset is a variation of the SAMPA Greek Phoneset. The following table shows our phoneset

```
(defPhoneSet
  tuc_gr

  (
   (vc + -)
   (vlng s l b d 0)
   (vheight 1 2 3 0)
   (vfront 1 2 3 0)
   (vrnd + - 0)
   (ctype s f a n l 0)
   (cplace l a p b d v 0)
   (cvox + - 0)
   )

  (
   (pau  - 0 0 0 - 0 0 -)  ;; silence ...
   (A   +   b   1   1   -   0   0   0)
   (E   +   s   2   2   +   0   0   0)
   (i   +   b   3   1   +   0   0   0)
   (o   +   s   2   2   +   0   0   0)
   (u   +   d   3   3   +   0   0   0)
   (j   +   l   3   1   +   0   0   0)
   (J   +   l   3   1   +   0   0   0)
   (ly  +   l   3   1   +   0   0   0)
   (N   +   l   3   1   +   0   0   0)
   (D   -   0   0   0   0   f   d   +)
   (G   -   0   0   0   0   f   v   +)
   (T   -   0   0   0   0   f   d   -)
   (C   -   0   0   0   0   f   p   -)
   (K   -   0   0   0   0   f   p   -)
   (R   -   0   0   0   0   l   0   0)
   (b   -   0   0   0   0   s   l   +)
   (d   -   0   0   0   0   s   a   +)
   (f   -   0   0   0   0   f   b   -)
   (g   -   0   0   0   0   s   v   +)
   (q   -   0   0   0   0   s   0   0)
   (k   -   0   0   0   0   s   v   -)
   (l   -   0   0   0   0   l   a   +)
   (m   -   0   0   0   0   n   l   +)
   (n   -   0   0   0   0   n   a   +)
   (p   -   0   0   0   0   s   l   -)
   (r   -   0   0   0   0   l   0   +)
   (s   -   0   0   0   0   a   d   -)
   (t   -   0   0   0   0   s   a   -)
   (v   -   0   0   0   0   f   b   +)
   (x   -   0   0   0   0   f   v   -)
   (z   -   0   0   0   0   f   a   +)
   (S   -   0   0   0   0   s   0   +)
   (Z   -   0   0   0   0   s   0   +)
   (X   -   0   0   0   0   s   0   +)
   (Y   -   0   0   0   0   s   0   +)
```

Table 4. Greek Phoneset in Festival.

As we can see, the features of phones supported with their variable names and values are

Feature Name	Variable Name	Variable Value
Vowel/Consonant	vc	+/-
Vowel Length	vlng	short/long/bi/dipthong
Vowel Height	vheight	high/mid/low
Vowel Frontness	vfront	front/mid/back
Lip Rounding	vrnd	+/-/0
Consonant Type	ctype	stop/fricative/affricative/nasal/liquid
Consonant Articulation	cplace	labial/alveolar/palatal/labio-dental/dental/velar
Consonant Voicing	cvox	+/-/0

Table 5. Phone Feature Names and Values.

The phoneset also includes a definition for the silence phones. In addition to the definition of the set the silence phone *(pau)* must be identified to the system. There may be many silence phones (e.g breathe, start silence etc) in any phoneset definition. However the first phone in this set is treated special and is canonical silence. Among other things, it is the phone that is inserted by the pause prediction module. In the next table we will show the phones we used in our voice, the SAMPA equivalent, an example per phone and their phonetic transcription.

TUC_GR Phones	SAMPA Phones	Example	Hellenic Transcription	English Transcription
A	a	AnEmi	άνεμοι	wind
E	e	EGEo	Αιγαίο	Egeo
i	i	ElpiDa	ελπίδα	hope
o	o	oros	όρος	clause
u	u	urAnos	ουρανός	sky
J	-	trJAdA	τριάντα	thirty
ly	L	CillyaDes	χιλιάδες	thousand
D	D	DoDEkAnisA	δωδεκάνησα	dodekanese
G	G	GenikA	γενικά	generally
th	T	thErmokRAsiA	θερμοκρασία	temperature
C	C	CJonJA	χιόνια	snow
b	b	bofor	μποφόρ	bofor
d	d	dinomE	ντύνομαι	dress

TUC_GR Phones	SAMPA Phones	Example	Hellenic Transcription	English Transcription
g	g	gREmizo	γκρεμίζω	destruct
f	f	fos	φως	light
q	gj	AgEliA	αγγελία	announcement
k	k	kirios	κυρίως	mostly
l	l	liGEs	λίγες	few
m	m	mikRi	μικρή	small
n	n	notJA	νότια	south
p	p	ptosi	πτώση	fall
r	r	ropi	ροπή	inclination
s	s	stadiAkA	σταδιακά	gradually
t	t	stREfo	στρέφω	turn
v	v	vorJA	βόρεια	north
x	x	xArtis	χάρτης	map
z	z	zodAnos	ζωντανός	alive
ts	ts	tsalakono	τσαλακώνω	crumple
tz	dz	tzAmAriA	τζαμαρία	glass
ks	ks	ksAnA	ξανά	again
Y	ps	YAxno	ψάχνω	look for
pau	_		(παύση)	

Table 6. Greek Phoneset with SAMPA equivalent, examples and Hellenic and English transcription.

In the Greek language, as in any language there are some certain phonotactic constraints which do not allow any combination of phones and also create the Greek allophones. These are the following:

1. /C/ must be followed by /E/ or /i/.
2. /x/ must be followed by /A/, /o/, or /u/ or consonant.
3. /q/ must not follow silence.

3.3 Definition of the diphone-list

The second step for the creation of our voice is to define the Greek diphone-list. The basic programming language in Festival is C++ but beyond C++ Festival also supports a Scheme Interpreter (SIOD) which offers a basic small LISP interpreter suitable for embedding in applications such as Festival, as scripting language. So, every part of code we will show will be in Scheme. The basic idea in creating the diphone-list is to define classes of diphones, e.g vowel-vowel, consonant-vowel, vowel-consonant and consonant, then define a carrier sentence for these and list the cases. For example, to generate all vowel-vowel diphones we define the carrier

(set! vv-carrier '((pau pau)))

and then the function that generates all the vowel-vowel transitions

```
(define (list-vvs)
  (apply
   append
   (mapcar
    (lambda (v1)
      (mapcar
       (lambda (v2)
         (list
          (string-append v1 "-" v2)
          (append (car vv-carrier) (list v1 v2) (car (cdr vv-
carrier)))))
       vowels))
    vowels)))
```

Table 7. Function generating the vowel-vowel diphones.

For those who are not used to read Lisp the above function lists all vowel-vowel combinations. The algorithm is really simple:

```
for v1 in vowels
  for v2 in vowels
     print pau $v1 $v2 pau
```

Although it is very easy to just list all contexts and pairs, there are some constraints listed in the previous paragraph which should take into consideration. Those seven limitations diminish the number of diphones. *Theoretically* the number of diphones is

$$TheoreticNumberOfDiphones = NumberOfPhones^2 \Rightarrow$$
$$TheoreticNumberOfDiphones = 35^2 \Rightarrow$$
$$TheoreticNumberOfDiphones = 1225.$$

Nevertheless, the *real* number of Greek diphones due to those limitations is

$$ActualNumberOfDiphones = 972.$$

The generated diphone-list has a particular format. Each line contains a file_id, a carrier sentence and a diphone name (or list of names if more than one diphone is extracted). A part of the Greek diphone-list is shown below:

```
( gr_0001 "pau A A pau" ("A-A") )
( gr_0055 "pau s E pau" ("s-E") )
( gr_0287 "pau l ly pau" ("l-ly") )
( gr_0343 "pau s - T pau" ("s-T") )
( gr_0763 "pau f - R pau" ("f-R") )
```

3.4 Corpus Design

After the creation of the Greek diphone-list the next task is to construct the corpus of our synthesizer. The success of our speech synthesis shema, crucially depends on an effective corpus design, such that instances of all necessary diphones can be found in matching prosodic context.

The domain that the application is built to cover is limited, but still quite large when compared with other closed-vocabulary tasks, such as the synthesis of telephone numbers. Its difficulty lies in the fact, that it involves the synthesis of whole sentences, rather than certain words within a sentence.

3.4.1 Selection of sentences

The creation of the corpus, the set of sentences to be recorded and later segmented into the diphones they contain, is crucial to the performance of our application. After all, the fragments extracted from this process are the basic units used to synthesize the output of our TTS system. It is the efficiency and the quality of the Corpus Creation procedure that largely defines the success of our application.

The sentences that were finally selected were chosen from a set of trancribed weather forecast reports, covering a week of each month during the period October 1999-September 2000 and January 2007- October 2007. These reports were provided by the National Meteorological Agency (EMY). For these data, these following information is given:

Total Number of sentences	4,310
Total Number of word instances	64,005
Total Number of Diphone instances	256,089
Total Number of Diphones	972

Table 8. Statistical analysis of the Original data set.

3.4.2 Corpus Characteristics

The following characteristics were to be met by the corpus:

1. The corpus should contain an adequate number of sentences. Totally, we had to make ten hours of recordings which was translated to 4310 sentences.
2. The corpus should contain all the words we want for our application. We need at least one instance of every word that be found in our application.
3. The corpus should contain these words in as many contexts as possible. Multiple recorded instances of commonly used words should be available to the application, in order to incorporate into the corpus as many prosodic features as possible.

3.4.3. Word Selection

In the original data set many words appeared a lot of times. For example, the conjuction ‘και’ appeared 286 times. Of course, in the final corpus should appear only once. On the other hand, the words ‘ἀνεμος’ and ‘ἀνεμοι’ may have the same context but different orthographic representation and as a result, are considered different words. In order to cope with this problem, we wrote a small program which took as input each sentence. Firstly, we created a hash table. Then, we took each word in the sentence and compared it with the entries in the hash table. If there wasn’t in the hash table we put the word in the hash table and also stored it in a separate file which we would use as the lexicon of our application. On the other hand, if there was in the hash table, that meant that already existed in the lexicon so we continued to the next word in the sentence.

3.4.4 The final corpus

As we have already stated, the original data set contained almost 4,300 sentences with 64,000 words and approximately 256,000 diphones. The selection procedure we described produced a corpus of words that we used in our application. The statistical analysis is shown in the next table:

	Original Data Set	Final Corpus
Total Words	64,005	2,439
Distinct Words	1,165	1,165
Diphone Instances	256,089	972

Table 9. Statistical analysis of the final Corpus.

3.5 The Recording Phase

Since our corpus was ready, we started the recording of our sentences. The recordings were made by myself. Of course, I am not a voice talent but since we needed the recordings only for my thesis, I did them. I read the sentences well articulated but as naturally as possible. The sentences were recorded in home enviroment during night hours with low levels of external noise and then were digitally stored using PCM coding at 16,000 Hz with 16bit/sample. The corpus had a size of 134,637,589 bytes equal to 35,571 sec of speech. Considering that, the data file would be used also by other modules in Festival had to be in specific format. Each line contains a file id, the sentence that was recorded and the list of diphones in the sentence. The file id is used in the filename for the waveform, label file and any others parameters file associated with the recording. For example:

(utt_001 “ i AnEmi TA pnEun vorJi AsTEnis” (“i-A” “A-n” “n-E” “E-m” “m-i” “i-T” “T-A” “A-p” “p-n” “E-u” “u-n” “n-v” “v-o” “o-r” “r-J” “J-i” “A-s” “s-T” “T-E” “E-n” “n-i” “i-s”)

As you will notice, normally there are two instances of the diphones “i-A” and “n-E” in the sentence. Nevertheless, in the same sentence there must be only one instance of the same diphone so we include only one of them.

3.6 Labeling

The next step was to label the recorded speech. In the early years of concatenative speech synthesis, every recorded prompt had to be hand labeled. Although a significant task, very skilled and mind bogglingly tedious it was a feasible task to attempt when databases were relative and the time to build voice was measured in years. With the increase in size of database and the demand for much faster turnaround we have moved away from hand labeling to automatic labeling.

In labeling recorded prompts we rely much on the work that has been done in the field of Speech Recognition. For synthesis however, we have different goals. In ASR *(Automatic Speech Recognition)*, we are trying to find the most likely set of phones that are in given acoustic observation. In synthesis labeling, we know the sequence of phones spoken and wish to find out where these phones are in the signal. We care very deeply about the boundaries of segments, while ASR can achieve adequate performance by only consider itself with the centers. There are also some other distinctions from the ASR task, since in synthesis labeling we only have one speaker, which simplifies the recognition a lot, and we are very concerned about the prosody and the spectral variation of speech. There are two basic techniques which are used for labeling recorded prompts easch one with its own advantages and limitations.

The first technique uses *dynamic time warping (DTW)* alignment techniques to find the phone boundaries in a recorded prompt by align it against a synthesized utterance where the phone boundaries are known. This is computationally easier than the other technique and is recommended for small databases which do not have full phonetic coverage.

The second technique uses *Baum-Welch training* to build complete ASR acoustic models from the database. This takes sometime, but if the database is phonetically balanced, as should be the case in databases designed for speech synthesis voices, can work very well. Also this technique can work really well in languages that do not yet have a synthesizer, hence making the *dynamic warping technique* hard without cross-language phone mapping techniques.

Firstly, we tried the *DTW* technique. However, as there is no Greek diphone LPC voice in Festival we had to map the Greek phones with the English phones of the English voice installed in Festival. The result was really bad. So, we used the second technique. For this purpose we used the *Hidden Markov Toolkit (HTK)* . HTK is a toolkit for building *Hidden Markov Models (HMMS)*. HMMS can be used to model any time series and the core of HTK is similarly general-purpose.

Speech recognition systems generally assume that the speech signal is a realisation of some message encoded as a sequence of one or more symbols. To affect the reverse operation of recognising the underlying symbol sequence given a spoken utterance, the continuous speech waveform is first converted to a squence of equally spaced discrete parameter vectors. This sequence of parameter vectors is assumed to form an exact representation of the speech waveform on the basis that for the duration covered by a single vector, the speech waveform can be considered as being stationary. Although this is not strictly true, it is a reasonable approximation. Typical parametric representations are smoothed spectra or Linear Pediction Coefficients plus various other representations derived from them.

The role of the recogniser is to extract from the speech vectors the underlying symbol sequences. Two problems make this very difficult. Firstly, the mapping from symbols to speech is not one-to-one since different underlying symbols can give rise to similar speech sounds. Furthermore, there are large variations in the realised speech waveform due to speaker variability, mood, enviroment etc. Secondly the boundaries between symbols cannot be identified explicitly from the speech waveform.

In HMM-based speech recognition, it is assumed that the sequence of observed speech vectors corresponding to each word is generated by a Markov model as shown in the next figure. A Markov Model is a finite state machine which changes state once every time unit and each time *t* that a state *j* is entered, a speech vector O_t is generated from the probability density $b_j(O_t)$. Furthermore, the transition from state *i* to state *j* is also probabilistic and is governed by the discrete probability a_{ij}. Figure 6 shows an example of this process where the six state model moves through the stage sequence X=1,2,3,4,5,6 in order to generate the sequence O_1 to O_6. The joint probability that O is generated by the model M moving through the state sequences X is calculated simply as the product of the transition probabilities and the output probabilities.

