ARABIC SPEECH SIGNAL PROCESSING

TEXT-TO-SPEECH SYNTHESIS

BY

MUSTAFA IBRAHIM YOUSIF

Supervisor

Dr. Mohammed Ali Abbas

REPORT SUBMITTED TO

University of Khartoum

In partial fulfillment of the requirement for the degree of

B.Sc. (HON) in Electrical and Electronics Engineering

(COMMUNICATION ENGINEERING)

Faculty of Engineering

Department of Electrical and Electronics Engineering

July 2011
DECLARATION OF ORIGINALITY

I declare that this report entitled “Arabic Speech Signal Processing, Text-To-Speech Synthesis” is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature: _________________________
Name: _________________________
Date: _________________________
ABSTRACT

Text-To-Speech synthesis (TTS) is one of the speech signal processing methods in which the ultimate goal is to produce as natural sounding speech as possible from arbitrary text. Arabic TTS synthesis has gained more attention in the near past for the educational and commercial fields and the support of the deaf-mute and visually and vocally handicapped. This report explains the methods and requirements for constructing a useful Arabic TTS synthesizer that generate a recognizable and acceptable quality speech. It also describes in details the concatenation method for constructing Arabic TTS synthesizer system in which the speech is produced using prerecorded speech units that are utilized by the system to produce concatenated speech. The system is developed in the visual C++ programming environment with many algorithms and functions to make it easy for future modifications and development. Lastly, Arabic TTS synthesizer is considered as intelligible; however, the system still requires further research and developments in many directions including handling various texts such as fractions, phone numbers, and mathematical and chemical equations which cannot be written in Arabic.
المستخلص

تركيب النص إلى كلام هو أحد طرق معالجة إشارة الكلام، حيث الهدف النهائي منه هو إصدار صوت طبيعي قدر الإمكان من أي نص تعسفي. تركيب إشارة كلام اللغة العربية لاقت إهتماماً زائداً في الماضي القريب في المجالات التعليمية والتجارية وتدعيم ذوي الاحتياجات الخاصة من الليم والعمى والمعاقين صوتياً. هذا التقرير يشرح أساليب ومطابقات بناء مركب إشارة كلام اللغة العربية الذي يولد خطاباً يمكن تعيينه وجودة قبوله. كما يشرح التفصيل الطريقة التسلسلية لبناء مركب كلام اللغة العربية الذي يتم إنتاجه باستخدام وحدات كلام مسجلة مسبقاً والتي تُستخدم من قبل النظام لإصدار كلام متسلسل. تم تطوير هذا النظام في بيئة Visual C++ البرمجية مع العديد من الخوارزميات والدوال لتسهيل التعديلات والتطوير في المستقبل. يعتبر مركب كلام اللغة العربية من النص من الوضوح يمكن؛ ولكن النظام لا يزال يتطلب المزيد من الأبحاث والتطويرات في إتجاهات عديدة بما في ذلك التعامل مع النصوص المختلفة مثل الكسور، وأرقام الهواتف، والمعادلات الرياضية والكيميائية التي لا يمكن أن تكتب باللغة العربية.
إهـداد

إلى من وهبت عمرها وحياتها لتربيتنا...
إلى الشمعة التي تضيء لتنير طريقنا...
امي الحبيبة...
إلى معلمي ومرببي...
إلى سندي وتنز رأسي...
أبي العز...
إلى أخ وزميلي...
إلى رفيق دربي...
مازن...
إلى مربي الأجاهل...
إلى صانعي العباقرة...
إلى معلمى الخير...
أساتذتي الإجلاء...
إلى أخواتي وأخواتي...
أحبائي في الله...
من كانوا دعما وسندا على طول الطريق...
دفعتي 464..
TABLE OF CONTENTS

DECLARATION OF ORIGINALITY i

ABSTRACT ii

المستخلص iii

TABLE OF CONTENTS v

LIST OF FIGURES viii

ABBREVIATIONS ix

GLOSSARY x

1  CHAPTER 1  INTRODUCTION 1

1.1  Introduction 1

1.2  Motivation 1

1.3  Problem Justification 2

1.4  Scope of the problem 3

1.5  Objectives 3

1.6  Outlines 3

2  CHAPTER 2  LITERATURE REVIEW 4

2.1  Introduction 4

2.2  Speech Production 5

2.3  Speech synthesis 7

2.3.1  History of speech synthesis 7

2.3.2  Source-Filter Theory 9

2.3.3  General TTS structure 10

2.3.4  Speech synthesis methods 11

2.4  Application of synthetic speech 15

2.4.1  Application for the blinds 15

2.4.2  Applications for the Deafened and Vocally Handicapped 16

2.4.3  Educational Applications 16

2.4.4  Applications for Telecommunications and Multimedia 16

2.4.5  Other application and future directions 17
2.5 Previous work

2.5.1 MBROLA Project

2.5.2 ACAPELA – GROUP

2.5.3 ARABTALK

2.5.4 SAKHR TTS

3 CHAPTER 3 DESIGN AND IMPLEMENTATION

3.1 Concatenative Synthesis

3.1.1 OVERVIEW

3.1.2 Details of Concatenative Synthesis

3.1.3 Speech Signal Representation for Concatenative Synthesis

3.1.4 Unit Selection Synthesis

3.1.5 Types of Concatenative Synthesis

3.1.6 Concatenative Synthesis: Pros and Cons

3.1.7 Requirements for a Concatenative Synthesis System

3.2 Arabic TTS System Model

3.3 Specifications of Processing Stages

3.3.1 Text and Linguistic Analysis (High-level Synthesis)

3.3.2 Speech (Waveform) Generation (Low-level Synthesis)

3.4 The general process of the program

3.5 Special cases

3.6 Software & Tools

3.7 Special implementation problems and their solutions

4 CHAPTER 4 RESULTS AND DISCUSSION

4.1 Introduction

4.2 TESTING AND EVALUATING TTS SYSTEMS

4.2.1 Intelligibility

4.2.2 Naturalness

4.2.3 Sound quality

4.2.4 Pronunciation
5  **CHAPTER 5  CONCLUSION AND FUTURE WORK**  51

5.1  Conclusion  51

5.2  Future work  52

**REFERENCES**  54

**APPENDIX A**  A-1

**APPENDIX B**  B-2

**APPENDIX C**  C-1

**APPENDIX D**  D-1
LIST OF FIGURES

Figure 2.1: methods of speech signal processing .......................................................... 4
Figure 2.2: Speech production organs ........................................................................... 6
Figure 2.3: Wheatstone's reconstruction of von Kempelen's speaking machine ............. 8
Figure 2.4: Source-filter model of speech production .................................................... 9
Figure 2.5: Spectrum of the component in the source-filter theory .............................. 10
Figure 2.6: General functional diagram of a TTS system ........................................... 11
Figure 2.7: Speech synthesis model based on the source-filter theory ....................... 12
Figure 2.8: Overview of the HMM-based text-to-speech system ................................. 14
Figure 3.1: Block diagram of a concatenative text-to-speech system ......................... 21
Figure 3.2: Full Model of Arabic TTS System ............................................................... 27
Figure 3.3: Functions of Structure Analysis module .................................................... 29
Figure 3.4: Structure Analysis Flowchart .................................................................. 30
Figure 3.5: Arabic Diacritization module .................................................................... 31
Figure 3.6: Arabic Diacritization Flowchart ................................................................. 32
Figure 3.7: Classification charts of Arabic phonemes .................................................. 33
Figure 4.1: The final form of the program .................................................................. 43
Figure 4.2: intelligibility 1st result .............................................................................. 44
Figure 4.3: intelligibility 2nd result .............................................................................. 45
Figure 4.4: naturalness result ...................................................................................... 46
Figure 4.5: sound quality result ................................................................................... 46
Figure 4.6: pronunciation 1st result ............................................................................. 47
Figure 4.7: pronunciation 2nd result ........................................................................... 48
Figure 4.8: pronunciation 3rd result ............................................................................ 48
Figure 4.9: pronunciation 4th result ............................................................................. 49
ABBREVIATIONS

TTS  Text-To-Speech
OCR  Optical Character Recognizer
TTP  Text-To-Phoneme
GTP  Grapheme-To-Phoneme
VLSI Very Large Scale Integration
Bps  Bits per Second
A-To-D Analog-To-Digital
D-To-D Digital-To-Analog
VoIP Voice over Internet Protocol
HF   High Frequency
DSP  Digital Signal Processing
NLP  Natural Language Processing
ASCII the American Standard Code for Information Interchange
IPA  International Phonetic Association
PSOLA Pitch Synchronous Overlap Add
TD-PSOLA Time Domain Pitch-Synchronous Overlap Add
HMM  Hidden Markov Model
LPC  Linear Predictive Coding
FIFO  First In First Out
API  Application Programming Interface
SAPI Speech Application Programming Interface
GLOSSARY

**Articulation** - It is the approach or contact of two speech organs, such as the tip of the tongue and the upper teeth.

**Consonant** - A consonant is any articulation that is acoustically produced by forming a constriction by one or more articulators along the vocal tract to the flow of air from the lungs. A long-duration consonant is a syllable closing sound, while a short-duration consonant occurs between two vowels.

**Diphone** - A diphone is a sound that starts from the middle of one phoneme and ends in the middle of a neighboring phoneme.

**Formant** - A formant is the resonating frequency of the air in the vocal tract.

**Intonation** - Intonation is the pattern of fundamental frequency changes that occur during a phrase or a sentence.

**Diacritics** in Arabic are optional orthographic symbols typically representing short vowels.

**morphology** (linguistics) - is the identification, analysis and description of the structure of morphemes and other units of meaning in a language like words, affixes, and parts of speech and intonation/stress.

**Linguistics** - The scientific study of language, which may be undertaken from many aspects, for ex: - structure of words (morphology), meanings (semantics) etc.

**Prosody** - The study of speech-rhythms.

**Phoneme** - The smallest unit of sound in a language. It is an abstract unit that represents sounds and writing in a systematic, unambiguous way.

**Phonetics** - Phonetics is the representation of the sounds of a language.

**Phonology** - It is the description of the systems and patterns of sounds that occur in a language.

**Syllable** - A syllable is a word or a part of a word that is uttered by a single effort of the voice. A syllable is structured from phonemes that are consonants and vowels. A vowel is usually the syllable nucleus while a consonant usually represents the syllable margin.

**Synthesis unit** - A synthesis unit is an acoustical entity that represents a complete or partial sound. These units join to form syllables, words, or phrases.

**Vocal Tract** - The vocal tract is the air passage above the larynx. It consists of the oral and nasal tracts.

**Vowel** - A vowel consists of a single vocal sound that is produced by a continuous passage of air from the lungs on an open vocal tract. This definition applies to both short and long vowels.
CHAPTER 1
INTRODUCTION

1.1 Introduction

Speech is a complex signal from which we extract many types of information, from the message content and meaning, to the nature of the transmission medium, to the identity and condition of the speaker.

Speech signal processing refers to the acquisition, manipulation, storage, transfer and output of vocal utterances by a computer. Speech signal processing can be divided into several types according to its definition; these types are speech recognition which focuses on capturing the human voice as a digital sound wave and converting it into a computer-readable format, speech synthesis; and it the reverse operation of recognition, speaker recognition which is the computing task of validating a user's claimed identity using characteristics extracted from their voices, also it include Speech compression that is important in the telecommunications area for increasing the amount of information which can be transferred, stored, or heard, for a given set of time and space constraints.

A TTS synthesizer System is a computer-based system that has the ability to read any text aloud, whether it is directly introduced in the computer by an operator or scanned and submitted to an Optical Character Recognition (OCR) system. OCR is the process that allows the transformation of a string of phonetic/syllabic and prosodic symbols into a synthetic signal, (i.e. the automatic production of speech, through a Grapheme-To-Phoneme transcription of the sentences to utter). Grapheme is the letters in a words’ dictionary, while, Phoneme is the smallest unit of speech that differentiates one word from another [1].

1.2 Motivation

TTS synthesis system has gain a lot of attention in the last few years for that it could be useful in many ways in different fields, e.g. educational; for the nonnative speakers, the support for the deaf-mute and the visually handicapped, economical; because converting fundamental data stored in Web sites, databases and files into human voice using the traditional expensive and time-consuming human recordings in studios is becoming a hard and long process since the information is usually dynamic. English TTS synthesis systems in addition to the rest of the speech processing methods are now integrated in various devices and applications such as
operating systems, security devices, and many software programs. Arabic TTS synthesizers in the other hand must be developed by its native speakers otherwise it will not exist (i.e. no one will develop your language's system for you). The points stated above are the main reasons behind developing an Arabic TTS synthesizer.

1.3 Problem Justification

Although a lot of TTS systems for different or the same language have been developed, these systems differ in their structure because the goal behind developing each one vary from the other, and the same goes for Arabic TTS systems.