So for the state sequence X in Figure 6 we have

$$P(O, X \mid M) = a_{12} b_2(O_1) a_{22} b_2(O_2) a_{23} b_3(O_3)...$$

However, in practice only the oservation sequence O is known and the underlying state sequence X is hidden. This is why it is called *Hidden Markov Model.*

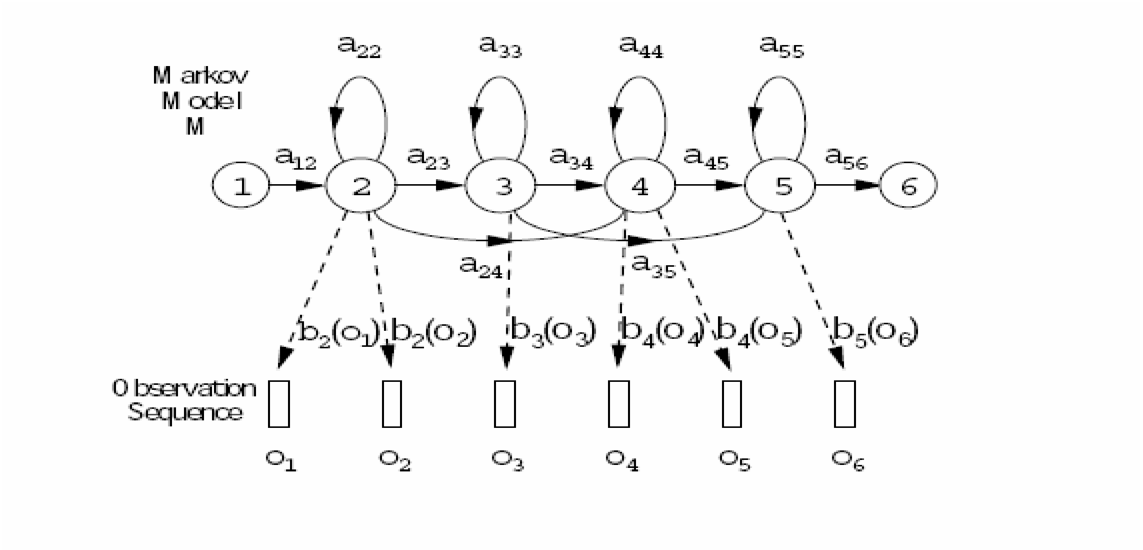


Figure 6. The Markov Generation Model.

Given that X is unknown, the required likelihood is computed by summing over all possible state sequences $X = x(1), x(2), x(3), \ldots x(T)$, that is

$$P(O \mid M) = \sum_X a_{x(0)x(1)} \prod_{t=1}^{T} b_{x(t)}(O_t) a_{x(t)x(t+1)}$$

where x(0) is constrained to be the model entry state and x(T+1) is constrained to be the model exit state.

In order to do our labeling, we used a single speech recognition tool of HTK called HVite. HVite uses the token passing algorithm to perform Viterbi-based speech recognition. HVite takes as input a network describing the allowable word sequences, a dictionary defining how each word is pronounced and a set of HMM models. We created a dictionary with all the words that used in our recordings with their pronunciation and used an existing set of full acoustic HMM models for the Greek language built in the Techical University of Crete Telecom Laboratory. HVite operates by converting the word network to a phone network and then attaching the appropriate HMM definition to each phone instance. Recognition can then be performed on either a list of stored speech files or on direct audio input. The two commands that we used to do the labeling are:

1. HVite -D -H models -S wav.list -y lab -C config -i result.mlf vocab-60k1-w2p1pass tri.list_clustered

2. HVite -l '*' -T 0040 -C config -a -m -H models -i align.mlf -I result.mlf vocab-60k1-w2p1pass tri.list_clustered -S wav.list

where

wav.list: The list with the full paths in the hard drive of our recordings.
models: Set of Greek full acoustic HMM models.
config: configuration file that set particular values in some command parameters.
vocab-60k1-w2p1pass: dictionary with the pronunciation of all the recorded words.
tri.list_clustered: the list of all triphones in the Greek language based on our phoneset.
result.mlf: the output file that contains the aligned words in every recording.
align.mlf: the output file that contains the aligned phones in every recording.

3.7 Text Analysis

In this section, we discuss some of the basic problems in analyzing text when trying to convert it to speech. A crucial stage in text processing is the initial tokenization of text. A *token* in Festival is an atom separated with whitespace from a text file (or string). After we defined punctuation for

the Greek language, characters matching that punctuation are removed from the beginning and end of a token and held as features of the token. The default list of characters to be treated as white space is defined as

```
(defvar token.whitespace " \t\n\r")
```

while the default set of punctuation characters is

```
(defvar token.punctuation "\"'`.,:;!?(){}[]")
(defvar token.prepunctuation "\"'`({[")
```

Tokens are further analysed into lists of words. A word is an atom that can be given a pronunciation by the lexicon (or letter to sound rules). A token may give rise to a number of words or none at all.

For example the basic tokens in the sentence

“Οι άνεμοι θα πνέουν ασθενείς”

would give a word relation of

οι άνεμοι θα πνέουν ασθενείς

Due to the fact that, the relationships between tokens and word in some cases are complex, a separate function should be specified for translating tokens to words. This is designed to deal with things like numbers, dates, addresses and other non-obvious pronunciations of tokens such as zero or others.

This function is set in our voice selection function as the function for token analysis

(set! token_to_words_tuc_gr_chris_token_to_words)

This function is added to to deal with all tokens that are not in our lexicon, cannot be treated by letter-to-sound rules, or are ambiguous in some way and require context to resolve.

For example suppose we wish to simply treat all tokens consisting of strings of digits to be pronounced as a string of digits (rather than numbers). We would add something like in the following table

```
(set! tuc_gr_chris_digit_names
   '((0 "μηδέν")
     (1 "ένα")
     (2 "δύο")
     (3 "τρία")
     (4 "τέσσερα")
     (5 "πέντε")
     (6 "έξι")
     (7 "εφτά")
     (8 "οχτώ")
     (9 "εννιά")))

(define (tuc_gr_chris_token_to_words token name)

  (cond
   ((string-matches name "[0-9]+") ;; any string of digits
    (mapcar
     (lambda (d)
      (car (cdr (assoc_string d MTLANG_digit_names))))
     (symbolexplode name)))
   (t
    (list name))))
```

Table 10. Token-to-Word Mapping function.

3.8 Segmentation

Once our recordings were labeled, we continued with the segmentation of our recordings. The result of the segmentation was a diphone index. The index identifies which diphone comes from which files and from where. This was automatically built from the label files. The Festival script *make_diph_index.scm* will take the diphone-list, finds the occurrence of each diphone in the label files and builds an index. The index consists of a simple header, followed by a single line for each diphone: the diphone name, the file_id, start time, mid-point(phone boundary) and end time. The times are given in *seconds*. An example of the start of our diphone index file is given below:

```
EST_File index
DataType ascii
NumEntries 7611
EST_Header_End
A-p utt_4311 0.0025 0.005 0.0134
p-o utt_4311 0.0133333 0.03 0.045
o-t utt_4311 0.045 0.06 0.11
t-o utt_4311 0.11 0.21 0.235
o-A utt_4311 0.235 0.26 0.31
o-G utt_4311 0.455 0.5 0.56
G-E utt_4311 0.56 0.62 0.685
E-v utt_4311 0.685 0.75 0.82
v-m utt_4311 0.82 0.89 0.925
m-A utt_4311 0.925 0.96 0.99
```

Table 11. Diphone Index File sample.

3.9 Energy Normalization

Although we tried to read the sentences during the recording phase without alterations in the volume of the voice, quality checks showed that the volume among certain sentences had some fluctuation. In order to overcome this problem, all sentences were subjected to mean energy normalization using the energy normalization tool of Cool Edit Pro v.2.

3.10 Pitchmark Extraction

The current method we use to create our Greek Weather Forecast synthesizer is residual excited LPC synthesis. This technique is pitch synchronous, that is it requires about where pitch periods occur in the acoustic signal. The basic program that we use to extract the pitchmarks is the *make_pm_wave.file* that uses the function ***pitchmark*** which is part of the **Edinburgh Speech Tools** distribution. The Edinburgh Speech Tools Library is library of general speech software, written at the Centre for Speech Technology Research at the University of Edinburgh. It is written in C++ and provides a range of common tasks found in speech processing. One of them is the script for pitchmark extraction. The key line in the script is the:

```
$ESTDIR/bin/pitchmark tmp$$.wav -o pm/$fname.pm -otype est  -min 0.005 -max 0.012 -
fill -def 0.01 -wave_end  -lx_lf 200 -lx_lo 51 -lx_hf 80 -lx_ho 51 -med_o 0
```

This program filters the incoming waveform with a low and high band filter and then uses autocorrelation to find the pitchmark peaks with the min and max specified. Finally, it fills in the unvoiced section with the default pitchmarks. For example, we take the word "τάματα" . The above script properly finds pitchmarks in the tree vowel sections.

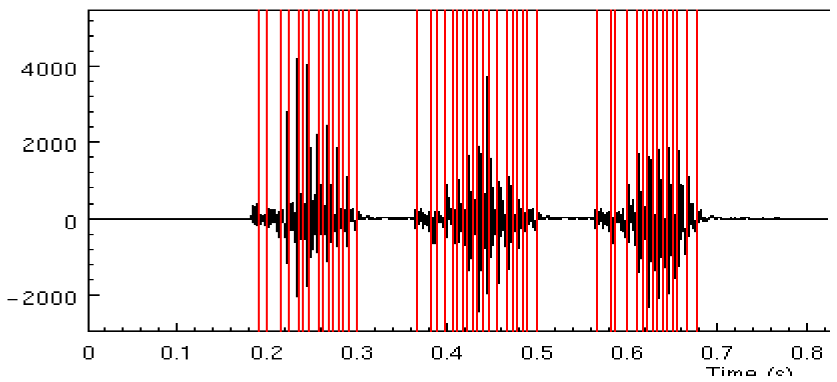


Figure 7. Pitchmarks in waveform signal.

If the high and low pass filter values –lx_hf and –lx_lf are not appropriate for the speaker's pitch range you will get either more or less pitchmark peaks. For example, if we change the high frequency value 200Hz to 60Hz we will get only two pitchmarks in the third vowel.

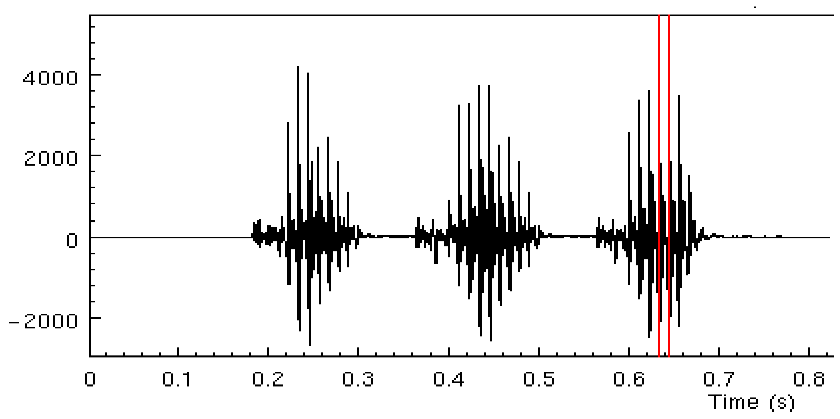


Figure 8. Bad pitchmarks in a waveform signal.

If we zoom in our first example we get the following

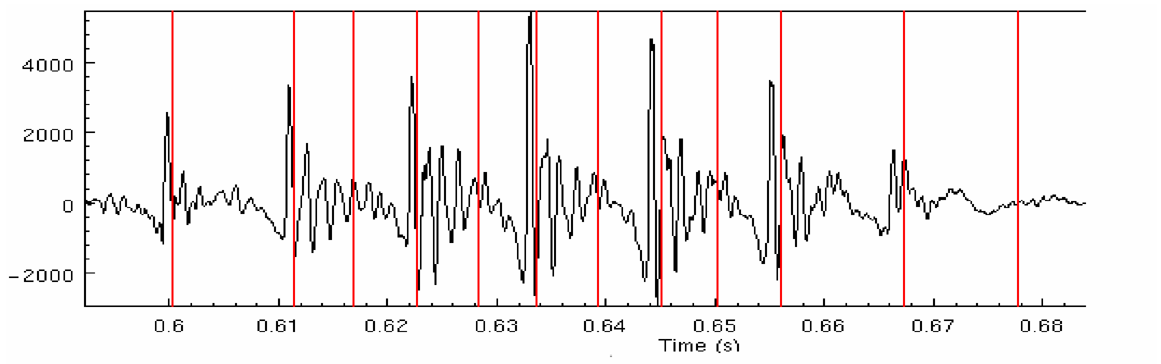


Figure 9. Close-up of pitchmarks in a waveform signal.

The pitch marks should be aligned to the largest (above zero) peak in each pitch period.

Here we can see there are too many pitchmarks (effectively twice as many). The pitchmarks at 0.617, 0.628, 0.639 and 0.650 are extraneous. This means our pitch range is too wide. If we rerun changing the min size, and the low frequency filter

```
$ESTDIR/bin/pitchmark tmp$$.wav -o pm/$fname.pm -otype est -min 0.007 -max 0.012 -
fill -def 0.01 -wave_end -lx_lf 150 -lx_lo 51 -lx_hf 80 -lx_ho 51 -med_o 0
```

We get the following

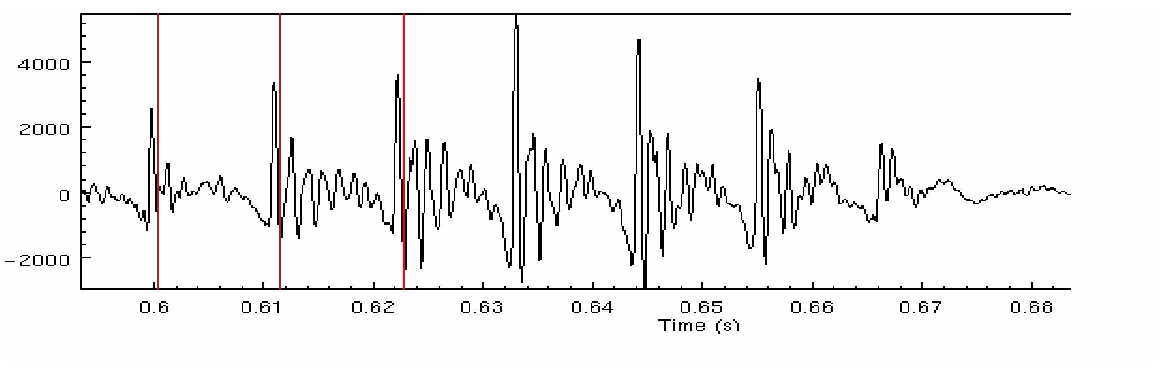


Figure 10. Close-up of pitchmarks in waveform signal (2)

Which is better but it is now missing pitchmarks towards the end of the vowel, at 0.634, .644 and 0.656. Giving more range for the min (0.005) gives slight better results, but still we get bad pitchmarks. The double pitch mark problem can be lessened by not only changing the range but also the amount order of the high and low pass filters (effectively allowing more smoothing). Thus when secondary pitchmarks appear increasing the -lx_lo parameter often helps

```
$ESTDIR/bin/pitchmark tmp$$.wav -o pm/$fname.pm -otype est  -min 0.005 -max 0.012 -
fill -def 0.01 -wave_end  -lx_lf 150 -lx_lo 91 -lx_hf 80 -lx_ho 51 -med_o 0
```

We get the following:

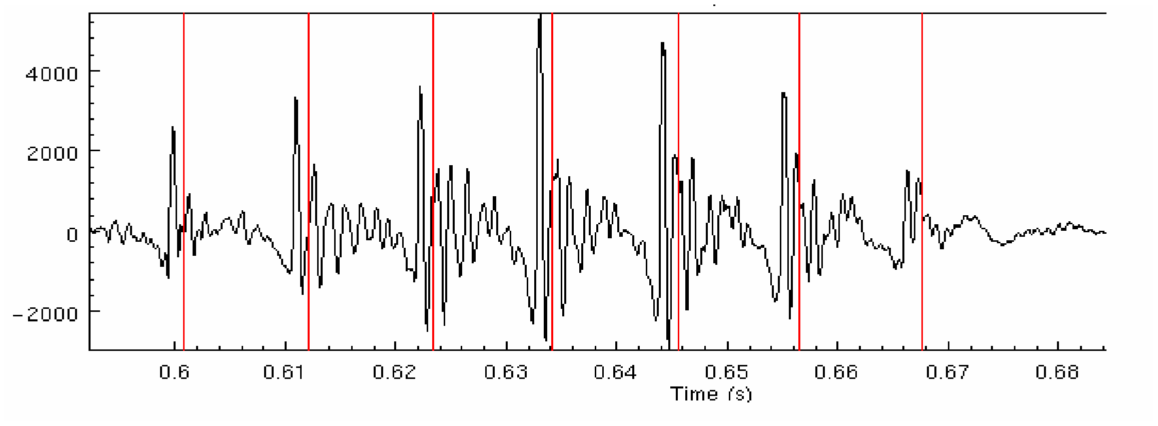


Figure 11. Close-up of pitchmarks in waveform signal (3)

This is satisfactory for this file and generally for the whole speech database of the speaker. As we can understand, correct extraction of the pitchmarks is crucial in order to have very good quality for our synthetic voice.

3.11 Building LPC Parameters

In general, the voice signal can be given by the following two equations:

$$S(z) = E(z) * V(z) * R(z) \quad \text{(Unvoiced Sounds)}$$

$$S(z) = E(z) * G(z) * V(z) * R(z) \quad \text{(Voiced Sounds)}$$

where:

E(z) : Excitation
G(z) : Glottal Pulse Filter
V(z) : Transfer Function – Vocal Tract
R(z) : Radiation

The voice signal can be modeled with three different ways:

1. The Digital Vocal Tract model.
2. Lips Radiation.
3. Excitation.

The Residual-Excited Linear Prediction Coding *(LPC)* method for speech synthesis is based on the Digital Vocal Tract model.

3.11.1 Digital Vocal Tract model

When a person speaks, his or her lungs work like a power supply of the speech production system. The glottis supplies the input with the certain pitch frequency (F_0). The vocal tract, which consists of the pharynx and the mouth and nose cavities, works like a musical instrument to produce sound. To form different vocal tarct shape, the mouth cavity plays the major role. To produce nasal sounds, nasal cavity is often included in the vocal tract. The nasal cavitis connected in parallel with the mouth cavity. The simplified vocal tract is shown in Figure 12.

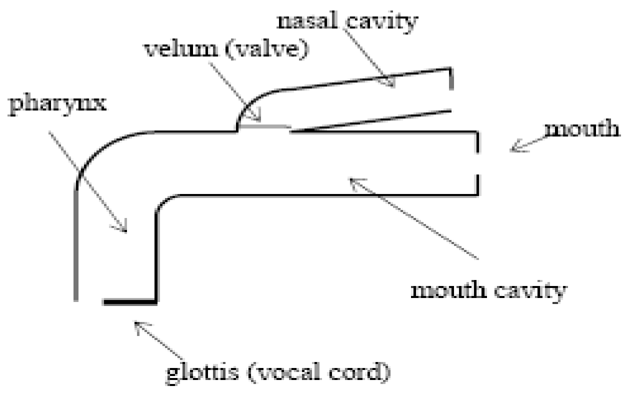
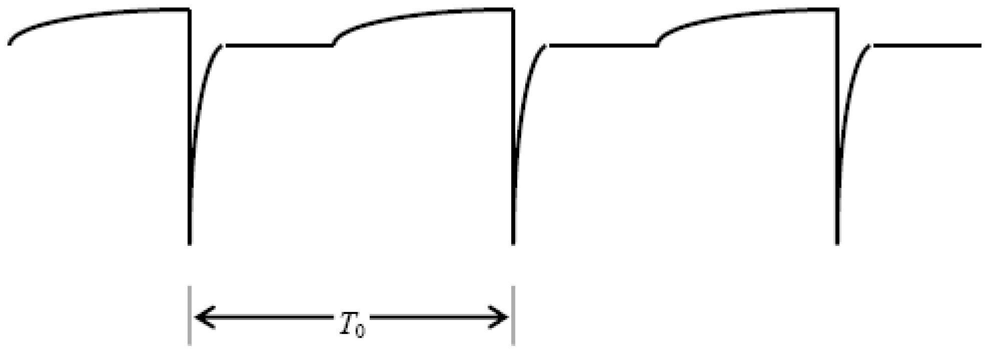
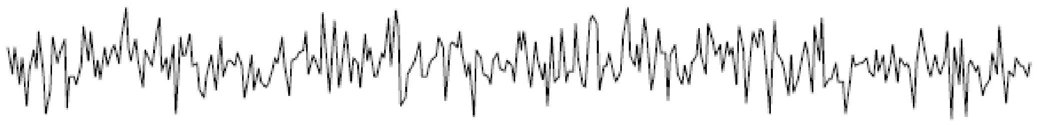


Figure 12. Simplified view of the vocal tract.

The glottal pulse generated by the glottis is used to produce vowels(voiced) sounds, while the noise-like signal is used to produce consonants (unvoiced) sounds. These are shown in Figure 13.



a) glottal pulse excitation for a voiced sound.



b) hiss(white noise) input for an unvoiced input.

Figure 13. Two kinds of input to generate sound (T_0 : pitch period).

Pitch frequency F_0 $(1/T_0)$ varies in different people. A little child's pitch frequency can go as high as 400 Hz. Adult male's pitch frequency is as low as 100 Hz. Adult female's pitch frequency is between 200 HZ and 300 Hz range. This glottal pulse excites a vocal tract cavity and produces a vowel (voiced sound).

In Linear Prediction Analysis the voice signal can be given by the following equation:

$$S(z) = G(z) * V(z) * R(z)$$

where:

G(z) : Glottal Pulse Response = $\dfrac{1}{(1-cz^{-1})^2}, c \approx 1$.

V(z) : Transfer Function – Vocal Tract = $\dfrac{1}{\prod_{k=1}^{p}(1-c_k z^{-1})}$.

R(z) : Radiation = $(1-z^{-1})$.

G : Gain.

The transfer function H(z) of the system can be given by the equation:

$$H(z) = \frac{S(z)}{U(z)} = \frac{G}{1-\sum_{k=1}^{p} a_k z^{-k}}$$

and the voice signal by the equation:

$$\overset{:}{s}(n) = \sum_{k=1}^{p} a_k s(n-k) + Gu(n)$$

The LPC speech synthesis model we used, resembles the vocal tract with a linear, time-variant digital filter which is excited by glottal pulse or hiss in order to generate voiced or unvoiced sounds. This filter is shown in the next figure:

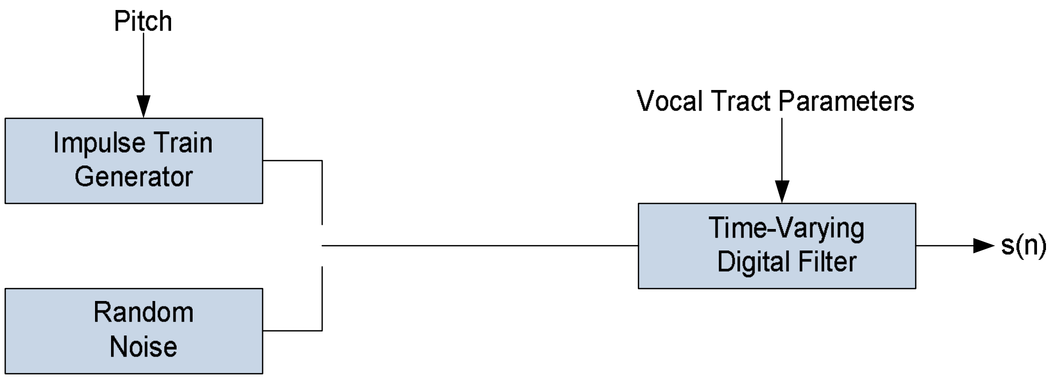
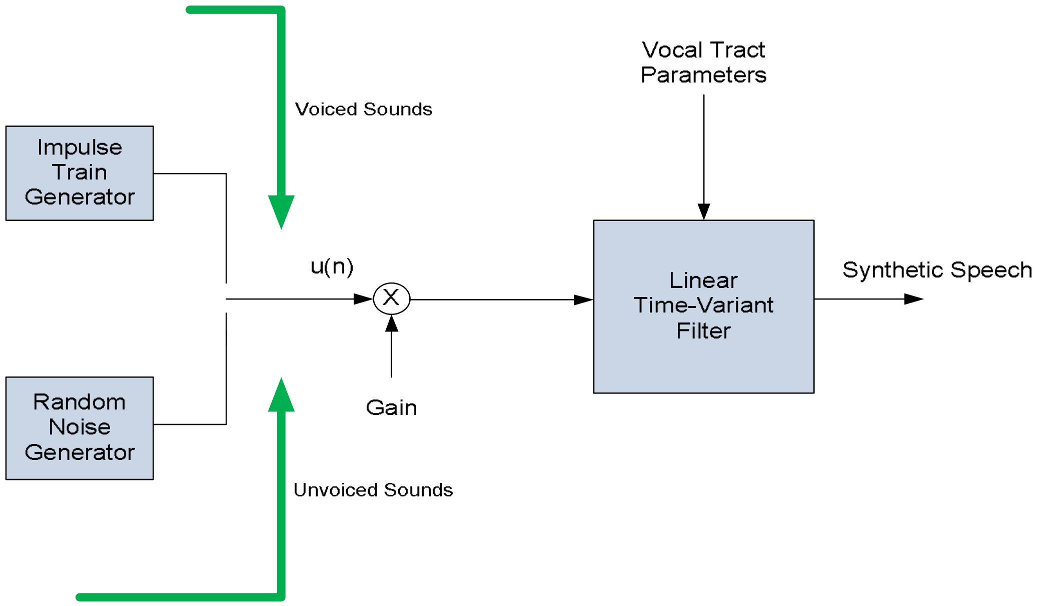


Figure 14 a). Simplified Model of speech production.



b) Detailed LPC Speech Production Model.

3.11.2 LPC Coefficients Calculation.

3.11.2.1 Recursive Levidson Durbin Algorithm.

There are several algorithms in order to calculate the Linear Prediction Model Coefficients. The Levidson Durbin Algorithm is the most efficient solution. It is recursive to the order of the Linear Prediction Model and has complexity linear to the order of the model. It is described in the next table

Step 1 – Model Order *i*=0: We set the initial Total Prediction Error equal to

$$E^{(0)} = R_n(0)$$

Recursion: Increase the order of the model from *i*=1,2,…p and calculate the following amounts

1. $k_i = \dfrac{\left[R_n(i) - \sum_{j=1}^{i-1} a_j^{(i-1)} * R_n(i-j) \right]}{E^{(i-1)}}$

2. $a_i^{(i)} = k_i$
   $a_j^{(i)} = a_j^{(i-1)} - k_i * a_{i-j}^{(i-1)}, 1 \le j \le 1$

3. $E^{(i)} = (1 - k_i^2) * E^{(i-1)}$

Table 12. Levidson-Durbin Recursive Algorithm.

Observations

1. The $a_1^{(i)},...,a_p^{(i)}$ coefficients are the Linear Prediction coefficients for the model with order *i*.

2. During our calculations for the model with order *p* we have calculated the coefficients for every model with order *i*=1,2,…,*p*-1.

3. The coefficients $k_i, i = 1, 2, ...p$ are called reflection coefficients of the vocal tract model.

4. The Levidson-Durbin Algorithm ensures the stability of the model with the a_i coefficients.

Complextiy

1. Reversion of a $p \times p$ matrix: Complexity $O(p^3)$.

2. Levidson-Durbin Algorithm, Calculation of the coefficients with order *p*: Complexity $O(p)$.

3.11.2.2 Gain Calculation

Gain G is selected so as to the voice signal *s(m)* has the same energy with that of the impulse response *h(n)*,

$$h(n) = \sum_{k=1}^{p} a_k h(n-k) + G \cdot \delta(n)$$

namely,

$$\sum_n s^2(n) = \sum_n h^2(n)$$

and as a result gain G is given by

$$G^2 = R(0) - \sum_{k=1}^{p} a_k \cdot R(k)$$

3.11.2.3 Some Practical Issues

1. Pre-emphasis filter: We used this kind of filter in order to boost the high frequencies of the voice signal, because the poles related to the low frequencies have more energy. The filter we used has the following transfer function

$$h(z) = 1 - 0.96z^{-1}$$

2. Smooth Window Usage: When we multiply the voice signal with square windows, we insert some distortion in the frequency domain. So we use windows with smooth characteristic such as Hamming Window.

The Hamming window we used is given by

$$w(n) = 0.54 - 0.46 g \cos\left(\frac{2p n}{N-1}\right), 0 \le n \le N-1$$

3.11.2.4 Voiced / Unvoiced Detection

As we described earlier, a very important task in LPC synthesis is segmentation and labeling each segment as voiced or unvoiced. To identify whether the speech segment is voiced or unvoiced speech there are two methods widely used spectral flatness measure, energy and zero-crossing rate. In our implementation, we used the first one but both of them are analyzed below.

a) Spectral Flatness Measure

The Spectral Flatness makes use of the property that the spectrum of pure noise is completely flat. In other words, the spectrum of the unvoiced sounds is expected to be flat and the spectrum of the voiced sounds is not flat. The Spectral Flatness Measure *(SFM)* is given by

$$SFM = \frac{G_m}{A_m}$$

where G_m is the geometric mean of the magnitude spectrum and is determined by multiplying all the spectral lines and raising the final product to one over the total number of spectral lines. A_m is the arithmetic mean of the magnitude spectrum and is obtained by taking the sum of the spectral lines divided by the number of the spectral lines.

$$SFM = \frac{\left(\prod_{k=0}^{N-1} X_j(k) \right)^{\frac{1}{N}}}{\frac{1}{N} \sum_{k=0}^{N-1} X_j(k)}$$

where $X_j(k)$ is the magnitude of the N-point DTFT of the j-th frame of the speech signal. The spectral flatness measure ranges from 0.9 for white noise to 0.1 for a voiced signal. The threshold was chosen to be 0.39. In general, the threshold is chosen to be $0.35 : 0.48$.

b) Energy and zero-crossing rate

Energy of the j-th frame of the speech signal is calculated by

$$E = \sum_{n=0}^{N-1} x_j^2(n)$$

where $x_j(n)$ is the n-th sample of the j-th frame of speech. Usually the energy of the voiced part of speech is bigger than the unvoiced part of speech.

Zero-crossing rate is obtained by counting the sign changes (either from positive to negative or negative to positive) in successive speech samples. Generally the ZCR of the voiced sound is lower than the ZCR of the unvoiced sound.

3.12 Building Prosodic Models

3.12.1 Phrasing

Prosodic phrasing in speech synthesis makes the whole speech more understandable. Due to the size of peoples lungs there is a finite length of time people can talk before they can take a breath, which defines an upper bound on prosodic phrases. However we rarely make our phrases this maximum length and use phrasing to mark groups within the speech. For the most case very simple prosodic phrasing is sufficient. For Greek, as in most languages simple rules based on punctuation is a very good predictor of prosodic phrase boundaries. It is rare that punctuation exists where there is no boundary, but there will be a substantial number of prosodic boundaries which are not explicitly marked with punctuation. Thus a prosodic phrasing algorithm solely based on punctuation will typically under predict but rarely make a false insertion. In Festival, there are two methods for predicting prosodic phrases. The first and the one we used, is by CART tree (*see Chapter 4 Section 4.1for further analysis)*. The following tree is very simple and simply adds a break after the last word of a token that has following punctuation. The first condition is done by a lisp function as we want to ensure that only the last word in a token gets the break.

```
(set! tuc_gr_chris_phrase_cart_tree
'
((lisp_token_end_punc in ("?" "." ":"))
  ((BB))
  ((lisp_token_end_punc in ("'" "\"" "," ";"))
   ((B))
   ((n.name is 0)  ;; end of utterance
    ((BB))
    ((NB))))))
```

Table 13. Phrase breaking.

The second method is more complex but more accurate. We did not use it because we created a Limited Domain system, with constrained dictionary and it was not necessary to use this phrasing model. In addition to this, we could not find another Greek LPC Diphone voice in order to use it to train the data. For this model, we need three basic functions: one to determine if the current word is a function (verb, pronoun, intention etc) or content (noun, adjective) word , one to determine number of words since previous punctuation and one to determine words until next punctuation or end of utterance.

3.12.2 Duration Models

The method we used to create duration models is to use average durations for the phones. In Greek, as in most languages, phones are longer at the phrase final and to a lesser extent phrase initial positions. A simple multiplicative factor was defined for these positions. In order to achieve this we use a simple CART tree which predicts longer durations in stressed syllables and in clause initial and clause final syllables. This is shown in the next table:

```
(set! tuc_gr_chris_dur_tree
 '
   ((R:SylStructure.parent.R:Syllable.p.syl_break > 1 ) ;; clause
initial
    ((R:SylStructure.parent.stress is 1)
     ((1.5))
     ((1.2)))
    ((R:SylStructure.parent.syl_break > 1)   ;; clause final
     ((R:SylStructure.parent.stress is 1)
      ((2.0))
      ((1.5)))
     ((R:SylStructure.parent.stress is 1)
      ((1.2))
      ((1.0))))))
```

Table 14. Duration Prediction Tree.