TTS systems convert the input text into sound using one of several techniques that will be described briefly in chapter 2. Many different rules and exceptions are required to produce correct pronunciation for synthesized speech. The difficulties facing this project begin with the plain text pre-processing in which we have several problems - language dependent - such as numerals, abbreviations and acronyms.

Text-To-Phonetic (TTP) also called Grapheme-To-Phoneme (GTP) conversion is the first problem arises in any TTS system. Text processing is usually very complex and language dependent. Some of these problems will be discussed for Arabic language. Numerals must be converted to words e.g. "123" will be converted to "مئة وثلاثة وعشرون" meaning "one hundred and twenty three". Dates, e.g. "2000/11/23" will be converted to "الثلاثة وعشرون من شهر نوفمبر من سنة" meaning "the day number twenty three of the month November in the year two-thousand".

Time, e.g. "2:45" will be converted to "الساعة الثانية وخمسة وأربعون دقيقة مساء" meaning "two o'clock and forty five minutes in the evening". Abbreviations in Arabic do not consider as problematic as in English. In English some abbreviations are read letter-by-letter e.g. "USA for United State of America", "KSA for Kingdom Saudi Arabia", others are read as one full word e.g. "UNICEF for United Nations Children's Emergency Fund" and there are no rules to restrict the two kind of reading. In Arabic if the abbreviations are written with separate letters that means they will be read letter-by-letter e.g. "قش" abbreviation for " قوات الشعب المسلحة". some abbreviations are written in one word but they never been spoken and instead they are replaced by the original words e.g. "كيلو جرام" for "كجم".

The second task faced by any TTS synthesizer system is to find correct pronunciation for different contexts in the text. Homographs which are the words that have the same spelling but differ in the meaning and usually in the pronunciation are one of the major problems in TTS systems, e.g. "يزيد" meaning "Yazeed is coming" and "يزيد الماء" meaning "the seller but water in the milk" in the first sentence "يزيد" is a name and in the second one is a verb.
Solving all language problems is nearly impossible for one researcher, thus a group of special cases and was selected and carried out through the project phases.

1.4 Scope of the problem

The Arabic TTS Synthesizer System is a stand-alone and event-based system. The scope of this dissertation looks into a few dimensions. These dimensions can be divided as follows. The TTS Synthesizer System is able to convert text to audio format using Arabic and English languages. That is the system supports only two languages. Additionally, the Arabic TTS Synthesizer System is able to pronounce the text input by the user, in a form of word-by-word, or in a form of a sentence. Arabic TTS Synthesizer System designed to be used by non-Arabic speaking, audiences, teachers -teachers may use this system to teach their students on the correct pronunciation- and disabled or impaired users.

1.5 Objectives

TTS systems are now available for many languages, and Arabic systems have been out of view until recently, thus the area of Arabic TTS Systems is still in its early stages. The objectives of this project are:

1. Obtain knowledge about speech synthesis.
2. Make a contribution in the development of Arabic TTS systems.
3. Design and implementation of an Arabic TTS synthesizer.

1.6 Outlines

The report is organized as follows:

**Chapter 2**: represent an overview of speech production, speech signal processing and TTS system methods and applications as well as the previous existing products.

**Chapter 3**: explains in details the objectives of the project and focus on how these objectives were achieved. Also, it presents the design methodology, components description, in addition to the methods, and tools used to implement the system.

**Chapter 4**: follows the results obtained and explains it.

**Chapter 5**: summarizes the project and gives ideas for future developments.
CHAPTER 2
LITERATURE REVIEW

2.1 Introduction

Speech is no doubt the most essential medium of human interaction. By means of modern digital signal processing (DSP), we can interact, not only with others, but also with machines. The importance of speech/audio signal processing lies in preserving and improving the quality of speech/audio signals. These signals are treated in a digital representation where various advanced digital-signal-processing schemes can be carried out adaptively to enhance the quality.

The advances of the VLSI technology have allowed the development of high performance DSP devices, enabling the implementation of very efficient and sophisticated algorithms, which have been successfully used in the solution of a large amount of practical problems in several fields of science and engineering such as echo problem in teleconference and inter symbol interference in high speed data communication [2].

As mentioned earlier speech signal processing is divided into several categories. Figure 2.1 illustrate these categories.

![Figure 2.1: methods of speech signal processing](image)

There are many situations where speech is not the best method for communicating with machines. For example, large amounts of text are much more easily received by reading from a screen, and positional control of features in a computer-aided design system is easier by direct
manual manipulation. However, for interactive dialogue and for input of large amounts of text or numeric data speech offers great advantages [3]. For all applications where the machine is only accessible from a standard telephone instrument there is no practicable alternative.

Several researches are being carried out in the area of speech synthesis; the method of speech processing that this report handle. In the early days of synthesis, research efforts were devoted mainly in simulating human speech production mechanism, using basic articulator models, which was based on electronic acoustic theories. Even though this way of modeling is one of the ultimate goals of synthesis research, advances in computer sciences have widened the research field to include text to speech processing. It is not only human speech generation is modeled but also text processing is modeled. This way of modeling is generally done by a set of rules derived, for example, from phonetic theories and acoustic analysis.

2.2 Speech Production

We use language every day without devoting much thought to the process, but articulatory movement -the movement of the lips, tongue, and other organs- is among the subtlest and most adept of any actions performed by human beings[4].

Speech is a natural form of communication for human beings, and computers with the ability to understand speech and speak with a human voice are expected to contribute to the development of more natural man-machine interfaces. Computers with this kind of ability are gradually becoming a reality, through the evolution of speech synthesis and speech recognition technologies. However, in order to give them functions that are even closer to those of human beings, the mechanisms by which speech is produced and perceived must be learned, and develop speech information processing technologies that make use of these functions. We use speech every day almost unconsciously, but an understanding of the mechanisms on which it is based will help to clarify how the brain processes information and will also lead to the development of more human-like speech devices through the imitation of these functions by computers.

The human speech production system is an interesting and complex mechanism. Initially, every person has vocal characteristics that are unique so that one can often recognize an individual by their voice alone. These characteristics directly relate to the physiology of the talker [4]. Features such as age, gender, height, weight, and the structure of the vocal chords, nasal and oral cavities, teeth and lips all play a major role in the speech production process.
Speech is produced by regulating the airflow from lungs through throat, nose and mouth. The air in the lungs is pressed upon chest and lung tissues, resulting in airflow to trachea and larynx. At larynx the airflow is modulated by vocal folds, which creates the main excitation for voiced speech. Pharynx connects the larynx to oral and nasal cavities, which are collectively called the vocal tract. The volume and dimensions of the pharynx and oral cavity can be adjusted, functioning as an acoustic time-varying filter. Finally sound is radiated to surrounding air at lips and nostrils [5]. The speech production main organs are shown in Figure 2.2.

![Speech production organs](image)

**Figure 2.2: Speech production organs**

Where:

1. Nasal cavity (حجٌٕف الأَف).
2. Hard palate (الحلب الصلب).
3. Alveolar ridge (الحرف السنخي).
5. Tongue tip (طرف اللسان).
6. Dorsum (ظهر اللسان).
7. Uvula (لِعَبَة الحلق).
8. Radix (الجذر).
10. Epiglottis (لِعَبَة).
11. False vocal folds (الحبال الصوتية الكاذبة).
12. Vocal fold (الحبال الصوتية).
13. Larynx (الحنجرة).
15. Trachea (القصبة الهوائية).

The produced speech sounds can be basically classified into three categories. Firstly, voiced speech sounds are produced by using the air pressure to get the vocal folds into vibratory motion. This generates a periodic signal rich in harmonics. Secondly, unvoiced sounds are produced by constricting the airflow somewhere in the vocal tract. This creates a continuous turbulent airflow characterized by a noise-like waveform without a harmonic structure. The continuous unvoiced speech sounds are called fricatives. Thirdly, unvoiced stop consonants are produced by completely stopping the airflow in the vocal tract. The release of the increased pressure creates a transient noise burst. Speech sounds are often a combination of both voiced and unvoiced components [5].

2.3 Speech synthesis

Speech synthesis is the artificial generation of speech and it has various useful applications, such as telecommunication services, man-machine communication, language education, aid to persons with disabilities, research on speech production and perception, and many others. Depending on the application, different implementations of a speech synthesizer may be used. Today a text-to-speech (TTS) system is maybe the most common and the most versatile solution. The ultimate goal of such a system is to read any text and convert it to speech. However, there are various criteria for evaluating the resulting speech or the system as a whole, and various approaches can be used to meet the required specifications.

2.3.1 History of speech synthesis

The earliest attempts to produce artificial speech were made more than two hundred years ago [6]. The early mechanical implementations of speech synthesizers modeled the physiology of the speech production organs. For example, in 1791 von Kempelen presented a speaking machine which consisted of bellows, a vibrating reed, and a rubber tube modeling the vocal tract as shown in Figure 2.3.
As the electrical technology evolved, interest in speech synthesis increased. The first formant synthesizer was built by Stewart in 1922 [7]. It consisted of two resonant circuits, which were excited by a buzzer. This early synthesizer was able to generate a static vowel with two lowest formants. The first electrical device that could produce continuous speech was the Voder developed at the Bell Telephone Laboratories in 1939. It was based on the idea of a vocoder, a voice coder, which could analyze speech into slowly varying parameters and then reconstruct an approximation of the original speech from the parameters. The first dynamically controlled formant synthesizers were introduced in 1953. Walter Lawrence’s PAT and Gunnar Fant’s OVE I, and especially their improved later versions, could generate intelligible speech. Shortly after that the first articulatory speech synthesizer was introduced in 1958. Consequently, the speech analysis and synthesis techniques split into two paradigms: modeling of the speech production mechanism itself, and modeling only the speech signal [8]. This division stands even today, though much co-operation exists between the fields. The first full text-to-speech system was developed in 1968 by Noriko Umeda, and in 1972 John Holmes demonstrated that synthetic speech could be so natural sounding that the average person could not tell the difference between the synthetic and the original sentence [7]. Since the late 1970s, many commercial speech synthesis and text-to-speech products have been introduced, with MITalk [7] being probably the best known TTS system. In the mid-1980s the concept of high quality TTS synthesis appeared, mostly due to new technologies. Modern synthesizers have largely moved from electronic circuitry to simulation on a digital computer. The methods used in speech synthesis technology today are very sophisticated as the latest findings from research on information technology,
signal processing, acoustics, speech production, and linguistics are applied directly to speech synthesizers. The quality of speech synthesis has improved to a level of great intelligibility, but the naturalness is yet a problem. Nevertheless, more natural sounding speech synthesizers are constantly developed based on various different methods. In the next sections, TTS architecture and speech synthesis methods are considered in more details.

### 2.3.2 Source-Filter Theory

The source-filter theory of speech production states that speech signal can be represented in terms of source and filter characteristics. In human speech production the primary sound source is the excitation of the vibrating vocal folds. The periodic vibration generates a rich harmonic spectrum, whose energy declines with increasing frequency. The vocal tract modifies the excitation spectrum with a transfer function with formants or antiformants [APPENDIX C]. Finally the sound radiates to the surrounding air at lips and nostrils. This causes a frequency dependent effect called lip radiation, which acts as a high-pass filter. The source-filter theory is summarized in Figure 2.4 and Figure 2.5.

![Source-filter model of speech production](image)

**Figure 2.4: Source-filter model of speech production**

Where:

(a) Speech is initiated by the vibrations of the vocal folds. This generates a rich periodic spectrum, which energy declines with increasing frequency.
(b) The vocal tract modifies the glottal excitation by creating resonances.
(c) Spectrum of the signal before lip radiation.
(d) The radiation of sound from lips and nostrils to surrounding air creates an effect that enhances the higher frequencies of the signal. This is called the lip radiation.
(e) The spectrum of the speech signal.

![Spectrum of the component in the source-filter theory](image)

**Figure 2.5: Spectrum of the component in the source-filter theory**

Where:
(a) Spectrum of the glottal excitation.
(b) Amplitude response of the vocal tract filter.
(c) Amplitude spectrum of the lip radiation.
(d) Spectrum of the speech signal.

The source-filter theory is a linear mathematical model with many simplifying assumptions. Therefore some aspects of the theory are not really valid. For instance, it has been observed that interaction between source and vocal tract can occur in natural speech. Moreover, the all-pole model is not perfectly appropriate for modeling antiformants, which are present in nasal sounds. However, the (all-pole) source-filter model is sufficient for most applications since the benefits of the linear model are much greater than the disadvantages [9].

### 2.3.3 General TTS structure

The system is called a text-to-speech (TTS) synthesizer, if the input to a speech synthesizer is given as text. However, in the case of speech synthesizers with limited vocabulary, such as machines playing prerecorded samples, the definition is not unambiguous. According to the more specific definition, text-to-speech means "the production of speech by machines, by
way of the automatic phonetization of utter” [10]. A general functional diagram of a TTS system is shown in Figure 2.6.