In addition to the tree we had to define an average duration for each phone in our phoneset. The format of this information is *(SegName 0.0 Aver_Dur)*. The values of *Aver_Dur* are in seconds. This is shown in the next table

```
(set! tuc_gr_chris_phone_data
'(
   (pau 0.0 0.250)
   (A 0.0 0.090)
   (E 0.0 0.090)
   (i 0.0 0.080)
   (o 0.0 0.090)
   …
```

Table 15. Average Durations of Greek phones.

3.12.3 Intonation

Accent and boundary tones are what we will use to refer to the two main types of intonation. For Greek, and for many other languages the prediction of position of the accents and boundaries can be done as an independent process from F0 contour generation itself. As with phrase break prediction there are some simple rules that will go a surprisingly long way and as with most of the other statistical learning techniques simple rules cover most of the work. The placement of accents on stressed syllables in all content words is a quite reasonable approximation achieving about 80% accuracy on typical databases. The simplest rule for Greek is to put a hat accent on lexically stressed syllables in all content words and on all single syllable content words using an accent prediction CART tree. This is shown in the next table:

```
(set! tuc_gr_chris_accent_cart_tree
 '
  ((R:SylStructure.parent.gpos is content)
   ((stress is 1)
    ((Accented))
    ((position_type is single)
     ((Accented))
     ((NONE))))
   ((NONE))))
```

Table 16. Accent prediction CART tree.

We also need to specify the pitch range of the speaker.

```
(set! tuc_gr_chris_int_simple_params
    '((f0_mean 120) (f0_std 30)))
```

Table 17. Pitch range of the speaker.

Chapter 4

LIMITED DOMAIN CLUSTER UNIT-SELECTION SYNTHESIS

4.1 Introduction

This chapter discusses the building of the Greek Weather Forecast Synthesizer using the Unit-Selection technique in Festival. By “unit selection” we actually mean the selection of some unit of speech. The units range from whole phrases down to diphones. Diphone selection is a special case of this method. However, contrary to diphone synthesis, in unit selection synthesis there is more than one instance of the same unit and some mechanisms are used to select between them during run-time. With diphones, a fixed view of the possible space of speech units has been made which we all know is not ideal. There are articulatory effects which go over more than one phone. Nevertheless, it is not obvious which segmental effects cause variation in pronunciation. Syllable position, word/phrase initial and final position have typically a different level of articulation from segments taken from word internal position. Stressing and accents also cause differences. Rather than trying to explicitly list the desired inventory of all these phenomena and then recording all of them, a potential alternative is to take a natural corpus of speech and (semi-)automatically find the distinctions that actually exist.

The theory is obvious but the design of such systems is a non-trivial problem. Although techniques like this often produce very high quality, natural sounding synthesis, they also can produce some very bad synthesis too, when the database has unexpected holes and/or the selection costs fail.

As we want to create a Weather Forecast Synthesizer, we actually want to create a Limited Domain Unit Selection Synthesizer. By limited domain, we mean applications where the speech output is constrained as *(weather)*. Such domains may still be large but they have specific vocabulary and phrases. In fact, with today's current speech systems such limited domain applications are in fact the most common. The limited domain unit-selection offers the high quality of unit selection and avoids much of the bad selections.

4.2 Cluster Unit Selection

The idea behind Cluster Unit Selection is to take a database of speech and try to cluster each phone type into groups of acoustically similar units based on the (non-acoustic) information available at synthesis time, such as phonetic context, prosodic features (F0 and duration) and higher level features such as stressing, word position, and accents. The actual features used, may easily be changed and experimented with as the definition of acoustic distance between the units in a cluster.

The basic processes involved in building a waveform synthesizer for the clustering algorithm are the following:

- Collecting of the database of speech.
- Building utterance structures for our database.
- Building coefficients for acoustic distances, typically some form of cepstrum plus F0, or some pitch synchronous analysis (e.g. LPC).
- Building distance tables, pre-calculating the acoustic distance between each unit of the same phone type.
- Creating selection features (phone context, prosodic, positional and whatever) for each unit type.
- Building cluster trees using *wagon* with the features and acoustic distances dumped by the previous two stages.
- Building the voice description itself

First of all, we had to decide about what unit type we were going to use. Note there are two dimensions here. First is *size*, such as phone, diphone, demi-syllable. The second is *type* itself which may be simple phone, phone plus stress, phone plus word etc. The code here and the related files basically assume unit *size* is *phone*. However, because we include a percentage of the previous unit in the acoustic distance measure this unit size is more diphone-like. The cluster method has actual restrictions on the unit size, it simply clusters the given acoustic units with the given feature, but the basic synthesis code is currently assuming phone sized-units.

The second dimension, *type*, is very open. The simplest conceptual example is the one we used in the limited domain synthesis. There, we distinguished each phone with the word it comes from, thus a '*κ*' from the word '*καιρός*' is distinct from the '*κ*' in the word '*δυτικός*'.

Like diphone databases the more cleanly and carefully the speech is recorded, the better the synthesized voice will be. As we are going to select units from different parts of the database the more similar the recordings are, the less likely bad joins will occur. However, unlike diphones database, prosodic variation is probably a good thing, as it is those variations that can make synthesis from unit selection sound more natural. Good phonetic coverage is also useful, at least phone coverage if not complete diphone coverage. Also synthesis using these techniques seems to retain aspects of the original database. Since our database is weather forecast news, the synthesis from it will typically sound like read weather forecast (or more importantly will sound best when it is reading weather forecast).

The database we used, was the 4310 sentences recorded by myself *(for further analysis see Chapter 3 Section 3.3 Corpus Design).*

4.2.1 Building Utterance Structures

The first step for limited domain unit-selection synthesis method is to construct Festival utterance structures for each of the utterances in our database. In its basic form, it requires labels for: segments, syllables, words, phrases, F0 Targets, and intonation events. These were carefully labeled using the *Hidden Markov Toolkit* (HTK) *(for further analysis see
Chapter 3 Section 5 Labeling).*

Obviously real speech isn't always clean, so it is not always easy to build (reasonably) accurate structures for the real utterances. However, here we will itemize a number of functions that will make the building of utterance from real speech easier. Building utterance structures is probably worth the effort considering how easy it is to build various models from them.

In order to build an utterance structure of the type used for our Greek voice we will need label files for the following relations:

- Segment: segment labels with correct boundaries, in the phone set of our language.
- Word: words with boundaries aligned (close) to the syllables and segments.
- Syllable: Syllables, with stress marking (if appropriate) whose boundaries are closely aligned with the segment boundaries.
- IntEvent: intonation labels aligned to a syllable (either within the syllable boundary or explicitly naming the syllable they should be aligned too.
- Phrase: A name and marking for the end of each prosodic phrase.
- Target: The mean F0 value in Hertz at the mid-point of each segment in the utterance.

Segment labels and word labels were taken by training full acoustic models with *HTK*. Syllable labeling was also done automatically given segment (and word) labeling. The actual algorithm for syllabification may change but the choice is important because syllabification is consistently used throughout the rest of the system (e.g. in duration modeling).

The Target labeling required here is a single average F0 value for each segment. This currently is done fully automatically from the signal with a standard script of Festival. This is naive and a better representation of F0 could be appropriate. Phrases could potentially be determined by a combination of F0 power and silence detection but the relationship is not obvious. In general, we hand label phrases as part of the intonation labeling process. Realistically only two levels of phrasing can reliably be labeled, even though there are probably more. That is, roughly, sentence internal and sentence final. Phrase labeling is also done fully automatically with a Festival script as well as Intonation labeling. The function *build_utts* of the *build_clunits.scm* script is used to build the utterances.

4.2.2 Cepstrum Parameter Files Calculation

In order to cluster similar units, we build an acoustic representation of them. This is done by calculating *Mel-Frequency Cepstral Coefficients*. This is done pitch-sychronously. As a result, firstly we have to extract the pitchmarks of the recorded waveforms *(see Chapter 3 section 9 Extracting Pitchmarks)*.

Mel-Frequency Cepstrum (MFC) is a representation of the short-term power spectrum of a sound, based on a linear cosine transform of a log power spectrum on a non-linear mel scale of frequency.

Mel-Frequency Cepstral Coefficients (MFCC's) are coefficients that collectively create an MFC. They are derived from a type of cepstral representation of the waveform (a "spectrum of a spectrum"). The cepstral representation of a signal is the Fourier Transform of the log of the Fourier Transform of the signal. The difference between the cepstral and the MFC is that, in MFC the frequency bands are equally spaced on the mel-scale, which approximates the human auditory system's response, more closely than the linearly spaced frequency bands in the normal spectrum.

In order to derive the MFCC's we must do successively the following steps:

1. Take the Fourier Transform of (a windowed excerpt of) the signal.
2. Map the powers of the spectrum on the mel-scale, using triangular overlapping windows.
3. Take the logs of the powers at each of the mel frequencies.
4. Take the discrete cosine transform of the list of mel log powers, as if were signal.
5. The MFCC's are the amplitude of the resulting spectrum.

The mel scale is a perpetual scale of pitches judged by listeners to be equal in distance from one another. To convert f hertz into m mel we use the type

$$m = 1127.01048 \ln(1 + \frac{f}{700})$$

and the inverse

$$f = 700(e^{m/1127.01048} - 1)$$

4.2.3 Building the clusters

The script *build_clunits.scm* contains the basic parameters to build a cluster model for databases that has utterance structures and acoustic parameters. The function *build_clunits* will build the distance tables, dump the features and build the cluster trees. There are many parameters that are set for the particular database through the Lisp variable *clunits_params*. The function *build_clunits* runs through all the steps but in order to better explain what is going on, we will go through each step and at that time explain which parameters affect the substep.

The first stage is to load in all utterances in the database, sort them into segment type and name them with individual names using the function *build_clunits_init*. This uses the following parameters:

- `name` *`(STRING)`*: A name for our database.
- `db_dir` *`(FILENAME)`*: The path of the database, typically, as in the current directory.
- `utts_dir` *`(FILENAME)`*: The directory containing the utterances
- `utts_ext` *`(FILENAME)`*: The extension type of the utterances.
- `files` : The list of file_ids in the database.

The next table shows the set of the parameters in our voice.

```
  (name 'tuc_gr_chris)
   (db_dir "boubousis/festvox/festvox/data15")
   (utts_dir "festival/utts/")
   (utts_ext ".utt")
   (files ("utt_001" "utt_002" "utt_003" ... ))
```

Table 18. Initial Parameter set .

The next stage is to load the acoustic parameters and build the distance table. The acoustic distance between each segment of the same unit type is calculated and saved in the distance table. Pre-calculating this saves a lot of time as the cluster will require this numbers many times.

This is done by the next two function calls:

```
(format t "Loading coefficients\n")
 (acost:utts_load_coeffs utterances)
 (format t "Building distance tables\n")
 (acost:build_disttabs unittypes clunits_params)
```

Table 19. Function calls for loading the acousting parameters
and calculation of their acoustic distance.

The following parameters influence the behavior:

- `coeffs_dir` *`(FILENAME)`*: The directory that contains the acoustic coefficients (MFCC's)
- `coeffs_ext` *`(FILENAME)`*: The file extension for the coefficient files.

- `get_std_per_unit` : It takes the value t or nil. If t, the parameters for the type of segments are normalized by finding the means and standard deviations for the class are used. Thus a mean Mahalanobis distance is found between units rather than simply Euclidean distance. The Mahalanobis distance from a group of values with mean $m = (m_1, m_2, ...m_p)^T$ and covariance matrix Σ for a multivariate factor $c = (c_1, c_2, ...c_p)^T$ is defined as:

  $$D_M(c) = \sqrt{(c-m)^{\mathrm{T}}\Sigma^{-1}(c-m)}$$

  The unit for which the distance is minimal, is the unit with the highest probability.
- `ac_left_context` *(FLOAT)*: The amount of the previous unit to be included in the the distance. 1.0 means all, 0.0 means none. This parameter may be used to make the acoustic distance sensitive to the previous acoustic context. The recommended value, which we used, is 0.8.
- `dur_pen_weight` *(FLOAT)*:The penalty factor for duration mismatch between units.
- `ac_weights` *(FLOAT, FLOAT…)*: The weights for each parameter in the coefficient files used while finding the acoustic distance between segments. There must be the same number of weights as there are parameters in the coefficient files. The first parameter is F0. It is common to give proportionally more weight to F0 than to each individual other parameter. The remaining parameters are typically MFCCs (and possibly delta MFCCs). Finding the right parameters and weightings is one the key goals in unit selection synthesis.

The values of these parameters in our voice are shown in the next page:

```
(coeffs_dir "mcep/")
(coeffs_ext ".mcep")
(dur_pen_weight 0.1)
(get_stds_per_unit t)
(ac_left_context 0.8)
(ac_weights
    (0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5))
```

Table 20. Acousting parameters setting.

The next stage is to dump the features that will be used to index the clusters. We must remember that the clusters are defined with respect to the acoustic distance between each unit in the cluster, but they are indexed by these features. These features will be available at text-to-speech time when no acoustic information is available. Thus they include things like phonetic and prosodic context rather than spectral information. The name features may in general allow the decision tree building program wagon *(see Chapter 4 section 4 for further analysis)* to decide which of these feature actual does have an acoustic distinction in the units.

The function to dump the features is

```
(format t "Dumping features for clustering\n")
 (acost:dump_features unittypes utterances clunits_params)
```

Table 21. Function for dumping features.

The parameters which affect the function are:

- `fests_dir` *`(FILENAME)`*: The directory where the features will be saved by segment type.

- `feats` *(LIST)*: The list of features to be dumped. These are standard Festival feature names with respect to the Segment relations.

Now that we have the acoustic distances and the feature descriptions of each unit, the next stage is to find a relationship between those features and the acoustic distances. This we do using the CART tree builder wagon. It will find out questions about which features best minimize the acoustic distance between the units in that class. That is we are trying to classify *all* the units in the database, there is no test set as such. However in synthesis there will be desired units whose feature vector didn't exist in the training set.

The clusters are built by the following function:

```
(format t "Building cluster trees\n")
 (acost:find_clusters (mapcar car unittypes) clunits_params)
```

Table 22. Cluster building function.

The parameters that affect the tree building process are:

- `tree_dir` *(FILENAME)*: The directory where the decision tree for each segment type will be saved.
- `wagon_field_desc` *(LIST)*: A filename of a wagon field descriptor file. This is a standard field description (field name plus field type) that is required for wagon.
- `wagon_progname` *(FILENAME)*: The pathname for the wagon CART building program.
- `wagon_cluster_size` *(INT)*: The minimum cluster size.
- `prune_reduce` *(INT)*: The number of elements in each cluster to remove in pruning. This removes the units that are furthest from the center. This is done with the wagon training.

- `cluster_prune_limit` *`(INT)`*: This is a post wagon build operation on the generated trees (and perhaps a more reliably method of pruning). This defines the maximum number of units that will be in a cluster at a tree leaf. The wagon cluster size the minimum size. This is useful when there are some large numbers of some particular unit types which cannot be differentiated. Format example silence segments without context of nothing other silence. Another usage of this is to cause only the center example units to be used.
- `unittype_prune_threshold` *`(INT)`*: This defines the minimal number of units of that type required before building a tree.

In our voice, the above parameters have the following values:

```
(trees_dir "festival/trees/")
(wagon_field_desc "festival/clunits/all.desc")
(wagon_progname "/boubousis/speech_tools/bin/wagon")
(wagon_cluster_size 10)
(prune_reduce 0)
```

Table 23. Cluster building parameters.

The final stage in building a cluster model is collecting the generated trees into a single file and dumping the unit catalogue, i.e. the list of unit names and their files and position in them. This is done by the following lisp function

```
(acost:collect_trees (mapcar car unittypes) clunits_params)
(format t "Saving unit catalogue\n")
(acost:save_catalogue utterances clunits_params)
```

Table 24. Function for the creation of the unit catalogue.

The only parameter that affect this is

- `catalogue_dir` *`(FILENAME)`*`:` The directory where the catalogue will be saved.

By default this is

```
(catalogue_dir "festival/clunits/")
```

Moreover, there is a number of parameters that are specified within our cluster voice. These are related to the run-time aspects of the cluster model. These are

- `join_weights` *`(FLOATLIST)`*`:` These are a set of weights, in the same format as `ac_weights` that are used in optimal coupling to find the best join point between two candidate units.
- `continuity_weight` *`(FLOAT)`*`:` The factor to multiply the join cost over the target cost. This is probably not very relevant given the the target cost is merely the position from the cluster center.
- `log_scores :` If specified the join scores are converted to logs. For databases that have a tendency to contain non-optimal joins (non-limited domain databases), this may be useful to stop failed synthesis of longer sentences. The problem is that the sum of very large number can lead to overflow. This helps reduce this. For our voice, as it is for limited-domain, we did not need to use it.
- `optimal_coupling` *`(INT)`*`:` It takes two values 1 and 2. If 1, this uses optimal coupling and searches the cepstrum vectors at each join point to find the best possible join point. This is computationally expensive (as well as having to load in lots of cepstrum files), but does give better results. If the value is 2 this only checks the coupling distance at the given boundary (and doesn't move it). This is often adequate in good databases (e.g. limited domain), and is certainly faster
- `extend_selections` *`(INT)`*`:` If 1 then the selected cluster will be extended to include any unit from the cluster of the previous segments candidate units that has correct phone type (and isn't already included in the current cluster). This means that instead of selecting just units selection is effectively selecting the beginnings of multiple segment units. This option encourages far longer units.

- `pm_coeffs_dir` *(FILENAME)*: The directory where the pitch marks are.
- `pm_coeffs_ext` *(FILENAME)*: The file extension for the pitchmark files.
- `sig_dir` *(FILENAME)*: The directory containing waveforms of the units.
- `sig_ext` *(FILENAME)*: File extension for waveforms.
- `join_method` (*METHOD)*: It specifies the method used for joining the selected units. Currently it supports `simple`, a very naive joining mechanism, and `windowed`, where the ends of the units are windowed using a hamming window then overlapped (no prosodic modification takes place though). The other two possible values for this feature are `none` which does nothing, and `modified_lpc` which uses the standard UniSyn module to modify the selected units to match the targets.
- `clunits_debug` *(1 or 2)*: With a value of `1` some debugging information is printed during synthesis, particularly how many candidate phones are available at each stage (and any extended ones). Also where each phone is coming from is printed. With a value of `2` more debugging information is given include the above plus joining costs (which are very readable by humans).

4.3 Classification and Regression Trees (CART)

The CART algorithm builds classification and regression trees for predicting continuous dependent variables *(regression)* and categorical predictor variables (*classification)*. The classic CART algorithm was popularized by Breiman at 1984.

Regression-type problems. Regression-type problems are generally those where one attempts to predict the values of a continuous variable from one or more continuous and/or categorical predictor variables. For example, you may want to predict the selling prices of single family homes (a continuous dependent variable) from various other continuous predictors (e.g square footage) as well as categorical predictors (e.g., style of home, such as ranch, two-story, etc.; zip code or telephone area code where the property is located, etc.; note that this latter variable would be categorical in nature, even though it would contain numeric values or codes). If you used simple multiple regression, or some general linear model (*GLM*) to predict the selling prices of single family homes, you would determine a linear equation for these variables that can be used to compute predicted selling prices. There are many different analytic procedures for fitting linear models (*GLM*, *GRM*, *Regression*), various types of nonlinear models (e.g., *Generalized Linear/Nonlinear Models (GLZ)*, *Generalized Additive Models (GAM)*, etc.), or completely custom-defined nonlinear models (see *Nonlinear Estimation*), where you can type in an arbitrary equation containing parameters to be estimated. CHAID also analyzes regression-type problems, and produces results that are similar (in nature) to those computed by *CART*. Note that various neural network architectures are also applicable to solve regression-type problems.

Classification-type problems. Classification-type problems are generally those where one attempts to predict values of a categorical dependent variable (class, group membership, etc.) from one or more continuous and/or categorical predictor. For example, you may be interested in predicting who will or will not graduate from college, or who will or will not renew a subscription. These would be examples of simple binary classification problems, where the categorical dependent variable can only assume two distinct and mutually exclusive values. In other cases one might be interested in predicting which one of multiple different alternative consumer products (e.g., makes of cars) a person decides to purchase, or which type of failure occurs with different types of engines. In those cases there are multiple categories or classes for the categorical dependent variable. There are a number of methods for analyzing classification-type problems and to compute predicted classifications, either from simple continuous predictors (e.g., binomial or multinomial logit regression in *GLZ*), from categorical predictors (e.g., *Log-Linear Analysis* of multi-way frequency tables), or both (e.g., via ANCOVA-like designs in *GLZ* or *GDA*).

4.3.1 Advantages of CART Methods

As mentioned earlier, there are a large number of methods that an analyst can choose from when analyzing classification or regression problems. Tree classification techniques, when they "work" and produce accurate predictions or predicted classifications based on few logical if-then conditions, have a number of advantages over many of those alternative techniques.

Simplicity of results: In most cases, the interpretation of results summarized in a tree is very simple. This simplicity is useful not only for purposes of rapid classification of new observations (it is much easier to evaluate just one or two logical conditions, than to compute classification scores for each possible group, or predicted values, based on all predictors and using possibly some complex nonlinear model equations), but can also often yield a much simpler "model" for explaining why observations are classified or predicted in a particular manner (e.g., when analyzing business problems, it is much easier to present a few simple if-then statements to management, than some elaborate equations).

Tree methods are nonparametric and nonlinear. The final results of using tree methods for classification or regression can be summarized in a series of (usually few) logical if-then conditions (tree nodes). Therefore, there is no implicit assumption that the underlying relationships between the predictor variables and the dependent variable are linear, follow some specific non-linear link function [e.g., see *Generalized Linear/Nonlinear Models (GLZ)*], or that they are even monotonic in nature. For example, some continuous outcome variable of interest could be positively related to a variable Income if the income is less than some certain amount, but negatively related if it is more than that amount (i.e., the tree could reveal multiple splits based on the same variable Income, revealing such a non-monotonic relationship between the variables). Thus, tree methods are particularly well suited for data mining tasks, where there is often little *a priori* knowledge nor any coherent set of theories or predictions regarding which variables are related and how. In those types of data analyses, tree methods can often reveal simple relationships between just a few variables that could have easily gone unnoticed using other analytic techniques.

4.3.2 CARTs in Festival

The construction of CARTs is the basic method in Festival for building statistical models from simple feature data. CART is powerful because it can deal with incomplete data, multiple types of features (floats, unumerated sets) both in input features and predicted features, and the trees it produces often contain rules which are humanly readable.

Decision trees contain a binary question (yes/no answer) about some feature at each node in the tree. The leaves of the tree contain the best prediction based on the training data. Decision lists are a reduced form of this where one answer to each question leads directly to a leaf node. A tree's leaf node may be a single member of some class, a probability density function (over some discrete class), a predicted mean value for a continuous feature or a gaussian (mean and standard deviation for a continuous value). Theoretically, the predicted value may be anything for which a function can defined that can give a measure of *impurity* for a set of samples, and a distance measure between impurities.

The basic algorithm is given a set of samples (a feature vector), finds the question about some feature which splits the data minimizing the mean impurity of the two partitions and recursively applies this splitting on each partition until some stop criteria is reached (e.g a minimum number of samples in the partition). The basic CART building algorithm is a *greedy algorithm* that chooses the locally best discriminatory feature at each stage in the process. This is suboptimal but a full search for a fully optimized set of questions would be computationally very expensive. Although there are pathological cases in most data sets this greediness is not a problem. The basic building algorithm starts with a set of feature vectors representing samples. At each stage, all possible questions for all possible features are asked about the data finding out how the question splits the data. A measurement of impurity of each partitioning is made and the question that generates the least partitions is selected. This process is applied recursively on each sub-partition until some stop criteria is met.

4.3.2.1 Impurity

The impurity of a set of samples is designed to capture how similar the samples are to each other. The smaller the number, the less impure the set is.

For sample sets with continuous predictees `Wagon` uses the variance time number of sample points. The variance alone could be used by this overly favor very small sample sets. As the test that uses the impurity is trying to minimize it over a partitioning of the data, multiple each part with the number of samples will encourage larger partitions, which we have found lead to better decision trees in general.

For sample sets with discrete predictees `Wagon` uses the entropy time number of sample points. Again the number of sample points is used in so that small sample set are not unfairly favored. The entropy for a sample set is calculated as

```
sumof (for each x in class)
            prob(x)*log(prob(x))
```

Table 25. Calculation of the Entropy.

Other impurity measure could be used if required. For example an experimental cluster technique used for unit selection actually used impurity calculated as the mean Euclidean distance between all vectors of parameters in the sample set. However the above two are more standard measures.

4.3.2.2 Question Forming

`Wagon` has to automatically form questions about each feature in the data set.

For discrete features, questions are built for each member of the set, e.g. if feature n has value x. Our implementation does not currently support more complex questions which could achieve better results (though at the expense of training time). Questions about features being some subset of the class members may give smaller trees. If the data requires distinction of values a, b and c, from d e and f, our method would require three separate questions, while if subset questions could be formed this could be done in one step which would not only give a smaller tree but also split the samples for a, b and c. In general subset forming is exponential on the number items in the class though there are techniques that can reduce this with heuristics. However these are currently not supported.

For continuous features, `Wagon` tries to find a partition of the range of the values that best optimizes the average impurity of the partitions. This is currently done by linearly splitting the range into a predefined subpart (10 by default) and testing each split. This again isn't optimal but does offer reasonably accuracy without requiring vast amounts of computation.

4.3.2.3 Tree Building Criteria

There are many ways to constrain the tree building algorithm to help build the "best" tree. Wagon supports many of these (though there are almost certainly others that is does not).
In the most basic forms of the tree building algorithm a fully exhaustive classification of all samples would be achieved. This, of course is unlikely to be good when given samples that are not contained within the training data. Thus the object is to build a classification or regression tree that will be most suitable for new unseen samples. The most basic method to achieve this is not to build a full tree but require that there are at least n samples in a partition before a question split is considered. We refer to that as the *stop* value. A number like 50 as a stop value will often be good, but depending of the amount of data you have, the distribution of it, etc various stop value may produce more general trees.

A second method for building "good" trees is to *hold out* some of the training data and build a (probably over-trained) tree with a small stop value. Then prune the tree back to where it best matches the held out data. This can often produce better results than a fixed stop value as this effectively allows the stop value to vary through different parts of the tree depending on how general the prediction is when compared against held out data. It is often better to try to build more balanced trees. A small stop value may cause the tree building algorithm to find small coherent sets of samples with very specific questions. The result tree becomes heavily lop-sided and (perhaps) not optimal. Rather than having the same literal stop value more balanced trees can built if the stop value is defined to be some percentage of the number of samples under consideration. This percentage we call a *balance* factor. Thus the stop value is then the largest of the defined fixed stop value or the balance factor times the number of samples. To some extent the multiplication of the entropy (or variance) by the number of samples in the impurity measure is also way to combat imbalance in tree building.

A good technique is to build trees in a *stepwise* fashion. In this case, instead of considering all features in building the best tree, we incrementally build trees looking for which individual feature best increases the accuracy of the tree on the provided test data. Unlike within the tree building process where we are looking for the best question over all features, this technique limits which features are available for consideration. It first builds a tree using only the features provided, looking for which individual feature provides the best tree. The selection feature builds n-1 trees with the best feature from the first round with each of the remaining features. This process continues until no more features add to the accuracy or some stopping criteria (percentage improved) is not reached. This technique is also a greedy technique but we've found that when many features are presented, especially when some are highly correlated with each other, stepwise building produces a significantly more robust tree on external test data. It also typically builds smaller trees, but of course
there is a cost in computation time. Stepwise tests each success tree against the specified test set. As this is using the test set which optimizes the tree, it is not valid to view the specified test set as a genuine test set. Another externally held test set should be used to test the accuracy of generated tree.

4.3.2.4 Tree Format

The generated tree files are written as Lisp s-expressions as this is by far the easiest external method to represent trees. Even if the trees are read by something other than Lisp it is easy to write a reader for such a format. Note that not all of the question types are generated by `Wagon` but they are supported by the interpreters. The leaf nodes differ depending on the type of the predictee. For continuous predictees (regression trees) the leaves consist of a pair of floats, the stddev and mean. For discrete predictees (classification trees) the leaves are a probability density function for the members of the class. Also the last member of the list is the most probable value. Note that in both cases the last value of the leaf list is the answer desired in many cases. The syntax of the CART tree is shown in the next table:

```
TREE ::= LEAF | QUESTION-NODE

QUESTION-NODE ::= "(" QUESTION YES-NODE NO-NODE ")"

YES-NODE ::= TREE

NO-NODE ::= TREE

QUESTION ::= "(" FEATURENAME "is" VALUE ")" |
             "(" FEATURENAME "=" FLOAT ")" |
             "(" FEATURENAME "<" FLOAT ")" |
             "(" FEATURENAME ">" FLOAT ")" |
             "(" FEATURENAME "matches" REGEX ")" |
             "(" FEATURENAME "in" "(" VALUE0 VALUE1 ... ")" ")"

LEAF ::= "(" STDDEV MEAN ")" |
         "(" "(" VALUE0 PROB0 ")" "(" VALUE1 PROB1 ")" ... MOSTPROBVAL
")" |
         any other lisp s-expression
```

Table 26. Syntax of a CART.

4.4 Wagon CART-building Program

`Wagon` is a program that is included in the Edinburgh Speech Tools, the speech processing library which is distributed with Festival. `Wagon` is used to build decision CART trees from feature data. Its basic features include:

- Both decision trees and decision lists are supported.
- Predictees can be discrete or continuous.
- Input features can be discrete or continuous.
- Many options for controlling tree building
    - fixed stop value
    - balancing
    - held-out data and pruning
    - stepwise use of input features
    - choice of optimization criteria (correct/entropy for classification rmse/covariance for regression)

The input data for `wagon` (and some other model building tools in the Edinburgh Speech Tools library), should consist of feature vectors, and a description of the fields in these vectors.

4.4.1 Feature Vectors

A feature vector is a file with one sample per line, with feature value as white space separated tokens. The Festival program `dumpfeats` is specifically designed to generate such files from databases of utterances but these files may be generated from any data source. Each vector must have the same number of features and in the same order. Features may be specified as "ignored" in the description (or in actual use) so it is common that data files contain more features than are always used in model building. By default the first feature in a data file is the predictee, though at least in wagon the predictee field can be named at tree building time to be other than the first field. Features can be discrete or continuous but at present must be single valued, "multi-valued" or "list-valued" features are not currently supported. Note this means that a feature in different samples may have different values but in a particular sample a particular feature can only have one value. You must also note that it is common to have thousands, even hundreds of thousands of samples in a data file, and the number of features can often be in the hundreds, though can also be less than ten depending on the what it describes. A typical example is shown in the next table:

```
0.399 pau i   0   0 1 1 0 0 0 0 0 0
0.082 i   A   pau 0 1 0 0 1 1 0 1 1
0.074 A   n   i   1 0 1 0 1 1 1 1 0
0.048 n   E   A   0 1 0 1 1 1 1 0 0
0.062 E   m   n   2 0 0 1 1 1 0 0 0
0.020 m   i   E   0 0 1 1 1 1 0 0 0
0.082 i   th  m   3 1 0 1 1 1 0 0 0
0.082 th  A   i   1 0 0 1 1 1 0 0 0
```

Table 27. Example of feature vector.