![Diagram](image)

**Figure 2.6: General functional diagram of a TTS system**

A TTS synthesizer consists of two main components, called the high-level and low-level synthesis. The high level synthesis converts the text input to a form that corresponds to the desired acoustic phonation of the utterance. This means converting the text input into a phonetic or some other linguistic representation and predicting the desired prosody. In the process, the input text is first normalized into plain letters, and the structural properties of the text are analyzed. After that, the text is converted to a phonetic level, which is called the letter-to-sound conversion. Varying amount of linguistic analysis is performed on the text in order to predict the prosodic features of the utterance, such as phrasing and accentuation patterns. Based on the prosodic analysis and the structural information, actual f0 contour and phone durations are predicted for the utterance, typically using statistical methods. From the linguistic and prosodic information, the low-level synthesis generates the speech waveform. For the waveform generation, today’s TTS systems commonly employ techniques based either on the source-filter theory or modification and concatenation of prerecorded speech samples.

### 2.3.4 Speech synthesis methods

Once the high-level synthesis of a TTS system has completed its task, the low-level synthesis starts generating the speech waveform. The waveform generation can be accomplished in many ways, and the synthesis methods can be categorized according to various criteria. A basic division can be made according to whether the speech is completely artificially generated from parameters, or are real speech samples used in the process. This property greatly affects the functioning of the synthesizer. Formant synthesis [11], articulatory synthesis [12], and linear predictive coding (LPC) based synthesis [9] can be placed to the first category, whereas concatenative synthesis belongs to the latter.
2.3.4.1 Formant Synthesis

The most basic acoustic speech synthesis technique, formant synthesis, employs the source-filter theory of speech production. The vocal tract model consists of individually adjustable formant filters connected in serial, parallel, or often both. Different phonemes are constructed by adjusting the center frequency, bandwidth, and gain of each filter. If the adjustment is made at certain time intervals, for example every 5 ms, continuous speech can be generated. The source can be modeled with voice pulses or noise. A basic speech synthesis model based on the source-filter theory is shown in Figure 2.7.

![Figure 2.7: Speech synthesis model based on the source-filter theory](image)

Formant synthesis received a big boost in 1980 with Dennis Klatt’s publication of a sophisticated formant synthesizer with a complete computer program for speech synthesis. Today, the quality of formant synthesizers is inferior compared to the latest synthesis methods, such as concatenative and LPC-based methods, but formant synthesis has many applications in reading machines for the blind and in speech perception experiments for creating stimuli [5].

2.3.4.2 Articulatory Synthesis

Articulatory synthesis tries to model the natural speech production process as accurately as possible. Thus it is theoretically the best method for high quality speech synthesis, but it is also by the same token the most difficult in terms of implementation and computational load. Because of the limitations of the current speech production models and computational power, articulatory synthesis has not achieved as much success as other speech synthesis methods. However, it has many useful applications in basic speech research, and it might have a promising future since better articulatory models are steadily developed and computational resources are increasing [5].
2.3.4.3 Concatenative Synthesis

In concatenative synthesis, prerecorded samples of real speech are smoothly combined to create an arbitrary synthetic utterance. Common unit lengths are word, syllable, demisyllable, phoneme, diphone, and triphone. Because the natural characteristics of the speech are preserved in the units, concatenative synthesis is capable of generating highly intelligible and natural synthetic speech. However, the discontinuities in points can cause distortion despite the use of various smoothing algorithms. Also, the set of speech units is always limited. It is highly impractical or impossible to store all the necessary units for various speakers in various contexts. This constraint makes the concatenative speech synthesis less flexible: it can imitate the specific speaker with only one voice quality. Another constraint is the need for vast storage for all the recorded units, but with the cost of computer storage decreasing, and with the development of fast database access techniques, this problem is not as serious as it used to be. Today the concatenative speech synthesis is probably the most widely used and most natural sounding, but due to the mentioned limitations, it might not be the best solution [5]. More details about the concatenation method are explained in chapter 3.

2.3.4.4 LPC-Based Synthesis

In linear predictive coding (LPC) based speech synthesis, source-filter theory of speech production is utilized the same way as in formant synthesis, but in the LPC-based synthesis the filter coefficients can be automatically estimated from a short frame of speech instead of finding the parameters for individual formant filters. With an appropriate excitation, the filter coefficients can be used to synthesize speech. The excitation is either periodic source signal or noise, depending on whether the synthesized speech segment is voiced or unvoiced. Linear prediction (LP) is a widely used method in speech technology. Though the quality of a basic LPC vocoder is considered poor, high quality synthetic speech can be produced with more sophisticated LPC-based synthesis methods [5].

2.3.4.5 HMM-Based Synthesis

One widely applied method in speech synthesis is the use of hidden Markov models (HMMs). HMM is a statistical model, which can be used for modeling the speech parameters extracted from a speech database, and then generating the parameters according to text input for creating the speech waveform.

HMM-based speech synthesis systems are able to produce speech in different speaking styles with different speaker characteristics and even emotions. They also benefit from better adaptability and clearly smaller memory requirement. However, the HMM-based TTS systems
often suffer from degraded naturalness in quality compared to concatenative based speech synthesizers. Nonetheless, the HMM-based TTS systems are developing fast, and much work is carried out for finding techniques to enhance the quality and naturalness of synthetic speech. The current prevalent platform for HMM-based speech synthesis is the HTS system developed in Japan. It is widely used among speech synthesis researchers and developers, and lately numerous HMM-based TTS systems have been introduced for various languages. Figure 2.8 shows an overview of the HMM-based text-to-speech system.

![Figure 2.8: Overview of the HMM-based text-to-speech system](image)

Table 2.1 shows in brief the main difference between the concatenative and parameter types of speech synthesis (Formant and Articulatory).

<table>
<thead>
<tr>
<th></th>
<th>Concatenative synthesis</th>
<th>Parameter type synthesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic</td>
<td>Human-voiced fragments</td>
<td>Algorithm</td>
</tr>
<tr>
<td>Quality</td>
<td>More natural</td>
<td>More synthetic</td>
</tr>
<tr>
<td>Prosody</td>
<td>Lower</td>
<td>Higher</td>
</tr>
<tr>
<td>Memory requirement</td>
<td>Higher</td>
<td>Lower</td>
</tr>
<tr>
<td>Algorithm</td>
<td>Splice phoneme</td>
<td>Model vocal tracts</td>
</tr>
<tr>
<td>Effort to develop new voice language</td>
<td>Higher</td>
<td>Lower</td>
</tr>
</tbody>
</table>

Table 1: comparison between concatenative and parameter synthesis
2.4 Application of synthetic speech

Synthetic speech may be used in several applications. Communication aids have developed from low quality talking calculators to modern 3D applications, such as talking heads. The implementation method depends mostly on used application. In some cases, such as announcement or warning systems, unrestricted vocabulary is not necessary and the best result is usually achieved with some simple messaging system. With suitable implementation some funds may also be saved. On the other hand, some applications, such as reading machines for the blind or electronic-mail readers, require unlimited vocabulary and a TTS system is needed.

The application field of synthetic speech is expanding fast whilst the quality of TTS systems is also increasing steadily. Speech synthesis systems are also becoming more affordable for common customers, which makes these systems more suitable for everyday use. For example, better availability of TTS systems may increase employing possibilities for people with communication difficulties.

2.4.1 Application for the blinds

Probably the most important and useful application field in speech synthesis is the reading and communication aids for the blind. Before synthesized speech, specific audio books were used where the content of the book was read into audio tape. It is clear that making such spoken copy of any large book takes several months and is very expensive. It is also easier to get information from computer with speech instead of using special bliss symbol keyboard, which is an interface for reading the Braille characters.

The first commercial TTS application was probably the Kurzweil reading machine for the blind introduced by Raymond Kurzweil in the late 1970's. It consisted of an optical scanner and text recognition software and was capable to produce quite intelligible speech from written multifont text [7]. The prices of the first reading machines were far too high for average user and these machines were used mostly in libraries or related places. Today, the quality of reading machines has reached acceptable level and prices have become affordable for single individual, so a speech synthesizer will be very helpful and common device among visually impaired people in the future. Current systems are mostly software based, so with scanner and OCR system, it is easy to construct a reading machine for any computer environment with tolerable expenses. Regardless of how fast the development of reading and communication aids is, there are always some improvements to do.
2.4.2 Applications for the Deafened and Vocally Handicapped

People who are born-deaf cannot learn to speak properly and people with hearing difficulties have usually speaking difficulties. Synthesized speech gives the deafened and vocally handicapped an opportunity to communicate with people who do not understand the sign language. With a talking head it is possible to improve the quality of the communication situation even more because the visual information is the most important with the deaf and dumb. A speech synthesis system may also be used with communication over the telephone line [7].

Adjustable voice characteristics are very important in order to achieve individual sounding voice. Users of talking aids may also be very frustrated by an inability to convey emotions, such as happiness, sadness, urgency, or friendliness by voice. Some tools, such as HAMLET (Helpful Automatic Machine for Language and Emotional Talk) have been developed to help users to express their feelings. The HAMLET system is designed to operate on a PC with high quality speech synthesizer, such as DECTalk (developed by Digital Equipment Corporation).

2.4.3 Educational Applications

Synthesized speech can be used also in many educational situations. A computer with speech synthesizer can teach 24 hours a day and 365 days a year. It can be programmed for special tasks like spelling and pronunciation teaching for different languages. It can also be used with interactive educational applications.

Especially with people who are impaired to read (dyslexics), speech synthesis may be very helpful because especially some children may feel themselves very embarrassing when they have to be helped by a teacher [7]. It is also almost impossible to learn write and read without spoken help. With proper computer software, unsupervised training for these problems is easy and inexpensive to arrange.

A speech synthesizer connected with word processor is also a helpful aid to proof reading. Many users find it easier to detect grammatical and stylistic problems when listening than reading. Normal misspellings are also easier to detect.

2.4.4 Applications for Telecommunications and Multimedia

The newest applications in speech synthesis are in the area of multimedia. Synthesized speech has been used for decades in all kind of telephone enquiry systems, but the quality has been far from good for common customers. Today, the quality has reached the level that normal customers are adopting it for everyday use.
Electronic mail has become very usual in last few years. However, it is sometimes impossible to read those E-mail messages when being for example abroad. There may be no proper computer available or some security problems exist. With synthetic speech e-mail messages may be listened to via normal telephone line. Synthesized speech may also be used to speak out short text messages (SMS) in mobile phones.

For totally interactive multimedia applications an automatic speech recognition system is also needed. The automatic recognition of fluent speech is still far away, but the quality of current systems is at least so good that it can be used to give some control commands, such as yes/no, on/off, or ok/cancel.

2.4.5 Other application and future directions

In principle, speech synthesis may be used in all kind of human-machine interactions. For example, in warning and alarm systems synthesized speech may be used to give more accurate information of the current situation. Using speech instead of warning lights or buzzers gives an opportunity to reach the warning signal for example from a different room. Speech synthesizer may also be used to receive some desktop messages from a computer, such as printer activity or received e-mail.

In the future, if speech recognition techniques reach adequate level, synthesized speech may also be used in language interpreters or several other communication systems, such as videophones, videoconferencing, or talking mobile phones. If it is possible to recognize speech, transcribe it into ASCII string, and then resynthesize it back to speech, a large amount of transmission capacity may be saved. With talking mobile phones it is possible to increase the usability considerably for example with visually impaired users or in situations where it is difficult or even dangerous to try to reach the visual information. It is obvious that it is less dangerous to listen than to read the output from mobile phone for example when driving a car.

During last few decades the communication aids have been developed from talking calculators to modern three-dimensional audiovisual applications. The application field for speech synthesis is becoming wider all the time which brings also more funds into research and development areas.

2.5 Previous work

This section introduces some of the commercial TTS Synthesis Systems available today. More than 28 TTS Synthesizer Systems currently existing in the market. Some of the text in this
section is based on information collected from Internet, fortunately, mostly from the manufacturers and developers official homepages.

First commercial TTS Synthesis Systems were mostly hardware based and the developing process was very time-consuming and expensive. Since computers have turned out to be more and more powerful, currently most synthesizers are software-based systems. Software based systems are easy to configure and update, and usually they are much less expensive than the hardware systems. However, a stand-alone hardware device may still be the best solution when a portable system is needed.

2.5.1 MBROLA Project

MBROLA-project is one of the main systems that have an Arabic voice. The MBROLA project was initiated by the TCTS Laboratory in the Faculté Polytechnique de Mons, Belgium. The main goal of the project is to have a speech synthesis for as many languages as possible. MBROLA is used for non-commercial purposes. Another purpose with it is to increase the academic research, especially in prosody generation.

The MBROLA speech synthesizer is based on diphone concatenation. MBROLA produces speech samples on 16 bits (linear) if it is provided with a list of phonemes as input together with prosodic information. MBROLA uses the PSOLA (Pitch Synchronous Overlap Add) method that was originally developed at France Telecom (CNET) [13].