4.4.2 Data Descriptions

A data file also requires a description file which names and classifies the features in a datafile. Features must have names so they can be referred to in the decision tree (or other model output) and also be classified into their type. The basic types available for features are:

- continuous: for features that range over reals (e.g duration of phones)
- categorial: for features with a pre-defined list of possible values (e.g phone names)
- string: for features with an open class of discrete values (e.g words)

The data description consists of a parenthesized list of feature descriptions. Each feature description consists of the feature name and its type (and/or possible values). Feature names, by convention, should be features names in the sense for features (and pathnames) used throughout the utterance structures in the Edinburgh Speech Tools. The expected method to use models generated from features sets in the Edinburgh Speech Tools is to apply them to items. In that sense having a feature name be a feature of an item (or relatve) pathname will avoid having the extra step of extracting features into a separated table before applying the model. However, it should also be stated that to `wagon` these names are arbitrary tokens and their semantic irrelevant at training time. A typical description file is shown in the next table, and it is suitable for the data file given above

```
((segment_duration float)
 ( name  A E i o u J ly D G C b p d th g q t k f v s z l m n r x ts tz
Y ks pau )
 ( n.name 0 A E i o u J ly D G C b p d th g q t k f v s z l m n r x ts
tz Y ks pau )
 ( p.name 0 A E i o u J ly D G C b p d th g q t k f v s z l m n r x ts
tz Y ks pau )
 (pos_in_syl float)
 (syl_initial 0 1)
 (syl_final   0 1)
 (R:Sylstructure.parent.R:Syllable.p.syl_break float)
 (R:Sylstructure.parent.syl_break float)
 (R:Sylstructure.parent.R:Syllable.n.syl_break float)
 (R:Sylstructure.parent.R:Syllable.p.stress 0 1)
 (R:Sylstructure.parent.stress 0 1)
 (R:Sylstructure.parent.R:Syllable.n.stress 0 1)
)
```

Table 28. Example of a description file.

There is also a number of special symbols that may be used in a description file. If the type (first token after the name) is `ignore` the feature will be ignored in the model building process. We may also specify features to ignore at tree building time but it is often convenient to explicitly ignore feature(s) in the description file.

A description file can't be generated automatically from a data set though an approximation is possible. Particularly it is not possible to automatically decide if a feature value is continuous of which its example values happen to look like numbers. The script `make_wagon_desc` takes a datafile, a file containing only the names of the features, and the name of the description file it will create. However, it should be checked manually.

Chapter 5

EVALUATION OF THE GREEK WEATHER FORECAST SYNTHESIZER

5.1 Introduction

In spite of the rapid progress that is being made in the speech technology, any speech synthesis available today can still be spotted for what it is: nonhuman, machine. Although there have been major improvements in the quality of the output of TTS systems, as long as synthetic speech is inferior to human speech, synthesis evaluation will be useful.

Speech synthesis assessment is important to two parties: system designers on one hand and prospective buyers on the other hand. Designers always try to improve their TTS systems. However, designers who have “grown up” with their systems are used to all its habits; they are likely to understand its output better than first-time users, and will often overrate its performance level. More meaningful quality assessment techniques are needed in order to determine how well a system performs relative to a benchmark test or how favorably it compares it with a previous edition of the system or with another’s designer product. To the extent that a system performs less than perfect (something of which the *author* is aware), the designer can learn which aspects of the problem are flawed.

On the other hand, the needs of buyers and end user are different than those of the designers but they too heavily rely on assessment techniques. Prospective buyers will always have a specific ose of their TTS system in mind. Understandably, they will want the simplest and therefore cheapest system that satisfies their needs. The buyer therefore needs an absolute yardstick in order to determine beforehand if the TTS system is good enough to get a message across in the given application.

5.2 Taxonomy of Evaluation Tasks & Techniques

To justify our selections for the evaluation strategy used for the quality assessment of our TTS system, we will first discuss a number of distinguished parameters and explain the relationships between them.

The next figure shows the various dichotomies in the hierarchical order in which they have been listed in the diagram. Any path from the root to a terminal that does not cross an horizontal gap constitutes a meaningful combination of test attributes

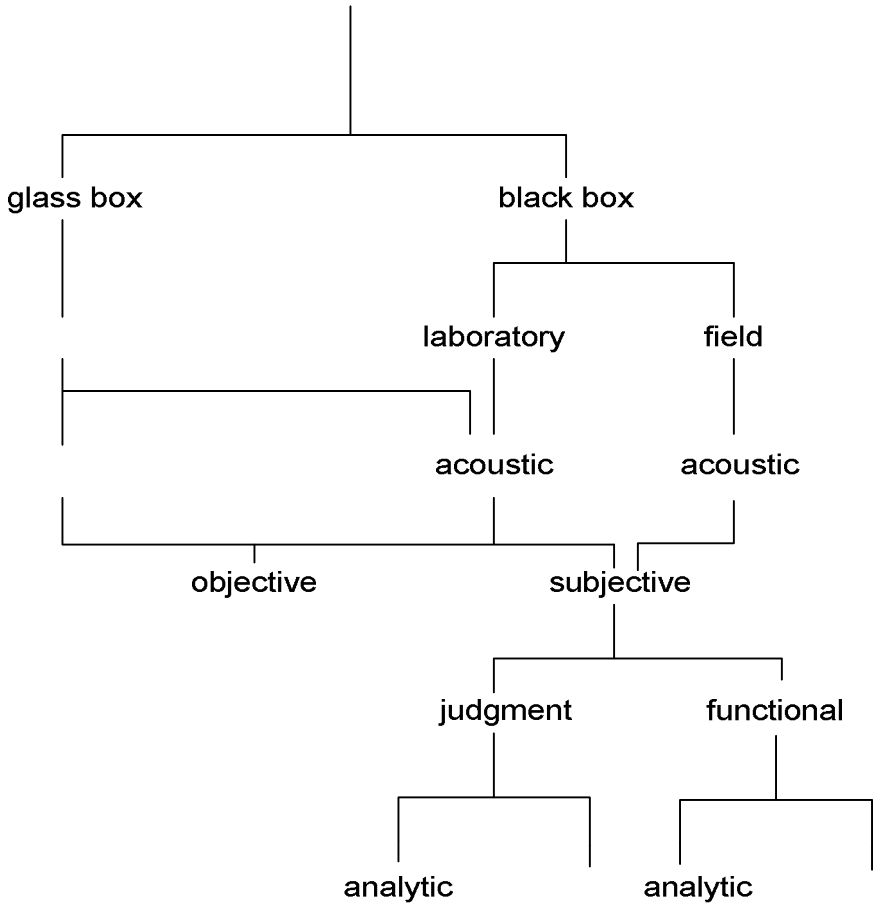


Figure 18. Relationships among dimensions involved in Taxonomy of speech output evaluation methods

5.2.1 Black Box vs Glass Box

TTS systems generally comprise of a range of modules that take care of specific tasks (e.g concatenation, signal processing). End users will be interested in the performance of a system as a whole. They will consider the system as a *black box* that accepts text and outputs speech, without any internal structure, since the quality of the output speech is the only thing that matters.

However, if the output is less than optimal, it is impossible to pinpoint a specific cause of the specific malfunction. Hence, the designers set the evaluations in a more experimental way (glass box). This is achieved by keeping all modules but one constant, while systematically varying the characteristics of the latter, allows for any difference in the assessment of the system to be attributed to the variations of the target module.

5.2.2 Laboratory vs Field

TTS systems are often part of a human-machine user interface in a specific application. Typically, the vocabulary and types of information exchanges are restricted and domain specific, so that situational redundancy can often make up for bad intelligibility. On the other hand, TTS systems will often be used in complex information processing tasks, so that the listener has only limited resources available for attending to the speech input.

It is generally impossible to predict beforehand, on the basis of *laboratory tests*, exactly how successful a TTS-system will be in the practical application. The system needs to be tested in the *field*, i.e. in the real application, with real users. However, the use of *field tests* is limited to one system in one specific application; results of the test cannot, as a rule, be generalized to other systems and/or other applications.

5.2.3 Linguistic vs Acoustic

Complex TTS systems can roughly be divided into a linguistic interface that transforms spelling into an abstract phonological code and an acoustical interface that transduces this symbolic representation to an audible waveform.

The quality of the intermediary representation can be tested directly at the *symbolic-linguistic* level or indirectly at the level of the *acoustic* output. Testing the audio has the advantage that only errors in the symbolic representation that affect the audio output will affect the evaluation. However, it concerns human listeners and is therefore costly and time consuming. Moreover the designer is not informed on the origin of any problems (linguistic or acoustic).

As an alternative, the intermediate representations in the linguistic interface are often evaluated in the symbolic level. It involves the comparison of the symbolic output of the linguistic model to a pre-stored model representation. The non-trivial problem is to obtain this model representation, which will have to be compiled manually, and will often involve multiple correct solutions

5.2.4 Subjective vs Objective

When an assessment technique involves the responses of human subjects, the measurement is called *subjective*. It is most common that human subjects are called upon in order to evaluate the quality of a TTS system. This is to be expected, since the end user of a TTS system is a human listener. However there are certain drawbacks inherent to the use of human subjects. Firstly, humans are often somewhat noisy in their judgments, i.e. the results of tests are never perfectly reproducible. It often makes sense to use an expert listener as a shortcut to a preliminary evaluation, since he will be able to determine in great accuracy problems related to coarticulation, temporal organization and intonation. However he will
not be able to predict in numerical terms how well the TTS system would perform as a communication tool with naïve listeners. Since this is what we need to assess, expert listeners should be used during the initial stages of development, as a design tool, while non-expert users should be used for the final evaluation of the system. In this case, a group of users may be used, and the average of their responses could somewhat compensate for the noisiness of their measurements. This is what is called *inter-subjective* measurement.

In addition to yielding noisy measurements, quality tests involving human listeners are also time consuming and therefore expensive to run. Automatic quality assessment for TTS systems that automatically measure the discrepancy in acoustical terms between a system's output and its human model is still a field under investigation. This is the type of *objective* evaluation technique that one would ultimately want to come up with, since it avoids the use of human listeners, providing perfectly reproducible results in as little time as needed to run that particular test program. Unfortunately, these types of services are not yet available for usage.

5.2.5 Judgement vs Functional

By *judgment* testing we mean a procedure whereby a group of listeners is asked to judge the performance of a TTS system, along a number of rating scales. The scales are typically bi-polar adjectives that allow the listeners to express the quality of the system.

A TTS system may also be assessed in terms of how well it actually performs its communicative purpose. This is called *functional* testing. For instance, if we want to know to what extent the output speech is intelligible, we may measure its intelligibility not by asking the listener how intelligible he things it is, but by determining, for instance, whether the listener correctly identifies the sounds.

5.2.6 Global vs Analytic

Judgment test usually include one or more rating scales covering such *global* aspects as “overall quality”, “naturalness” and “acceptability”. On the other hand, one may be interested in determining the quality of specific aspects of a TTS system, in an *analytic* listening mode, where listeners are requested to pay particular attention to selected aspects of the speech output.

5.3 Evaluation of the Greek Weather Synthesizer

The quality assessment of our synthesizer was based on the evaluation of its acoustic aspects. There are three layers that are distinguished in speech: a segmental layer, a prosodic layer and the voice quality layer. We will make the same distinction in the evaluation of acoustic aspects.

5.3.1 Segments

The primary function of segments is to enable the listener to identify words. In the LPC synthesis the units are diphones, while in the Cluster Unit Selection Method are phones. The evaluation done in this level, is to what extent the listener understands the synthesized words with the two synthesis methods.

5.3.2 Prosody

By prosody we mean the ensemble of properties of speech utterances that cannot be derived in a straightforward fashion from the identity of the phonemes constituting the words of the speech utterance. Prosody comprises the melody of the speech, word and phrase boundaries, word stress, sentence accent, tempo and changes in speaking rate.

The more important functions of prosody are located at the linguistic levels above the word:

- prosody tells the listener which words go together and should be interpreted as making up a coherent chunk of information; it also allows the user to determine whether he has come to the end of a word group, clause, sentence, etc.
- prosody provides an indication for the listener which words are presented by the speaker as expressing important information.

- prosody, especially melody, carries its own intonational meaning, allowing for instance the speaker to present a sentence as a statement or a question.

These observations suggest that prosody affects comprehension, which is what most functional tests of prosody try to evaluate.

5.3.3 Voice Quality

Voice quality can be viewed as the background against which segmental and prosodic variation is produced and perceived. It is used by the listener to form a (sometimes incorrect) idea of the speaker‟s mood and personality, physical size, sex, and also to identify the speaker. This information may have practical consequences for the continuation of the communication procedure, since it may influence the listener‟s attitude towards the speaker in a positive or negative sense, and may affect the listener‟s interpretation of the message.

5.3.4 Overall Output Quality

In most situations good intelligibility of specific words is not enough for TTS output to be called functionally adequate. One would want to have at one's disposal a functional test to evaluate the adequacy if the complete TTS output in all respects. In practice, the functional quality of overall TTS output has been equated with comprehension, based upon the integration of "bottom-up" speech signal information at different levels (segments, prosody, voice quality) and "top-down" knowledge and expectations based on previous experience, specific properties of the extra-linguistic context, and word internal and word combinatory redundancy.

5.4 Test Method

The importance of application specific test materials has been stressed by ITU-T's standardization sector. They developed a test specifically aimed at evaluating the quality of telephone speech, and which has been modified to fit our purposes. It is a judgment test comprising rating on eight scales, namely one 2-point scale *acceptance* and seven 5-point scales *overall impression*, *listening effort*, *comprehension problems*, *articulation*, *pronunciation*, *speaking rate*, and *voice pleasantness.*

Strictly speaking, only the first four scales can be captured under the heading *overall quality*; the other four scales are directed at more specific aspects of the output and require analytic listening. The content of the speech samples are synthesized in accordance with the application.

For our purposes, we modified the ITU-T test. We maintained the 2-point scale *acceptance*, replaced the first four scales with a 5-point *overall quality* scale, the next two *articulation* and *pronunciation* with a 5-point *phoneme juncture* and the final two with a 5-point *intonation* scale.

Then, we synthesized with both synthesis methods fifty sentences forty of which were relevant to weather and ten irrelevant. We did this, in order to check what would be the response of our synthesizer to text, in which it was not trained. We then passed an evaluation form to ten people who had to listen to all sentences and then evaluate the speech using the four scales mentioned above. For the first scale (acceptance), the evaluation should determine whether the synthesized speech is accepted or not. For the rest three scales, the evaluation was done by assigning a grade in the range of [0, 5], with 5 denoting the best performance. The evaluations for each method among all people were averaged, providing a measure for the performance of the method in each of these four scales.

One example of the evaluation form is the following:

TTS	Αποδοχή	Ποιότητα Φωνής	Προσωδία	Σύνδεση Φωνημάτων	Κατανόηση Περιεχομένου
Greek Weather Forecast Synthesizer	2.Ναι 1. Όχι	5. Τέλεια 4. Πολύ Καλή 3. Καλή 2. Ok 1. Κακή	5. Απόλυτα Φυσική 4. Κανονική 3. Αποδεκτή 2. Αστεία σε κάποια σημεία 1. Παράξενη	5. Τέλεια 4. Καλή 3. Αστεία σε κάποια σημεία 2. Αφύσικη 1. Παράξενη	2. Ναι 1. Όχι
tts_199.wav					

Table 29. Example of our Evaluation form.

The tts_001.wav is the original recording, tts_1001.wav is the synthesized utterance with the Diphone LPC Synthesis method and tts_2001.wav is the synthesized utterance with the Limited-Domain Cluster Unit-Selection Synthesis method.