2.5.2 ACAPELA – GROUP

Acapela group constitutes all speech technologies that have been developed over the last 20 years. Speech synthesis and speech recognition have been created and improved by Acapela. Acapela Group evolves from the strategic combination of three major European companies in vocal technologies: "Babel Technologies" created in Mons, Belgium, "Infovox" created in Stockholm, Sweden and "Elan Speech" created in Toulouse, France. Acapela owns currently three technologies, TTS by diphone, TTS by Unit Selection and Automatic Speech Recognition. Acapela is currently available for US English, UK English, Arabic, Belgian Dutch, Dutch, French, German, Italian, Polish, Spanish and Swedish[14].

2.5.3 ARABTALK

The ARABTALK TTS Synthesis System was developed at Research and Development International (RDI), for Arabic language. ARABTALK is a state-of-the-art corpus based concatenative TTS System. The system employs Artificial Neural Networks (ANN) statistical prosody based models for duration, energy, and global pitch contour prediction. In addition, it has a real time synthesis by selection algorithm to explore large speech corpus. ARABTALK has
HMMs based procedure to automatically time-align new voices transcriptions to their acoustic phoneme boundaries. The system is multi-user and safe-threaded enabled for server-based applications [15].

2.5.4 SAKHR TTS

Sakhr TTS engine converts any Arabic/English text into a human voice. Sakhr has been focusing in the last 5 years on creating an Arabic TTS engine that can match in its quality the human voice. This technology gives businesses a competitive edge by allowing them to provide their customers with the latest static and dynamic information anytime, anywhere using normal telephones and mobiles. Sakhr developed the Diacritizer engine (examples of diacritics are “ًٌٍَُِّْْْْْْْْْْ”). This engine can put the diacritics needed in Arabic texts automatically. The Diacritizer is the main component in Arabic TTS. Without the Diacritizer, the output quality of the TTS engine would be inaccurate and not clear. Since Arabic native speakers write Arabic text without diacritics, the TTS engine should handle the non-diacritized text. The Diacritizer will convert the non-diacritized text into a diacritized text and then the TTS engine will convert it to a clear and human Arabic voice [16].
CHAPTER 3
DESIGN AND IMPLEMENTATION

3.1 Concatenative Synthesis

3.1.1 OVERVIEW

One of the most popular methods of synthesizing speech from text is by stringing together or concatenating, prerecorded words, syllables, or other speech segments [17]. This avoids many of problems encountered in phoneme-to-phoneme synthesis, such as the coarticulatory effect between neighboring speech sounds [18].

Still, even words do not usually occur in isolation: the words immediately preceding or following a given word influence its articulation, its pitch, its duration and stress often depending on the meaning of the utterance. This section describes in details Concatenative Synthesis Method and its types.

Synthetic voices are made by concatenating units of sound that have been previously stored in a reference database. The contents of these units and methods of concatenation vary, but the principle of concatenation is universal for TTS involving all but the briefest messages. Nowadays, the use of actual speech waveforms has become increasingly popular, where stored waveforms of various sizes are fetched as needed, with adjustments made mostly at unit boundaries, but sometimes more generally throughout the utterance [19].

Concatenative synthesis method uses a large database of source sounds, segmented into units, and a unit selection algorithm that finds the sequence of units that match best the sound or phrase to be synthesized, called the target. The selection is performed according to the descriptors of the units, which are characteristics extracted from the source sounds, or higher-level descriptors attributed to them. The selected units can then be transformed to fully match the target specification, and are concatenated. However, if the database is sufficiently large, the probability is high that a matching unit will be found, so the need to apply transformations is reduced. The units can be non-uniform, i.e. they can comprise a sound snippet, an instrument note, up to a whole phrase. Concatenative Synthesis can be more or less data driven, where, instead of supplying rules constructed by careful thinking as in a rule based approach; the rules are induced from the data itself. The advantage of this approach is that the information contained in the many sound examples in the database can be exploited.
3.1.2 Details of Concatenative Synthesis

Concatenative synthesis (the so called cut and paste synthesis) uses actual short segments of recorded speech that were cut from recordings and stored in an inventory (“voice database”), either as “waveforms” (uncoded), or encoded by a suitable speech coding method.

A block diagram of a typical concatenative TTS system is shown in Figure 3.1. The front-end on the left converts a given input text string into a string of phonetic symbols and prosody (fundamental frequency, duration, and amplitude) targets. The front-end employs a set of rules and/or a pronunciation dictionary. Together with a string of phonetic symbols, it produces target values for fundamental frequency (pitch), phoneme durations, and amplitudes. The center block in Figure 3.1 assembles the units according to the list of targets set by the front-end. These units are selected from a store that holds the inventory of available sound units.

Figure 3.1: Block diagram of a concatenative text-to-speech system

Different types of speech units may be stored in the inventory of a concatenative TTS system. Storing whole word units is impractical for general TTS because of the tremendous demands on a voice talent that would have to read a few hundreds of thousands of words in a consistent voice and manner. Even if recorded successfully in multiple sessions spread over several weeks, a lack of coarticulation and phonetic recoding at word boundaries may result in unnatural sounding speech. On the other extreme, using phones (e.g., about 50 for English about 36 for Arabic) is also unsatisfactory because of the large coarticulatory effects that exist between adjacent phones. Therefore, transitions from one unit to the next may be audible as “glitches” that introduce perceptually disruptive discontinuities. Intuitively, longer units are more likely to result in higher quality synthesis, given that the rate of concatenations (how many unit-to-unit transitions occur per second of speech) is lower than in the case of shorter units. On the other hand, we need a larger set of longer units to “cover” any application domain, for example,
travel (with TTS-generated prompts such as “From which airport do you want to leave?” or “من أي المطارات سوف تغادر؟”), because of the tremendous multiplicity of possible unit variants [20]. Given these contradictory requirements, most practical TTS implementations until the mid-1990s compromised by using one of two types of inventory units, the diphone and the demi-syllable.

A diphone is the snippet of speech from the middle of one phone to the middle of the next phone. (Note that the average length of a diphone is identical to that of a phone!) The middle of a phone tends to be its acoustically most stable region. Therefore, diphones represent acoustic transitions from the stable midsection of one phone to the next. A minimum inventory of about 1,000 diphones is required to synthesize unrestricted English text. Because concatenative synthesis preserves the acoustic detail of natural speech, diphone synthesis is generally highly intelligible. A disadvantage of strict diphone synthesis is that coarticulation is only provided with the immediately preceding and following phonemes, whereas some phonemes can affect the articulation over several phonemes.

Demi-syllables are alternative units for concatenative synthesis. A demi-syllable encompasses half a syllable; that is, either the syllable-initial portion up to the first half of the syllable nucleus, or the syllable-final portion starting from the second half of the syllable nucleus. The number of demi-syllables in English in Arabic is roughly the same as the number of diphones. Because demi-syllable units are usually longer than diphones, and allow for better capture of coarticulation effects compared to diphones, they should pose fewer concatenation problems.

Note that a typical database (a database that covers all possible diphone units in a minimum amount of sentences) might contain as little as 30 minutes of speech of a single speaker (voice), given that the units must be modified by signal processing to match the front-end predictions and to smooth over the concatenation points. In the following, we will look into some of the signal representations used in TTS. Note, however, that the latest high-quality TTS systems all employ voice databases containing many hours of recorded speech, because not having to modify a segment of speech (because you have the right one in the inventory) will always produce higher quality speech synthesis.

3.1.3 Speech Signal Representation for Concatenative Synthesis

A good speech signal representation for concatenative synthesis approximates the following set of requirements:
1) The speech signal can be stored in a highly compressed (i.e., coded) form so that a large voice database can be used even under tight memory limitations. Coder and decoder are of low computational complexity.

2) Coding/decoding is perceptually transparent. Since we would like to mimic all the voice characteristics of a real person, subjecting the speech signal to “vocoder”-like degradations will not lead to speech synthesis of high naturalness.

3) Coding algorithms have to allow for “random access.” Since most speech coders contain some sort of autoregressive memory, all state variables of the coder have to be made available at concatenation points since the decoder will have to switch between units of speech that are very unlikely to have been recorded consecutively in time.

4) An ideal speech representation must allow for natural-sounding modifications of pitch, duration, and amplitude. This is particularly important for small inventories with one, or just a few, “typical” examples for each unit. Unfortunately, experiments show that, for most signal processing algorithms, modifying pitch more than a few percent may destroy perceived naturalness; that is, a pitch-modified speech signal is likely to be perceptually much different from a speech signal that has been recorded without modifications from the speaker producing the desired pitch value directly (This is the reason why “singing TTS” does not sound like an opera star.)

5) For some advanced applications, it even might be desirable to allow for fine-tuning of the voice, for example, to add more aspiration, mellowness, or let the voice “scream” when needed. Instead of recording different voice inventories for different speaking “styles,” advanced “voice conversion” might be used to approximate an “angry” voice using a “happy” (or “neutral”) voice as a starting point. Today, algorithms for voice conversion (usually concerned with converting the speech of one speaker to sound like speech from another speaker) still do not produce consistently good enough results for sounding like the “real thing,” but might be sufficient for applications such as computer games where even the original voice does not sound “human.”

To optimally achieve this set of requirements, there are three classes of speech signal processing algorithms: low-complexity time-domain algorithms such as Time Domain Pitch-Synchronous Overlap Add (TD-PSOLA) algorithm and its variants, Linear Predictive Coding (LPC-based) algorithms, and frequency domain-based speech representations.

Recording large voice databases can bring with it daunting organizational tasks that should not be underestimated. Selecting the right voice talent, choosing the optimal material to record, providing a consistent recording environment (low background noise, negligible acoustic
CHAPTER 3

reflections from walls, tables, manuscript, constant microphone position relative to mouth, etc.),
and ensuring a correct and consistent speaking style are all part of the effort.

3.1.4 Unit Selection Synthesis

With the availability of good automatic speech labeling tools, concatenative speech
synthesis has now embraced the use of multi-hour voice databases. With the availability of
several (potentially hundreds of thousand) instances of a specific type unit (only different in
pitch, duration, linguistic context at recording time), so-called Unit-Selection Synthesis has
become viable for obtaining high-quality TTS. Based on early research done at ATR in Japan,
this method enables the use of large speech databases recorded using specific, carefully crafted
and controlled, speaking styles (e.g., angry, happy, apologetic, etc.). In addition, a given database
may be focused on narrow-domain applications (such as “travel reservations” or “telephone
number synthesis”) that commonly allow for the use of smaller databases for a preset level of
quality, or it may be used for general applications like e-mail or news reading (requiring a larger
voice database). In the latter case, unit-selection synthesis can require on the order of ten hours
of recording of spoken general material to achieve a desired level of quality, and several dozen
hours for “natural quality” (that can be mistaken for direct recordings). In contrast with earlier
concatenative syntheses, unit-selection synthesis automatically picks the optimal synthesis
units (on the fly) from an inventory that may contain thousands of tokens of a specific unit, and
concatenates the selected units to produce the synthetic speech. This is in stark contrast with the
fact that as late as the mid-1990s voice inventories for concatenative TTS were always carefully
crafted by hand, trying to find the one or few units of each type that lead to optimum results in
all possible synthesis scenarios (contexts). In more than one sense, the unit-selection approach
has succeeded in putting “the expert into the box,” that is, has automated the process of finding
the optimal sequence of inventory units given a “search query” of tagged phoneme strings. One
important difference between “old” and “new” TTS is that unit selection synthesis “knows”
the text to be synthesized at selection time, while previous methods tried to satisfy more global
selection criteria without explicit knowledge of the text to be synthesized.

For any database query, the optimal choice of units selected from the database depends
on factors such as spectral similarity at unit boundaries (components of the “join cost” between
two units) and on matching prosodic targets set by the front-end (components of the “target
cost” of each unit).
3.1.5 Types of Concatenative Synthesis

Natural sounding speech can be produced by concatenating (or stringing together) segments from a database of recorded speech. In general, this approach yields the most natural sounding synthesized output. However, variations in speech and the automated techniques used for segmenting analysis speech waveforms sometimes result in audible glitches in the output. There are many types to concatenative synthesis depending on the fundamental speech unit stored to be used for concatenation (Allophone, Diphone, Syllables and Demi-syllable, Waveform, Words, etc.). Table 2 shows the summary of the advantages and the pitfalls of units of Concatenative Synthesis in terms of a sentence, a word, a syllable, an allophone, a diphone, and a demi-syllable.