5.5 Results

As we discussed in the previou section, we splitted the evaluation process into two sections. The first section included synthesizing sentences related exclusively to weather, while the second section general sentences irrelative to weather. We did that, in order to check the performance of our Weather Synthesizer to an input that was not related to weather. We wanted to evaluate the performance of any combination of the selection criteria comparing the two synthesis methods, LPC and Cluster Unit-Selection for each synthesized sentence. The evaluations for each method were averaged, providing a measure for the performance of the method in each of the four scales. The extracted charts from this process are the following:

Weather Input

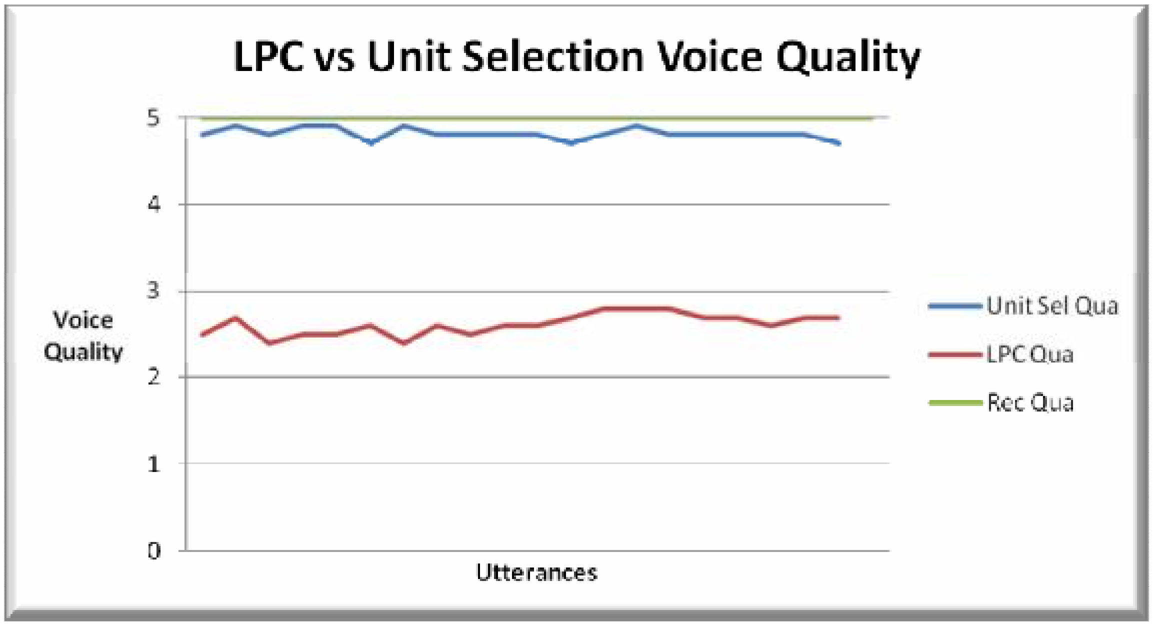


Figure 19. Evaluation Results (Voice Quality)

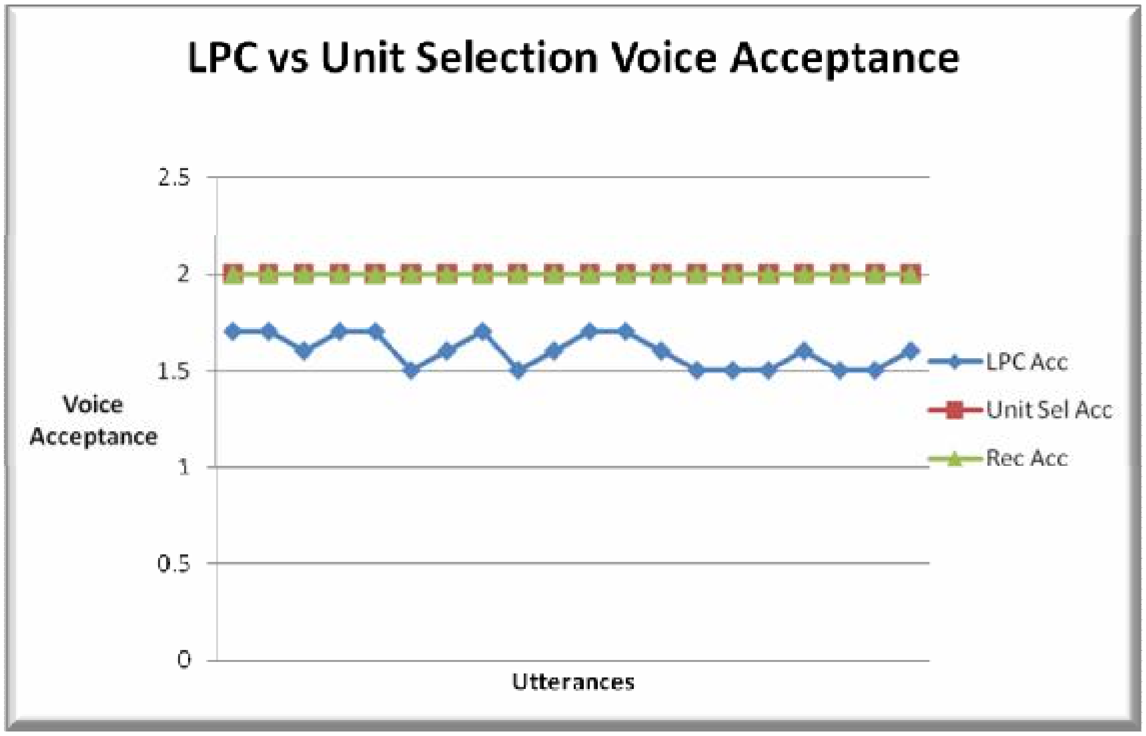


Figure 20. Evaluation Results (Voice Acceptance).

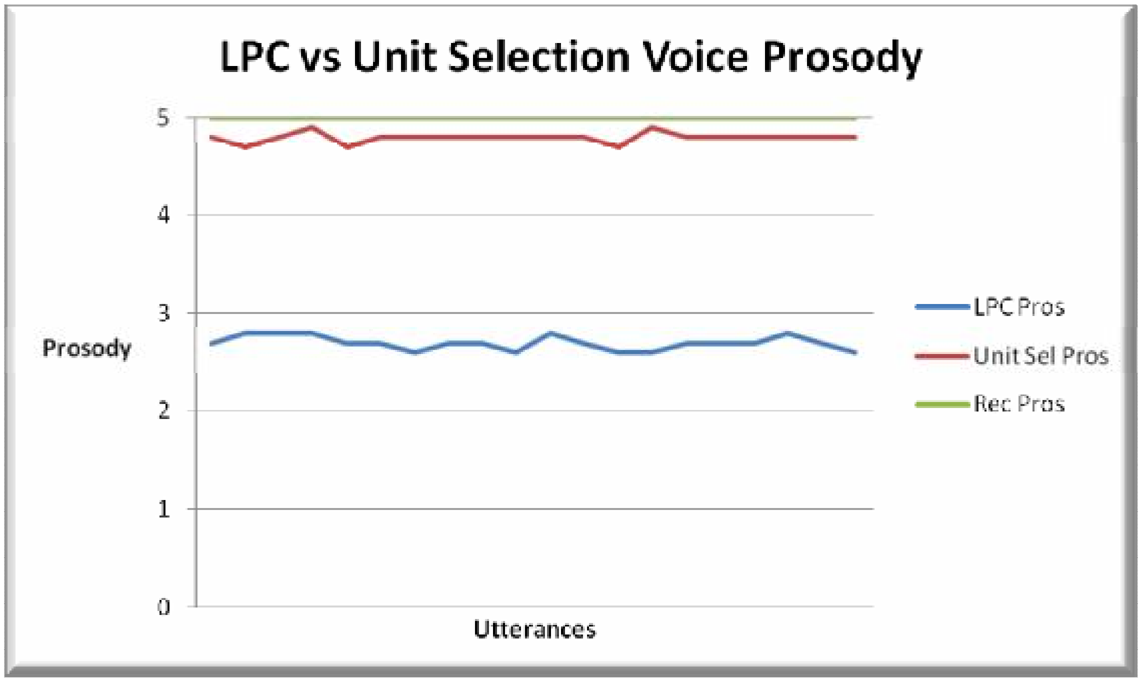


Figure 21. Evaluation Results (Prosody).

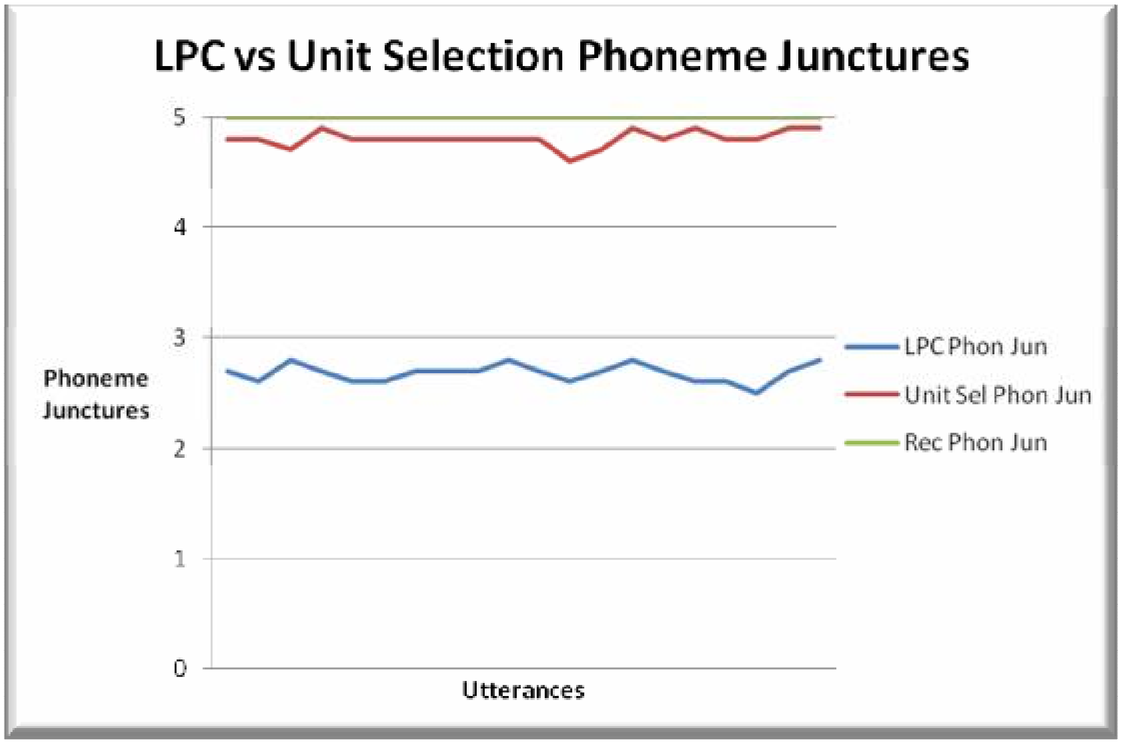


Figure 22. Evaluation Results (Phoneme Junctures).

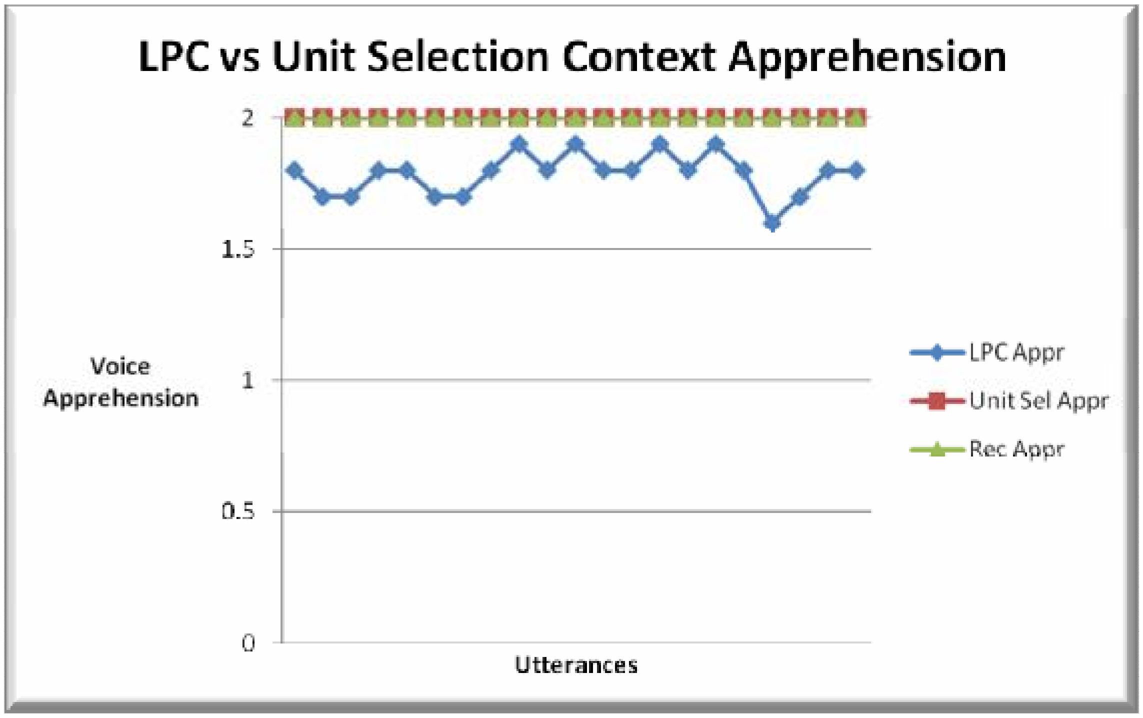


Figure 23. Evaluation Results (Context Apprehension).

As we can understand from the above figures the performance of the Cluster Unit-Selection Voice is much greater than the LPC voice. *Figure 20* shows us that the majority of the subjects did not accept the LPC synthesis method as an adequate method of Speech synthesis while the Unit-Selection method was accepted by every subject. The rest three *figures* indicate us that the scores of the Unit-Selection Method for each scale (*Voice Quality*, *Intonation*, *Phoneme Junctures*) are close to perfect (5) while the scores of the LPC Method flunctuate between 1 and 2. Then, we set as input in our synthesizer non-weather sentences and the results are shown in the following figures:

Generic Input

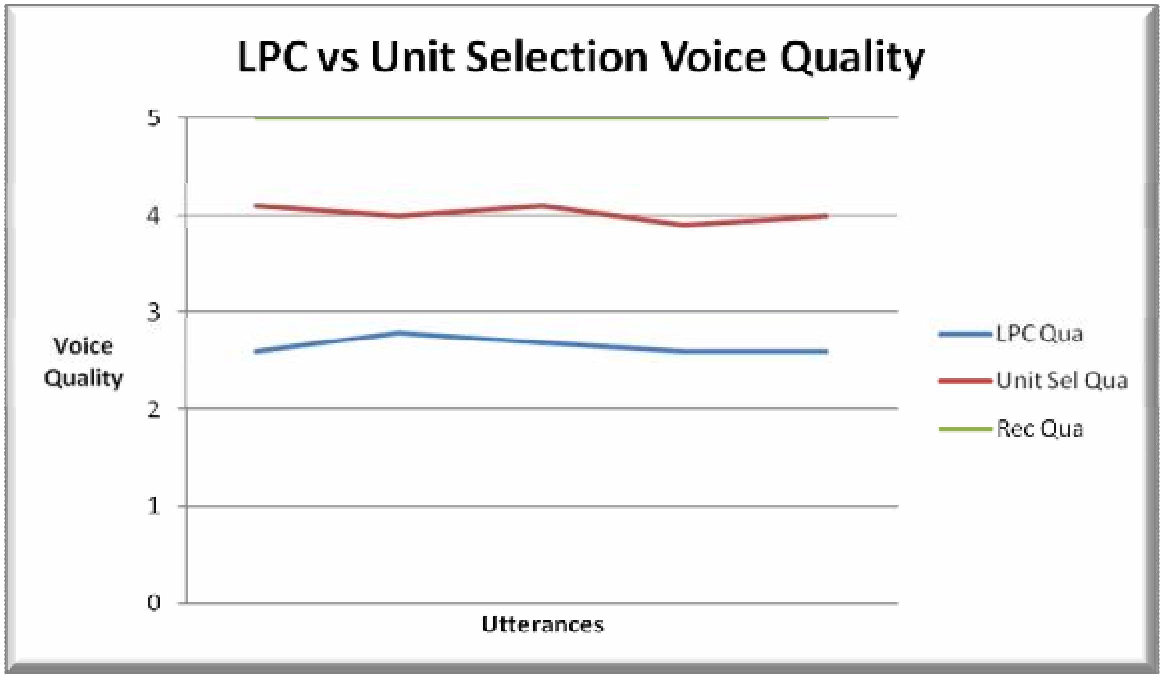


Figure 24. Evaluation Results (Voice Quality).