<table>
<thead>
<tr>
<th>No.</th>
<th>Concatenation Unit</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sentence</td>
<td>Naturalness throughout</td>
<td>Usually impractical to prerecord every sentence that might be needed.</td>
</tr>
<tr>
<td>2</td>
<td>Word</td>
<td>Naturalness within the word</td>
<td>Inter-word articulation may sound mechanical; usually impractical to prerecord all the words that might be needed.</td>
</tr>
<tr>
<td>3</td>
<td>Syllable</td>
<td>Naturalness within the syllable</td>
<td>Good inter-syllable articulation is difficult to achieve, making individual words unintelligible; usually impractical to prerecord all the syllables that might be needed, though not as seriously as with words or sentences.</td>
</tr>
<tr>
<td>4</td>
<td>Allophone</td>
<td>Naturalness within the allophone, especially vowels; much fewer units need be prerecorded than with other choices</td>
<td>Good articulation between most allophones is difficult to achieve, making individual words unintelligible; stop consonantal allophones are difficult to isolate for prerecording.</td>
</tr>
</tbody>
</table>
3.1.6 Concatenative Synthesis: Pros and Cons

Table 3 summarizes fundamental advantages and disadvantages of concatenative synthesis method:

<table>
<thead>
<tr>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Units include difficult sounds and transition</td>
<td>Long-distance co-articulation not captured</td>
</tr>
<tr>
<td>Units capture local co-articulation</td>
<td>DSP: at least smoothing of concatenative points</td>
</tr>
</tbody>
</table>

Table 3: Concatenative Synthesis: Pros and Cons

Neither pros nor cons:
± DSP: Prosodic modifications possible/ required.
± Compromise between coverage and inventory size.

3.1.7 Requirements for a Concatenative Synthesis System

Any Concatenative Sound Synthesis system must perform the following tasks, which may sometimes perform implicitly. These tasks or steps are:
• **Analysis**: The source sound files are segmented into units and analyzed to express their characteristics with sound descriptors.

• **Database**: Source file references, units and unit descriptors are stored in a database. The subset of the database that is pre-selected for one particular synthesis is called the corpus.

• **Target**: The target specification is generated from a symbolic score (expressed in notes or descriptors), or analyzed from an audio score (using the same segmentation and analysis methods as for the source sounds).

• **Selection**: Units are selected from the database that match best the given target descriptors according to a distance function and a concatenation quality function. The selection can be local (the best match for each target unit is found individually), or global (the sequence with the least total distance if found).

• **Synthesis**: is done by concatenation of selected units, possibly applying transformations.

### 3.2 Arabic TTS System Model

In this section, the complete design and implementation of the Arabic TTS system is introduced. The subsequent sections define the project in terms of input and output (globally for the whole system and locally for each sub-module), the steps which are followed in this project to produce the speech, tools and software that are used in the project, the most important challenges faced in the project, and how were worked around it (how the challenges are solved), some pseudo-codes and flow charts. The code fragments are also available in [APPENDIX D].

![Figure 3.2: Full Model of Arabic TTS System](image)

The input to the program is a plain Arabic text; this text is processed through different module operations, producing a read out text.
3.3 Specifications of Processing Stages

3.3.1 Text and Linguistic Analysis (High-level Synthesis)

3.3.1.1 Structure Analysis

This module is responsible for manipulation of both punctuations ("."/ ","/ " "/ "eterminate"...etc) and the special text categories (times, dates, English words, e-mail or website addresses and numbers).

The input for this module is the whole program input coming either from the user directly through keyboard or from stored text file (.txt). The function of this module is to produce an intermediary text (output) composed only from words (Arabic + English words, e-mails and website addresses) that represent the input for the next stage, namely the Arabic Diacritizer. Also, the input text is processed to determine where paragraphs start and end, sentences and punctuations.

Figure 3.3 depicts the input/output for each sub-module showing each function.
Figure 3.3: Functions of Structure Analysis module
Figure 3.4 shows the flowchart corresponding to Structure Analysis module.

3.3.1.2 Arabic Diacritizer

One of the biggest challenges that faced the Arabic text preprocessing is that the text must be diacritized to be read correctly by the synthesizer, so in the preprocessing step each character and its diacritic must be determined. A half-diacritized lexicon of sample Arabic words (deacritization database) is developed using Microsoft Access 2003 [APPENDIX B]. The fully diacritized Arabic word can be obtained by passing the retrieved half-diacritized word (from diacritization database) back to the user to add the missing diacritic of the last letter in each word, otherwise the default diacritic “َّ” is assumed and added automatically to the word’s end. It is worth mentioning that all spaces (silence) and English words remaining from the previous stage passed this step untouched “English words have no diacritization”. The output of this module includes diacritized Arabic words plus English words from the previous stage in the
sequence entered by the user (a FIFO queue is used to reserve the order). The diacritization process is shown in Figure 3.5.

Figure 3.5: Arabic Diacritization module
Figure 3.6 shows the flowchart corresponding to Arabic Diacritization module.

![Arabic Diacritization Flowchart](image)

3.3.2 Speech (Waveform) Generation (Low-level Synthesis)

3.3.2.1 Arabic Grapheme-to-Phoneme conversion module

The Arabic language has about 445 different phonemes which are classified to vowels and constants as shown in Figure 3.7 and classified in more details in Table 4.
Figure 3.7: Classification charts of Arabic phonemes

<table>
<thead>
<tr>
<th>outlets</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>plosive</td>
</tr>
<tr>
<td></td>
<td>voiced</td>
</tr>
<tr>
<td>labial</td>
<td>ب</td>
</tr>
<tr>
<td>Labiodental</td>
<td></td>
</tr>
<tr>
<td>Interdental</td>
<td>ذ</td>
</tr>
<tr>
<td>Gingival</td>
<td>ض</td>
</tr>
<tr>
<td>Gingiva</td>
<td>س</td>
</tr>
</tbody>
</table>
Table 4: Classifications of Arabic phonemes

The Arabic Grapheme-to-Phoneme conversion module is just as simple as a look-up table that consists of the Arabic word graphemes and their corresponding phonemes. It is important to be realized that each grapheme is formed from both a letter and its diacritics, and the name of the correspondent phoneme is identical to that of its sound file (.wav) in the pre-recorded sounds database (Arabic Speech Synthesing database).

In order to assign the closest phoneme (later, sound) to each grapheme taking into consideration that (in Arabic) pronunciation depends on the diacritic, where there are thirteen different diacritic, each of them gives different pronunciation for the letter. For that reason, sixteen sound files have been recoded for each character and its diacritic. For example: the letter "ف" have the following set of phonemes depending on its diacritic:

It should be noted that For transcribing pronunciations usually the International Phonetic Association’s (IPA) symbol set is used as a standard [APPENDIX A].

| Bucco-gingiva | ش | ج |
| Mid-buccal   | ق | ك | خ |
| Far-buccal   | ك | خ | غ |
| Ovular       | ق | ق |
| Pharyngeal   | ع | ح |
| Laryngeal    | الهنزة | ه |
3.3.2.2 Arabic Phoneme-to-Sound conversion module (Arabic Synthesizer)

After Arabic grapheme-to-phoneme conversion, all Arabic words are represented, each, by its phonemes corresponding to sequence of wave files that must be read sequentially to synthesize the target Arabic word, the following procedure is used.

Since the required wave files names are available implicitly (as parts of the phonemic word representation), consecutive file names must be rewritten in an order that is suitable to the sequential wave player. This is achieved by simply entering a back slash (\) between each .wav file name and another, and preceding the first .wav file name by the absolute path in which the Arabic synthesis database is resident. For example, if the Arabic synthesis database is in folder that has an absolute path name as (C:\ArabicTTS\SynthesisDatabase) and we want the synthesizer to read aloud the word “لَهَاىٰ”\(^t\), then the following string should be used for the source file name:

C:\ArabicTTS\SynthesisDatabase\ga,la,mam2.wav

3.3.2.3 English speech generation module (English Synthesizer)

The main target of the project is to synthesize the Arabic text; thus the English synthesizer module is not of much interest. However, since the system also deals with special classes of text that cannot be written in Arabic language such as e-mail, website addresses and abbreviations which are provided in English language; an English TTS engine should be integrated with the Arabic synthesizer. English TTS systems have reached high stages in development compared to the Arabic ones, that’s now these systems are integrated in a lot of daily user applications (PDF and text processors).

Microsoft Speech API

The Speech Application Programming Interface or SAPI is an API developed by Microsoft to allow the use of speech recognition and speech synthesis within Windows applications. To date, a number of versions of the API have been released, which have shipped either as part of a Speech SDK, or as part of the Windows OS itself. Applications that use SAPI include Microsoft Office, Microsoft Agent and Microsoft Speech Server.

In general all versions of the API have been designed such that a software developer can write an application to perform speech recognition and synthesis by using a standard set of interfaces, accessible from a variety of programming languages. In addition, it is possible for a 3rd-party company to produce their own Speech Recognition and Text-To-Speech engines or adapt existing engines to work with SAPI. In principle, as long as these engines conform to the defined interfaces they can be used instead of the Microsoft-supplied engines.
In general the Speech API is a freely-redistributable component which can be shipped with any Windows application that wishes to use speech technology. Many versions (although not all) of the speech recognition and synthesis engines are also freely redistributable [21].

### 3.4 The general process of the program

The program reads the character and it’s diacritic to know the name of the sound file that represent this character. After knowing the name of all sound files that represent input text, the program has the location/path and the name of files wanted to be played in that directory. Using the tool, the program runs those files in sequence order until a read out text is obtained.

In general the program works as follows:

- The user enters the text directly or specify a text file to be used as an input text for the whole Arabic TTS synthesizer:
  
  ```java
  String str = input text;
  ```

- This text string (str) is processed with the low-level synthesizer, starting from the structure analyzer as in Figure 3.2 and the string (str) is produced as a combination of only undiacritized Arabic, English words and silent periods (spaces) properly.

- Each instance of the string (str), either a word or a silent period, is sent in order to the Arabic so as to be half-diacritized (Arabic words) or passed untouched (English words and silent periods):
  
  $$W_{out}(\text{half-diacritized Arabic}) = W_{in}(\text{undeacritized Arabic word}) + \text{diacritization after each letter except the last;}$$

- $$W_{out}(\text{fully-diacritized Arabic}) = W_{out}(\text{half-diacritized Arabic}) + \text{‘ٰ‘}; // \text{Default diacritization}$$

- The source file name used for the synthesizing tool is formed from $$W_{out}(\text{fully-diacritized Arabic})$$ as explained in section (3.3.2.1). The resultant words sounds are saved temporarily in the tool output buffer.

- As the end of the text is reached, the saved speech is brought back from the buffer and read out aloud in the order of the input text i.e. the final output speech (output of the whole system) is read as a one continuous unit.
3.5 Special cases

The following are special cases to be considered and they are dealt with differently than general cases:

1) If character is a "أ".
2) If the diacritic is "َّ".
3) If character is "ة".
4) "ِّّ".
5) The end of the paragraph.

Firstly:

"أ" – In Arabic language "أ" can be written in four ways:

1) "أ" or "إ" – in this case:
   - "أ": its diacritic is either 'َ', 'َّ', or 'َّّ'.
   - "إ": its diacritic is 'َّ' only.
2) "ؤ" – in this case its diacritic either 'َّ', 'َّ', or 'َّّ' and in all cases it comes after letter whose diacritic is "َّ".
3) "ئ" – in this case its diacritic either 'َّ', or 'َّ' and the diacritic of letter before it is 'َّ'.
4) "أ" – in this case it is dealt with as a normal character that accepts all diacritic except 'َّ'.

Secondly:

Characters are not diacritic with "ّّ" only. However, with it comes a 'َّ', a 'َّّ' or a 'َّّّ'. Therefore, to get the phoneme (the voice file) corresponding to the character 'َّ', it is required to also know the following diacritic that comes with it. And also if the letter which its diacritic is "َّّّ" followed by "َّ" and its diacritic is "َّ" then single phoneme (sound file) is retrieved instead of phoneme (sound file) for each one of them. And also if "َّّّ" or "َّّ" are followed by "َّ" or "َّ" respectively, a single phoneme (sound file) is retrieved instead of phoneme (sound file) for each one.

The following is the pseudo code:
In case that word.chars(i) = "letter from Arabic letters" then
  If (word.chars(i+1) = 'َّ') then
    If (word.chars(i+2) = 'َّ') then
If (word.chars(i+3) = 'ٰ') then
    If (word.chars(i+4) = 'ُ') then
        x = name of the phoneme (sound file) which represents this two letters with their diacritics;
        i = i+ 4;
    End if
Else // in case the letter's diacritic is 'ُُُ' but is not followed by "ٰ"
    x = name of the phoneme (sound file) which represents this letter which its diacritic is 'ُُُ';
    x = i + 2
End if
Else if (word.chars(i+2) = 'ِ') then
    If (word.chars(i+3) = 'ي') then
        If (word.chars(i+4) = 'َ') then
            x = name of the phoneme (sound file) which represents this two letter with their diacritic;
            i = i+ 4;
        End if
    Else // in case the letter's diacritic is 'ِّْْ' but is not followed by "ي"
        x = name of the phoneme (sound file) which represents this letter which its diacritic is 'ِّْْ';
        x = i + 2
    End if
Else if (word.chars(i+2) = 'َ') then
    If (word.chars(i+3) = 'و') then
        If (word.chars(i+4) = 'َ') then
            x = name of the phoneme (sound file) which represents this two letter with their diacritic;
            i = i+ 4;
        End if
    Else // in case the letter's diacritic is 'ُُُ' but is not followed by "و"
        x = name of the phoneme (sound file) which represents this letter which its diacritic is 'ُُُ';
        x = i + 2
    End if
Else if (word.chars(i+2) = 'َ') then
    If (word.chars(i+3) = 'س') then
        If (word.chars(i+4) = 'ُ') then
            x = name of the phoneme (sound file) which represents this two letter with their diacritic;
            i = i+ 4;
        End if
    Else // in case the letter's diacritic is 'ُُُ' but is not followed by "س"
        x = name of the phoneme (sound file) which represents this letter which its diacritic is 'ُُُ';
        x = i + 2
    End if
Else // in case the letter's diacritic is 'ُُُ' but is not followed by "س"
    x = name of the phoneme (sound file) which represents this letter which its diacritic is 'ُُُ';
    x = i + 2
End if
End if
wordout =wordout + x

**Thirdly:**

"ة" – if the diacritic of the "ة" is "ْْ" then it is escaped, because for example the word "سَارَة" has a pronunciation looks like "سَأَرَة". thus for more natural pronunciation this letter is escaped. But if the diacritic of it not "ْْ" it’s phoneme is retrieved as usually done with other character.

The following is the pseudo code:

If (word.chars (i) = "ة")
    If (word.chars (i+1) = "ْْ")
        i = i+1     //escape
    Else if (word.chars (i+1) = "one of the diacritic except "ْْ")
        x= name of the phoneme (sound file) that represents this character and its diacritic;
        i = i+1
        wordout =wordout + x
    End if
End if

**Fourthly:**

"أَلْ" – if the character "أ" which its diacritic is "ْْ" followed by the letter "ل" which its diacritic is "ْْ", a single phoneme (sound file) is retrieved for those letters instead of retrieving phonemes (sound files) for each one; to be more natural. Also some people write it as follow "الْ" (i.e. they do not write the diacritic in the first character), thus this case was considered also.

The following is the pseudo code:

If (word.chars (i) = "أ")
    If (word.chars (i+1) = "ل")
        If (word.chars (i+2) = "ْْ")
            X= name of the phoneme (sound file) that represents those letters with their diacritics
            i =i+2
            wordout =wordout + x
        End if
        Else if (word.chars (i+1) = "ْْ")
            If (word.chars (i+2) = "ل")
                If (word.chars (i+3) = "ْْ")
                End if
            End if
        End if
    End if
Else if (word.chars (i+1) = "ْْ")
    If (word.chars (i+2) = "ل")
        If (word.chars (i+3) = "ْْ")
        End if
    End if
X= name of the phoneme (sound file) that represents those letters with their diacritics

\begin{verbatim}
i =i+3
wordout =wordout + x
End if
End if
End if
End if

Fifthly:

The end of the paragraph:

Sometimes the paragraph ends in the middle of the line and the other starts in the second line so the program encounters a lot of space until it reaches the new paragraph. In the normal situation (i.e. when there are at most three sequence of spaces), empty phoneme (sound file) is retrieved for the space, but here if there are more than three sequence of spaces, empty phoneme (sound file) is retrieved for each one of them so having too much silence. Thus if more than three spaces are encountered, the space is escaped until the first character in the new paragraph is found.

The following is the pseudo code:

\begin{verbatim}
If (word.chars (i) = " ") then
    If (word.chars (i+1) = " ") then
        If (word.chars (i+2) = " ") then
            If (word.chars (i+3) = " ") then
                While (word.chars (i) = " ") {
                    i = i+1
                }
            End if
        Else   // if there are three spaces
            x = name of the phoneme (sound file) that represents the three spaces
            i = i+2
        End if
    Else   // if there are two spaces
        x = the name of the phoneme (sound file) that represents the two spaces
        i = i+2
    End if
Else // if there is one space
\end{verbatim}
x = the name of the phoneme (sound file) that represents the space

End if

### 3.6 Software & Tools

The software used in the development of the Arabic TTS Synthesizer project is Visual C++.Net, In addition, TeleTool has been used as a tool to deal with sound files, which plays number of sound files sequentially after specifying the location of those files and the name of the sounds which to be played in that directory (e.g. C:\sound file\ x, y, z.wav where x, y, and z are the .wav sound files to be played in that directory). The sound files have been recorded by using software called CoolEdit [APPENDIX B].

### 3.7 Special implementation problems and their solutions

- In the Arabic language there are some special cases where there is no specific rule for the pronunciation. For example, the case of the ‘لا’ that is pronounced in the word but it is not written like in "طنا".

**Suggested Solution:**

The program is given the ability to deal with most of the familiar/known words such as (طنا، هذآ، هذى، هلل) those have no specific rule for pronunciation, by simply record those words (as a complete word not phonemes) and add them to the Arabic TTS synthesis database.

- Punctuations, full-stops, and how we know the paragraph is ends or new paragraph is start.

**Suggested Solution:**

Empty sound file is recorded to deal with punctuations. The length of this file depends on the punctuation, which can be long or short. For paragraph endings, the program ignores empty spaces until it reaches the beginning of a new paragraph.

- The problem with the character whose diacritic is "َ" is one of the main issues came across in the program implementation, because it is hard to record that character alone due to the fact that, in the Arabic language, a word cannot start with a letter whose diacritics is "ُ".

**Suggested Solution:**

There are two ways to solve the issue of the character whose diacritic is "َ". In first solution, it is required to record all the possible diacritics for the character before that character, while in the second solution, it is required to record the character whose diacritic is "ُ" then use Cool Edit tools and work with its recording until it becomes appropriate with the rest of the files. The
second solution is used in this program since the first solution increased the size of the program enormously.
CHAPTER 4
RESULTS AND DISCUSSION

4.1 Introduction

After the design and implementation phase is completed, the TTS system’s results are ready for assessment part. In some applications, for example reading machines for the blind, the speech intelligibility with high speech rate is usually more important feature than the naturalness. On the other hand, prosodic features and naturalness are essential when we are dealing with multimedia applications or electronic mail readers. The evaluation can also be made at several levels, such as phoneme, word or sentence level, depending on what kind of information is needed. In this chapter, test parameters plus design of the test and results are discussed.

The program final layout is in the form of windows based program that allow the user to go through the steps of language processing and modifying –if needed- in specific steps and then play the synthesized text. Figure 4.1 shows the final form of the program window.

![Figure 4.1: The final form of the program](image-url)
4.2 TESTING AND EVALUATING TTS SYSTEMS

This phase is done by selecting a group of random people and allows them to try the program while going through a questionnaire that will be used to evaluate the project throughput. The questionnaire was designed to assess the intelligibility (clearness), naturalness, sound quality and the pronunciation on the level of phoneme word and sentence. The group was consisting of 20 people of different professions and language knowledge in order to obtain a good assessment (overview of the program's operation).

4.2.1 Intelligibility

Two questions were asked concerning the intelligibility of the system. The first question was "How much you understand the voice?"

The results:

- Understood much 75%
- Understood very much 10%
- Understood neither much nor little 15%
- Understood little 00%

Figure 4.2 shows the percentage (%) vs. the degree.

![Intelligibility Chart]

**Figure 4.2: intelligibility 1st result**

The second question was "was the voice easy to grab/get?" The reason for this question was to establish if the difficulty in grabbing/getting is in the voice or in the listeners’ lack of knowledge and lexicology.
The results:

- Easy 40%
- Very easy 05%
- Neither easy nor hard 45%
- Hard 10%
- Very hard 00%

Figure 4.3 shows the percentage (%) vs. the degree.

![Figure 4.3: intelligibility 2nd result](image)

### 4.2.2 Naturalness

Regarding the question "Was the sound natural or not?" the obtained results were:

- Natural 25%
- Very natural 10%
- Ok 40%
- Unnatural 20%
- Very unnatural 05%

Figure 4.4 shows the percentage (%) vs. the degree.
4.2.3 Sound quality

In this part the question was "what level of quality do you think the system has?", and the results were:

- Good                    50%
- Very good                10%
- Neither good nor bad     25%
- Bad                      15%
- Very bad                 0%

Figure 4.5 shows the percentage (%) vs. the degree.

Figure 4.4: naturalness result

Figure 4.5: sound quality result
4.2.4 Pronunciation

The pronunciation part consists of four questions addressed to the participants to be able to get an idea of how difficult the speech uttered by the system is to grab/get and to be able to decide what sounds are the most difficult ones to catch and gradually process these sounds in some way and improve them.

The first question was "Is it very hard to grab/get some of the words?" And which one?" and the results were:

- Hard 35%
- Very hard 05%
- Neither hard nor easy 40%
- Easy 15%
- Very easy 00%

Figure 4.6 shows the percentage (%) vs. the degree.

Figure 4.6: pronunciation 1st result

The second question was "Did you have to concentrate hard to grab the speech?" this question can give information about how difficult the voice is to grab/get and how much the participants had to concentrate to grab/get the voice.

The results:

- A lot of concentration 15%
- Some concentration 35%
- Normal concentration 35%
- Little concentration 15%
• No concentration 00%

Figure 4.7 shows the percentage (%) vs. the degree.

Figure 4.7: pronunciation 2nd result

The third question was "How much annoying did you find the voice?" and the results were:

• Not annoying 30%
• Little annoying 50%
• Annoying 20%
• Very annoying 00%
• Too much annoying 00%

Figure 4.8 shows the percentage (%) vs. the degree.

Figure 4.8: pronunciation 3rd result
The last question in this test was about the error "Do you think the voice makes many pronunciation mistakes?" and the results were:

- Many: 05%
- Too many: 05%
- Neither many or few: 25%
- Few: 40%
- Very few: 25%

Figure 4.9 shows the percentage (%) vs. the degree.

![Figure 4.9: pronunciation 4th result](image)

From the above results and analysis, it implies that, when it comes to the intelligibility of the system, the Arabic TTS Synthesizer System is successful. The participants can hear what is being said and recognize changes with the synthesized speech. The majority of both words and sentences were correctly recognized and perceived by the majority of the listeners and the evaluation of the overall quality of the system is satisfying at this stage.

Since the concatenated speech is produced from a prerecorded phoneme units; the discontinuity problem arise in its clearest form because of the large co-articulatory effects that exists between adjacent phones (section 3.1.2) and longer units are more likely to result in higher quality synthesis, given that the rate of concatenation is lower than in the case of shorter units, also the quality with some consonants may vary considerably and the controlling of pitch and duration may be in some cases difficult, especially with longer units.

Putting into consideration that:

- The last sound has been recorded after two months from the start of the recording phase,
- the voice system state for the voice producer (Dr. El-Sadig) and
• The recording environment during these two months.

The degraded naturalness of the concatenated word that has appeared in the results is reasonable.

The diacritization function of the system gives only one morphological form for each word, thus an error in the pronunciation on the level of word and sentence will defiantly occurs—most of the Arabic words have more than one morphological form—resulting in the reduction of the pronunciation quality appears in the results. However, the quality of the pronunciation on the level of phonemes was pretty good since the system was based on this level. The results also showed that some phonemes were difficult to get and must be rerecorded such as letters diacritized with "ّ". The concentration results show that the concentration could be as normal as listening to the news.
CHAPTER 5
CONCLUSION AND FUTURE WORK

5.1 Conclusion

Text-To-Speech Synthesizer has been developed gradually over the last few decades and it has been incorporated into several new applications. For most applications, the intelligibility and comprehensibility of TTS Synthesizer have reached the acceptable level. However, in prosodic, text preprocessing, and pronunciation fields there is still much work and improvements to be done to achieve more natural sounding speech. Natural speech has so many dynamic changes that perfect naturalness may be impossible to achieve. However, since the markets of speech synthesis related applications are increasing steadily, the interest for giving more efforts and funds into this research area is also increasing. Present speech synthesis systems are so complicated that one researcher cannot handle the entire system. With good modularity it is possible to divide the system into several individual modules whose developing process can be done separately if the communication between the modules is made carefully.

As mentioned in the chapter 2, there are several methods for TTS systems, the most popular ones are based on the formant and the concatenative synthesis. The concatenation method is preferred over the parameter methods (formant and articulatory) since reducing the problems of the discontinuity in concatenation points are becoming more effective.

Some other techniques have been applied to speech synthesis, such as Artificial Neural Networks and Hidden Markov Models. These methods have been found promising for controlling the synthesizer parameters, such as gain, duration, and fundamental frequency.

This TTS Synthesizer System is based on concatenation method. The challenges with the Arabic language when building TTS Systems are so many. Examples of these problems and challenges are the diacritization problem, the existing of many dialects in the Arabic language, and the differences in gender. This mapping of the problems would be helpful for others who wish to build a TTS Synthesizer System in Arabic and other languages who have not been extensively studied and processed.