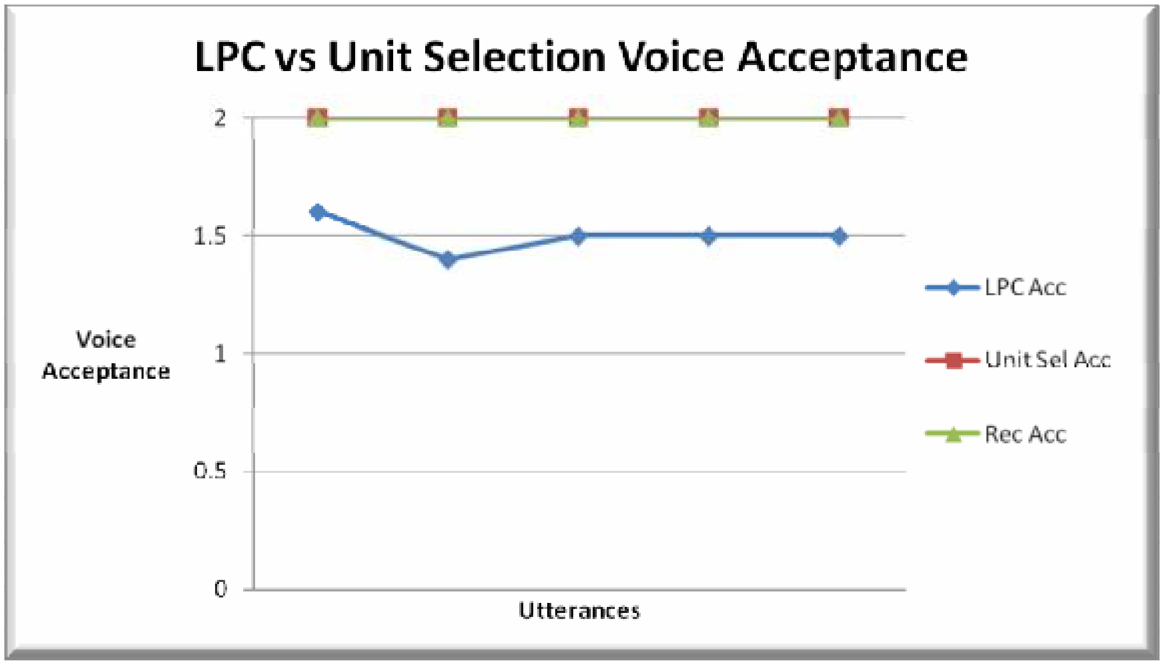


Figure 25. Evaluation Results (Voice Acceptance).

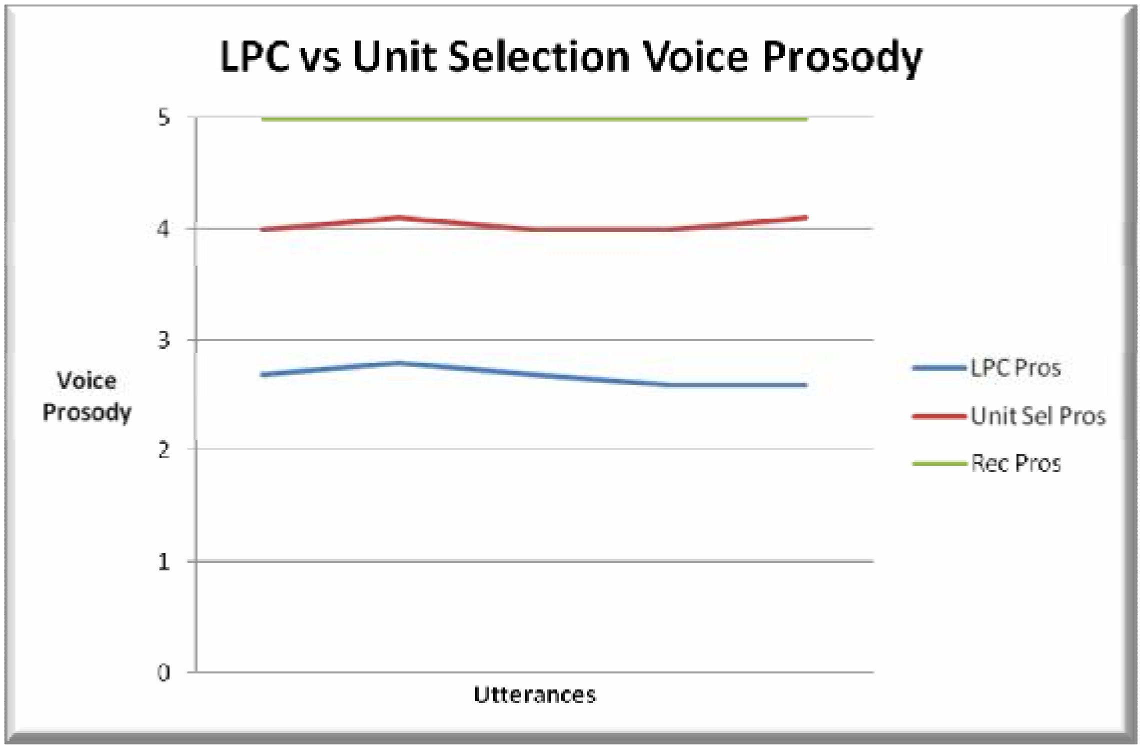


Figure 26. Evaluation Results (Prosody).

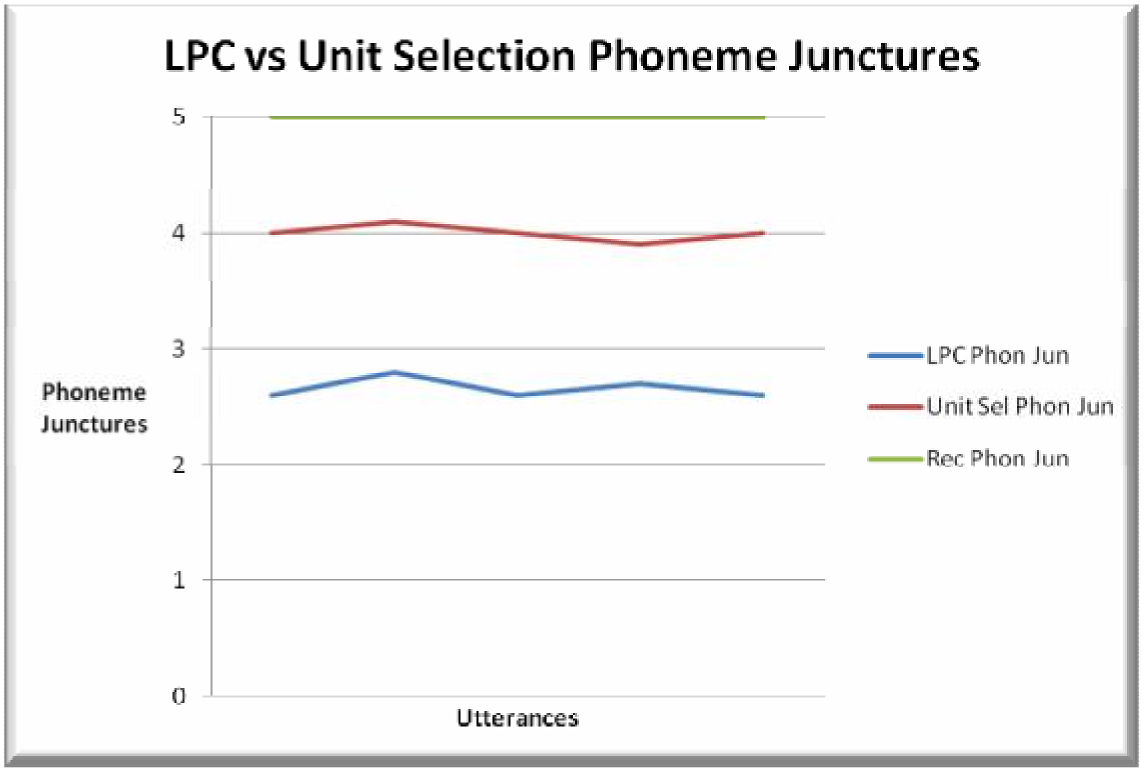


Figure 27. Evaluation Results (Phoneme Junctures).

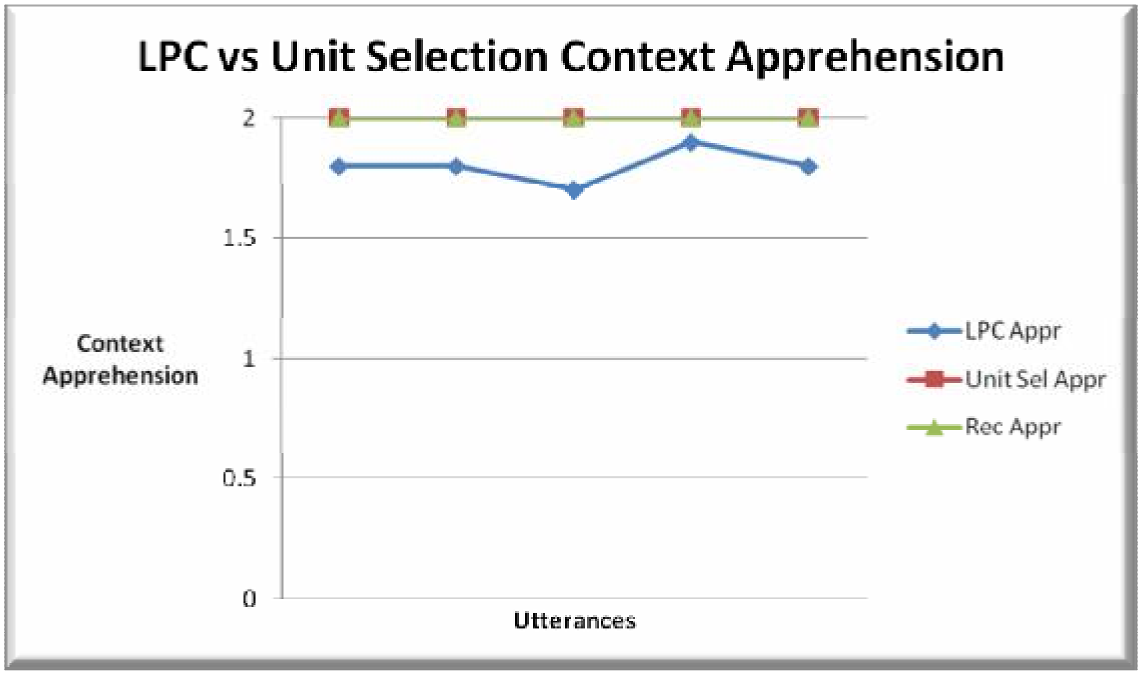


Figure 28. Evaluation Results (Context Apprehension).

The above figures show us that the performance of the LPC synthesis method is the same for a non-weather input as a weather input. The scores of LPC method still flunctuate between 1 and 2. Contrary to LPC method which performance is stable, Cluster Unit Selection presents a decrease in performance when the input is non-weather. This is due to the Cluster Algorithm itself. As we described in *Chapter 4 Section 2* each unit has two dimensions: its name and its type. The second dimension refers to the word that this unit comes from. As a result, if the word we want to synthesize does not exist in the training data of our synthesizer, the concatenation process becomes problematic. However, the synthesized sentence is totally comprehensible and the final result is decent.

Chapter 6

CONCLUSIONS & FUTURE WORK

6.1 Conclusions

Speech, or verbal communication, is one of the most important features which distinguish humans from other animals. Researchers in speech technology are still working on getting machines to interact with humans the same way human-to-human communication occurs. Human-computer interaction is a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use.

The main goal for speech technology research is to achieve communication between people and machines. By communication it is meant ability for machines to communicate with humans the way humans do with other people. Machines should be able to read and show facial expressions, body language and any other gestures used by humans. When humans are communicating with machines, they should have the same interest, concentration and emotion as they would have when talking to other people.

In this thesis, we tried to understand the rules of selecting phonetic units in a Text-To-Speech synthesis System. In order to accomplish that, we created a Greek Weather Forecast Synthesizer using two different methods of implementation: the Residual-Excited Linear Prediction Coding Synthesis and the Cluster Unit-Selection Synthesis.

From *Chapter 5 Section 5*, it can be concluded that in limited-domain synthesis (Weather) the Cluster Unit Selection Method produces a voice with high degree of understandability, naturalness and pleasantness that is required, while the LPC Diphone method produces a voice with unacceptable degree of understandability, naturalness and pleasantness. On the occasion of non-weather input, the LPC Diphone Method produces similar results. On the other hand, the Cluster Unit Selection Method, contrary to its previous results, produces a voice with worse degree of understandability and naturalness, but still better than the LPC Method's voice.

6.2 Future Work

The major task in speech synthesis technology is to produce a voice in unrestricted domains with the same quality of limited-domain synthesis. In order to accomplish that, the most significant task is the design of the proper corpus of training data. The selection of the utterances to be recorded, must be very assiduous in order to include the maximum range of words and as a result have the maximum language coverage.

The difference between a person and a talking computer is that the person understands the ideas and emotions conveyed through speech, while the computer doesn't. The ultimate goal for speech synthesis, as with all Artificial Intelligence applications, is to pass the Turing test - a blindfolded user should not be able to tell whether he is talking to a human or a machine. Of course, that is a long way away, but we believe that modifying speech recognition techniques could lead to better synthesis results. Ultimately, the right model might be the same both for recognition and synthesis.

Bibliography

[1] Black A. & Lazlo K. : "Building Voices in the Festival Speech Synthesis System" unpublished document, Language Technology Institute, Carnegie Mellon 2007 available at http://www.cstr.ed.ac.uk/projects/festival/docs/festvox/

[2] Black A., Taylor P. & Caley R.: "The Festival Speech Synthesis System Documentation", The Centre for Speech Technology, Research University of Edinburgh 2002 available at http://festvox.org/docs/manual-1.4.3/festival_toc.html

[3] Hunt A., Black A., "Unit Selection in a Concatenative Speech Synthesis System Using a Large Speech Database" *Proceedings of ICASSP-96, Atlanta, GA*, Vol. 1: 373-376, 1996.

[4] Dutoit, Th. *An Itroduction to Text-to-Speech Synthesis*, Dordrecht, Kluwer Academic Publishers, 1997.

[5] Taylor P., Black A., Caley R., "The Architecture of the Festival Speech Synthesis System" *3rd ESCA Workshop in Speech Synthesis*, 1998.

[6] Vosnidis C. "Speech Synthesis by Word Concatenation" Thesis Submitted for Diploma in Electronics and Computer Engineering, Technical University of Crete, Chania, Greece, 2001.

[7] Digalakis V. Introduction to Speech Processing, Chania: Technical University of Crete, Department of Electronics and Computer Engineering, 2006.

[8] Black, A, and Taylor, P. "Automatically clustering similar units for unit selection in speech synthesis", *Eurospeech97*, Rhodes, Greece, 1997.

[9] Breiman, L., Friedman, J. Olshen, R. and Stone, C. *Classification and regression trees*, Wadsworth and Brooks, Pacific Grove, CA. 1984.

[10] Mohasi L., Mashao D., "Text-To-Speech Technology in Human-Computer" Interaction", *5th Conference of Human-Computer Interaction*, Cape Town, South Africa, 2006.

[11] Y. Sagisaka, N. Kaiki, N. Iwahashi, and K. Mimura. “ATR -- nuu-TALK speech synthesis system”. In *Proceedings of ICSLP 92*, volume 1, pages 483--486, 1992.

[12] N. Campbell and A. Black. “Prosody and the selection of source units for concatenative synthesis”. In J. van Santen, R. Sproat, J. Olive, and J. Hirschberg, editors, *Progress in speech synthesis*, pages 279--282. Springer Verlag, 1996.

[13] M. Lieberman “Computer Speech Synthesis: Its Status and Prospects”, *Voice Communications between humans and machines,* Washington D.C., National Academy Press, 1994.

[14] T. Dutoit and H. Leich. MBR-PSOLA : Text-to-speech synthesis based on an MBE re-synthesis of the segments database. *Speech Communication*, 13:435--440, 1993.