The high-level synthesis is perhaps the least developed part of present synthesizers and needs special attention in the future. Especially controlling prosodic features has been found very difficult and the synthesized speech still sounds usually synthetic or monotonic. The methods for correct pronunciation have been developed steadily during last decades and the present systems are quite good, but improvements with especially proper names are needed. Text preprocessing
with numbers and some context-dependent abbreviations is still very problematic. However, the development of semantic parsing or text understanding techniques may provide a major improvement in high-level speech synthesis.

Lastly, this dissertation has fulfilled its purpose, through creating a fully working Arabic TTS Synthesizer System. It can be said that the system provides satisfactory results after the testing but extensive and continued work is required to develop the system further and to get a high quality TTS Synthesizer System. The results of this system are very promising, with high level of intelligibility. Although the questionnaire with the test and the evaluation of the system is quite simple, guidelines and information of the intelligibility, naturalness, speed and overall quality of the system can be identified. In observing a large and diverse group of participants would have enabled a better evaluation of the system. The small number of participants has affected the test results and the evaluation of this system negatively. The small number of subjects does not allow comparing the subjects and the results taking the level of fluency into consideration, which would have been interesting to see if it would be any differences. Therefore a better division of the group and a larger amount of people is recommended.

5.2 Future work

As mentioned, TTS systems are complicated (section 5.1) - the higher quality and naturalness the higher complexity – and need a lot of work and time to accomplish the ultimate goal (section 1.1).

Although the system has been created, yet it must be noted that there is still work to be done in order to improve the existing features and add new ones. Listed below, some of the possible improvements that can be made.

- Rerecord some of the sounds to overcome the problems faced in the test. Some letters are wrong pronounced and have to be replaced with the right one.
- Record more sounds in the sound library. More sounds can be recorded to have better performance and more vocabularies. Users can learn more words without much limitation.
- Modifying the diacritics database so that it includes all the morphological forms of the word, resulting in minimizing the error in pronunciation on the level of word and sentence.
- Adding function to handle the abbreviations and acronyms either by reading litter-by-litter or replace it with the full words. A database could be created to provide the most popular and important abbreviations, acronyms and their corresponding words.
- Advanced setting in the system could be added, allowing the professional users to modify in the characteristics of the sound such as pitch and duration, to minimize the discontinuity problem and generally to improve the quality.

- The facility to stop, pause and continue reading can be provided, i.e., the user should be able to pause the synthesizer at any time and then continue reading using just a mouse-click.

- Multiple voices (both male and female versions) can be provided to users to choose depending upon their interest.
REFERENCES


### APPENDIX A

**IPA SYMBOLS**

- The following are IPA (and optional) symbols with the corresponding Arabic letter.

<table>
<thead>
<tr>
<th>Arabic</th>
<th>Name</th>
<th>Symbol</th>
<th>Arabic</th>
<th>Name</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>ﮤ</td>
<td>hamzah’[ʔ]’</td>
<td>ء</td>
<td>ﮤ</td>
<td>sād’[s]’</td>
<td>َّ/ّ/ّ/ّ *</td>
</tr>
<tr>
<td>ﮣ</td>
<td>bā’[b]’</td>
<td>ﺏ</td>
<td>ﮣ</td>
<td>dād’[d]’</td>
<td>ﺚ/ّ/ّ/ّ * **</td>
</tr>
<tr>
<td>ﮥ</td>
<td>lā’[t]’</td>
<td>ﺔ</td>
<td>ﮥ</td>
<td>fār’[f]’</td>
<td>ﺮ/ّ/ّ/ّ *</td>
</tr>
<tr>
<td>ﺬ</td>
<td>yā’[θ]’</td>
<td>ﺖ</td>
<td>ﺬ</td>
<td>dār’[d]’</td>
<td>ﺚ/ّ/ّ/ّ/ّ ***</td>
</tr>
<tr>
<td>ﺣ</td>
<td>gīm [g]’</td>
<td>ﺣ</td>
<td>ﺣ</td>
<td>yāyn ’[i]’</td>
<td>ﺘ/ّ</td>
</tr>
<tr>
<td>ﺣ</td>
<td>hā’[h]’</td>
<td>ﺜ/ّ</td>
<td>ﺣ</td>
<td>yāyn ’[y]’</td>
<td>ﺘ</td>
</tr>
<tr>
<td>ﺣ</td>
<td>xā’[x]’</td>
<td>ﺔ</td>
<td>ﺣ</td>
<td>fār’[f]’</td>
<td>ﺘ</td>
</tr>
<tr>
<td>ﺥ</td>
<td>dāl’[d]’</td>
<td>ﺥ</td>
<td>ﺥ</td>
<td>qāf’[q]’</td>
<td>ﺙ/ّ/ّ **</td>
</tr>
<tr>
<td>ﺥ</td>
<td>dāl’[d]’</td>
<td>ﺥ</td>
<td>ﺥ</td>
<td>kāf’[k]’</td>
<td>ﺙ</td>
</tr>
<tr>
<td>ﺫ</td>
<td>rā’[r]’</td>
<td>ﺪ</td>
<td>ﺫ</td>
<td>lām’[l]’</td>
<td>ﺪ</td>
</tr>
<tr>
<td>ﺧ</td>
<td>zāy’[z]’</td>
<td>ﺧ</td>
<td>ﺧ</td>
<td>mīm’[m]’</td>
<td>ﺧ</td>
</tr>
<tr>
<td>ﺩ</td>
<td>sīn’[s]’</td>
<td>ﺩ</td>
<td>ﺩ</td>
<td>mīm’[n]’</td>
<td>ﺩ</td>
</tr>
<tr>
<td>ﺩ</td>
<td>sīn’[s]’</td>
<td>ﺩ</td>
<td>ﺩ</td>
<td>hāl’[h]’</td>
<td>ﺩ</td>
</tr>
</tbody>
</table>

* In current phonetic practice, the Arabic pharyngeal [ء، ِ] are considered epiglottal, so the symbols for these two consonants have shifted from [١، ِ] to [ّ] respectively.

** These symbols represent the voiceless and the voiced uvular stop.

*** This symbol may be presented as one symbol or a symbol with a superscript to indicate velarization of pharyngealization.
- Additional Optional Symbols (to be used consistently):

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>th</td>
<td>gh</td>
<td></td>
</tr>
<tr>
<td>j</td>
<td>s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>h</td>
<td>d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>kh</td>
<td>t</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dh</td>
<td>z</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sh</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Vowels:

<table>
<thead>
<tr>
<th></th>
<th>short front/back low a</th>
<th>Long front/back low ä</th>
</tr>
</thead>
<tbody>
<tr>
<td>short high back rounded u</td>
<td>u</td>
<td>ü</td>
</tr>
<tr>
<td>short high front rounded i</td>
<td>i</td>
<td>i</td>
</tr>
<tr>
<td>semi vowel w</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>doubled uww</td>
<td>iy</td>
<td>iy</td>
</tr>
<tr>
<td>diphthong aw</td>
<td>ãw</td>
<td>ãw</td>
</tr>
<tr>
<td>diphthong ay</td>
<td>ãy</td>
<td>ãy</td>
</tr>
</tbody>
</table>
APPENDIX B

SOFTWARE TOOLS

• **Visual C++ .Net 2008**

  Microsoft Visual C++ (often abbreviated as MSVC or VC++) is a commercial, integrated development environment (IDE) product from Microsoft for the C, C++, and C++/CLI programming languages. It has tools for developing and debugging C++ code, especially code written for the Microsoft Windows API, the DirectX API, and the Microsoft .NET Framework.

  **Visual C++ 2008** (known also as Visual C++ 9.0) was released in November 2007. This version supports .NET 3.5. Managed C++ for CLI is still available via compiler options. By default, all applications compiled against the Visual C++ 2008 Runtimes (static and dynamic linking) will only work under Windows 2000 and later. A feature pack released for VC9, later included into SP1, added support for C++ TR1 library extensions.


• **Microsoft Office Access 2003**

  Microsoft Office Access, previously known as Microsoft Access, is a relational database management system from Microsoft that combines the relational Microsoft Jet Database Engine with a graphical user interface and software-development tools. It is a member of the Microsoft Office suite of applications, included in the Professional and higher editions or sold separately. Access stores data in its own format based on the Access Jet Database Engine. It can also import or link directly to data stored in other applications and databases. Software developers and data architects can use Microsoft Access to develop application software, and "power users" can use it to build simple applications. Like other Office applications, Access is supported by all .Net languages (Visual Basic, C++, C#, and J#) for Applications, an object-oriented programming language that can reference a variety of objects including DAO (Data Access Objects), ActiveX Data Objects, and many other ActiveX components.
• **Cool Edit Pro 2000**

Cool Edit Pro is an audio recording and editing software program, originally marketed by a company called Syntrylliam. Adobe has since purchased the program and all the rights to the program, and now sells it as "Adobe Audition". It can be also used to record vocals over music.

It is a sound manipulation program that allows you to open, create or record new sound files and edit them. You can use Cool Edit to record your own music, voice or other audio, edit it, and mix it with other audio or musical parts.

It may be thought of as a paint program for audio—just as a paint program that enables one to create images with colors, brush strokes, and a variety of special effects, Cool Edit enables one to "paint" with sound: tones, pieces of songs and voices and miscellaneous noises, sine waves and saw tooth waves, noise, or just pure silence. Cool Edit also gives a wide variety of special effects to "touch up" sounds: reverberation, noise reduction, echo and delay, flanging, filtering, and many others.

**Features**

- Record files from a CD, keyboard, or any signal played through your sound card.
- Edit and master audio with professional digital tools.
- Convert files from one format to another
- Touch up files with functions like Filter, Amplify, Compress, Stretch, and Noise.
- Create sound effects of your own.

• **TeleTools**

TeleTools is a collection of ActiveX and VCL controls you call from your telephony application so it works with most Windows software development environments. These controls release you from the drudgery of writing low-level code. Can play, record, fast-forward, rewind, pause, save, change formats, and just about anything you need to do with wave files. The Buffers drop down list is used to select the wave memory buffer to be used when this value is set to zero the no buffer is selected. The Play button plays the wave file contained in the currently selected memory buffer. When this button is clicked the properties etPlay->Source->Buffer->Active is set to true so the etPlay control knows to use the memory buffer instead of a file. The property etPlay->Source->Buffer->Handle is set to the currently selected memory buffer.
Assuming a linear time-invariant system, the model can be described in Z-transform notation by the equation:

\[ S(z) = E(z) G(z) V(z) L(z) \]  \hspace{1cm} (C.1)

Where \( S(z) \) is the speech signal, \( E(z) \) the impulse excitation, \( G(z) \) the glottal shaping model, \( V(z) \) the vocal tract model, and \( L(z) \) the lip radiation model. The impulse excitation \( E(z) \) does not represent a physical signal, but is rather used as a mathematical input to the glottal model filter. Transfer functions \( G(z) \) and \( V(z) \) are usually described with all-pole linear filters, and \( L(z) \) is given by a differencing filter

\[ L(z) = 1 - \rho z^{-1} \]  \hspace{1cm} (C.2)

Where, \( \rho \) is set to 1. Lip radiation \( L(z) \) is the only numerator, but it is nearly canceled by one of the denominator terms. Thus the model can be described as

\[ S(z) = E(z) \frac{1}{A(z)} \]  \hspace{1cm} (C.3)

Where the all-pole filter is defined as

\[ A(z) \approx \frac{1}{G(z)V(z)L(z)} \]  \hspace{1cm} (C.4)
APPENDIX D

USEFUL CODE FRAGMENTS

Arabic TTS Synthesizer Program

Setting the program’s default language to Arabic:

```csharp
private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
{
    System::Globalization::CultureInfo^ myCI = gcnew System::Globalization::CultureInfo("ar-SA");
    InputLanguage::CurrentInputLanguage = InputLanguage::FromCulture(myCI);
}
```

Getting Text from file:

(user can enter text directly via keyboard into textBox1)

```csharp
private: System::Void button4_Click(System::Object^ sender, System::EventArgs^ e)
{
    this->textBox1->Clear();
    Stream^ myStream;
    OpenFileDialog^ openFileDialog1 = gcnew OpenFileDialog;
    openFileDialog1->InitialDirectory = "D:\OurTextFiles";
    openFileDialog1->Filter = "txt files (*.txt)|*.txt|All files (*.*)|*.*";
    openFileDialog1->FilterIndex = 2;
    openFileDialog1->RestoreDirectory = true;
    String^ file_name;
    if ( openFileDialog1->ShowDialog() == System::Windows::Forms::DialogResult::OK )
    {
        if ( (myStream = openFileDialog1->OpenFile()) != nullptr )
        {
            // Create an instance of StreamReader to read from a file.
            System::IO::StreamReader ^ MyStreamReader = gcnew System::IO::StreamReader
```
(openFileDialog1->FileName);
    file_name = openFileDialog1->FileName;

//begin
try
{
    // Create an instance of StreamReader to read from a file.
    StreamReader^ MyStreamReader = gcnew StreamReader( file_name );
    try
    {
        String^ line;
        String^ file_str;
        int count = 0;
        // Read and display lines from the file until the end of
        // the file is reached.
        while ((file_str = MyStreamReader->ReadLine())!= nullptr)
        {
            line = file_str + "\n";
            this->textBox1->AppendText (line);
        }
    }
    finally
    {
        if ( MyStreamReader )
            delete (IDisposable^)MyStreamReader;
    }
    catch ( Exception^ e )
    {
        // Let the user know what went wrong.
        this->textBox1->Text = "The file could not be read:" ;
        this->textBox1->AppendText (e->Message);
    }
//end
}
Processing time/date/English word, e-mail and web addresses/punctuations:

(putting result into textBox2)

```csharp
private: System::Void button1_Click(System::Object^ sender, 
System::EventArgs^ e) {
    this->textBox2->Font = gcnew System::Drawing::Font
        ("Microsoft Sans Serif", 8.25, FontStyle::Regular);
    this->textBox2->ForeColor = System::Drawing::Color::Brown;
    this->textBox2->Visible = true;
    // time processor
    String^ before = this->textBox1->Text;
    String^ number1 = "";
    String^ number2 = "";
    String^ number3 = "";
    String^ my_number = "";
    String^ after = before;
    Regex^ time_regex = gcnew Regex
        ("(?<hours>(\[0-1]\[0-9]|\[2]\[0-3]))+: (?<minutes>(\[0-5]\[0-9]+))\[ \]?(?<WHEN>(\[\|\])|)\)",
        static_cast<RegexOptions>(RegexOptions::Compiled | RegexOptions::IgnoreCase) );
    MatchCollection^ matches = time_regex->Matches(before);
    Match^ m = time_regex->Match(before);
    while (m->Success)
    {
        number1 = m->Groups["hours"]->Value;
        number2 = m->Groups["minutes"]->Value;
        if (m->Groups["WHEN"]->Value != "")
            number3 = m->Groups["WHEN"]->Value;
```
number3 = WHEN(number3);

double my_temp1 = Convert::ToDouble (number1);
double my_temp2 = Convert::ToDouble (number2);
number1 = Get_number(my_temp1);
number2 = Get_number(my_temp2);
my_number = " الساعة " + number1 + " و " +
number2 + " دقيقة " + number3;

after = time_regex->Replace(before, my_number);
before = after;
m = m->NextMatch(); // advance to the next match.

} // end of time processor

// date processor

Regex^ date_regex = gcnew Regex
("(?<month>([1-9]|0[1-9]|1[012]))[- /.]"
(?<day>([1-9]|0[1-9]|1[2][0-9]|3[01])
[- /.](?<year>[0-9]{4}))");
MatchCollection^ matches2 = date_regex->Matches(before);
Match^ m2 = date_regex->Match(before);
while (m2->Success)
{
    number3 = m2->Groups["month"]->Value;
    number2 = m2->Groups["day"]->Value;
    number1 = m2->Groups["year"]->Value;
    double my_temp1 = Convert::ToDouble (number1);
    double my_temp2 = Convert::ToDouble (number2);
    number3 = m2->Groups["month"]->Value;
number3 = Get_month(number3);
number1 = Get_number(my_temp1);
number2 = Get_number(my_temp2);
my_number = " يوم " + number2 + 
" من عام " + number3 + " من شهر " + number1;
after = date_regex->Replace(before, my_number);
before = after;
m2 = m2->NextMatch(); // advance to the next match.
} // end of date processor

// e-mail processor
Regex^ email_regex = gcnew Regex
("(?<user>[a-zA-Z0-9._%+-])@(?<host>[a-zA-Z0-9-]+).(?<domain>[a-zA-Z]+)[ ]");
/* you must enter a space after the email even if it 
was the last thing in the line.*/
MatchCollection^ matches3 = date_regex->Matches(before);
Match^ m3 = email_regex->Match(before);
while (m3->Success)
{
    number1 = m3->Groups["user"]->Value;
    number2 = m3->Groups["host"]->Value;
    number3 = m3->Groups["domain"]->Value;
    my_number = number1 + " at " + number2 + 
" dot " + number3 + " ";
    after = email_regex->Replace(before, my_number);
    before = after;
m3 = m3->NextMatch(); // advance to the next match.
// punctuation processor
Regex^ punct_regex = gcnew Regex("(?<punct>[:][]{1,}|[\[\]
:][\[\]])");
MatchCollection^ matches4 = punc_regex->Matches(before);
Match^ m4 = punc_regex->Match(before);
while (m4->Success)
{
    my_number = m4->Groups["punc"]->Value;
    after = punc_regex->Replace(before, my_punc);
    before = after;
    m4 = m4->NextMatch(); // advance to the next match.
} // end of punc processor

this->textBox2->Text = after;

Functions:

WHEN:

// مساء و صباحا
String^ WHEN (String^ am_pm)
{
    if (am_pm == "ص") return "صباحا";
    else return "مساء";
} // مساء و صباحا نهاية

Get_month:

// دالة
String^ Get_month (String^ shahr)
{ 
if ( (shahr == "1")||(shahr == "01"))  return "يناير";
if ( (shahr == "2")||(shahr == "02"))  return "فبراير";
if ( (shahr == "3")||(shahr == "03"))  return "مارس";
if ( (shahr == "4")||(shahr == "04"))  return "أبريل";
if ( (shahr == "5")||(shahr == "05"))  return "مايو";
if ( (shahr == "6")||(shahr == "06"))  return "يونيو";
if ( (shahr == "7")||(shahr == "07"))  return "يوليو";
if ( (shahr == "8")||(shahr == "08"))  return "أغسطس";
if ( (shahr == "9")||(shahr == "09"))  return "سبتمبر";
if (shahr == "10")  return "أكتوبر";
if (shahr == "11")  return "نوفمبر";
else return "ديسمبر";
} // شهر دالة نهاية

Get_number:

// Number conversion function
String^ Get_number(double X)
{
String^ Letter1 = nullptr;
String^ Letter2 = nullptr;
String^ Letter3 = nullptr;
String^ Letter4 = nullptr;
String^ Letter5 = nullptr;
String^ Letter6 = nullptr;
String^ c = Math::Floor(X).ToString("000000000000");
double C1 = Convert::ToDouble(c->Substring(11,1));
if (C1 == 1) {Letter1 = "واحد";}
}
else if (C1 == 2) {Letter1 = "اثنين";} 
else if (C1 == 3) {Letter1 = "ثلاثة";} 
else if (C1 == 4) {Letter1 = "أربعة";} 
else if (C1 == 5) {Letter1 = "خمسة";} 
else if (C1 == 6) {Letter1 = "ستة";} 
else if (C1 == 7) {Letter1 = "سبعة";} 
else if (C1 == 8) {Letter1 = "ثمانيه";} 
else if (C1 == 9) {Letter1 = "تسعة";} 

double C2 = Convert::ToDouble(c->Substring(10, 1)); 
if (C2 == 1) {Letter2 = "عشر";} 
else if (C2 == 2) {Letter2 = "عشرين";} 
else if (C2 == 3) {Letter2 = "ثلاثين";} 
else if (C2 == 4) {Letter2 = "أربعين";} 
else if (C2 == 5) {Letter2 = "خمسين";} 
else if (C2 == 6) {Letter2 = "ستين";} 
else if (C2 == 7) {Letter2 = "سبعين";} 
else if (C2 == 8) {Letter2 = "ثمانين";} 
else if (C2 == 9) {Letter2 = "تسعين";} 
if (Letter1 != nullptr && C2 > 1) 
{Letter2 = Letter1 + "و" + Letter2;} 
if (Letter2 == nullptr || Letter2 == nullptr) 
{Letter2 = Letter1;} 
if (C1 == 0 && C2 == 1) {Letter2 = Letter2 + "ة";} 
if (C1 == 1 && C2 == 1) {Letter2 = " seri عشر";} 
if (C1 == 2 && C2 == 1) {Letter2 = "اثنى عشر";} 
if (C1 > 2 && C2 == 1) {Letter2 = Letter1 + " " + Letter2;} 
double C3 = Convert::ToDouble(c->Substring(9, 1)); 
if (C3 == 1) {Letter3 = "منة";}

D-8
else if (C3 == 2) {Letter3 = "مئتين";}
else if (C3 > 2) {
Letter3 = Get_number(C3)->Substring(0, Get_number(C3)->Length - 1) + "منة";}
if (Letter3 != nullptr & Letter2 != nullptr) {Letter3 = Letter3 + "و" + Letter2;}
if (Letter3 == nullptr) {Letter3 = Letter2;}
double C4 = Convert::ToDouble(c->Substring(6, 3));
if (C4 == 1) {Letter4 = "ألف";}
else if (C4 == 2) {Letter4 = "ألفين";}
else if (C4 >= 3 && C4 <= 10) {Letter4 = SFormatNumber(C4) + "آلاف";}
else if (C4 > 10) {Letter4 = Get_number(C4) + "ألف";}
if (Letter4 != nullptr & Letter3 != nullptr) {Letter4 = Letter4 + "و" + Letter3;}
if (Letter4 == nullptr) {Letter4 = Letter3;}
double C5 = Convert::ToDouble(c->Substring(3, 3));
if (C5 == 1) {Letter5 = "مليون";}
else if (C5 == 2) {Letter5 = "مليونين";}
else if (C5 >= 3 && C5 <= 10) {Letter5 = SFormatNumber(C5) + "ملايين";}
else if (C5 > 10) {Letter5 = Get_number(C5) + "مليون";}
if (Letter5 != nullptr & Letter4 != nullptr) {Letter5 = Letter5 + "و" + Letter4;}
if (Letter5 == nullptr) {Letter5 = Letter4;}
double C6 = Convert::ToDouble(c->Substring(0, 3));
if (C6 == 1) {Letter6 = "مليار";}
else if (C6 == 2) {Letter6 = "مليارين";}
else if (C6 > 2) {Letter6 = Get_number(C6) + "مليار";}
if (Letter6 != nullptr & Letter5 != nullptr) {Letter6 = Letter6 + "و" + Letter5;}
if (Letter6 == nullptr) {Letter6 = Letter5;}

APPENDIX D

USEFUL CODE FRAGMENTS
(Letter6 = Letter6 + "," + Letter5;)
if (Letter6 == nullptr) {Letter6 = Letter5;)
return Letter6;
} // end of Number conversion function

Phonetization (Transliteration) Function:

// Transliterator Function: a target must be a phoneme name

string ret_str (wchar_t text)
{
    if (text==L'ا') return "a";
    else if (text==L'أ') return "a";
    else if (text==L'آ') return "a";
    else if (text==L'إ') return "e";
    else if (text==L'ة') return "b";
    else if (text==L'ث') return "t";
    else if (text==L'ث') return "th";
    else if (text==L'ج') return "j";
    else if (text==L'ح') return "h";
    else if (text==L'خ') return "kh";
    else if (text==L'د') return "d";
    else if (text==L'ذ') return "d";
    else if (text==L'ر') return "r";
    else if (text==L'ز') return "z";
    else if (text==L'س') return "s";
    else if (text==L'ش') return "sh";
    else if (text==L'ص') return "s";
    else if (text==L'ض') return "d";
}
else if (text==L'ء') return "t";
else if (text==L'ت') return "z";
else if (text==L'ؤ') return "e";
else if (text==L'ح') return "gh";
else if (text==L'ئ') return "f";
else if (text==L'ب') return "q";
else if (text==L'آ') return "k";
else if (text==L'ؤ') return "l";
else if (text==L'ت') return "m";
else if (text==L'ا') return "n";
else if (text==L'ت') return "h";
else if (text==L'ا') return "w";
else if (text==L'ث') return "y";
else if (text==L'ث') return "a";
else if (text==L'ث') return "e";
else if (text==L'ث') return "'e";
else if (text==L'ث') return "'e";
else if (text==L'ث') return "la";
else if (text==L'ا') return "h";
else if (text==L'ث') return "?";
else if (text==L'ا') return "!";
else if (text==L'ة') return "";
else if (text==L'ة') return ".";
else if (text==L'ث') return "a";
else if (text==L'ا') return "u";
else if (text==L'ث') return "e";
else if (text==L'ث') return "un";
else if (text==L'ا') return "an";
else if (text==L'콤') return "";
else if (text==L' ') return " ";
else if (text==L'\n') return "\n";
else
{
    char other[2] ;
wctomb( other, text );
    return other;
}
} // End of transliterator function

Diacritization Database queries (Insert & Select):

/* Insert query */
private: System::Void button5_Click
(System::Object^ sender, System::EventArgs^ e) {
    foundJustUsed=false;
    try {
        if ( !textBox4->Text->Equals( String::Empty ) &&
        !textBox5->Text->Equals( String::Empty ) &&
        !textBox6->Text->Equals( String::Empty ))
        {
            // create the SQL query to insert a row
            //***************
            oleDbDataAdapter1->InsertCommand->CommandText =
            "INSERT INTO `MyTable1` (`ID`, `Word`, `MorphWord`) VALUES
            ('"+textBox6->Text+"', '"+ textBox4->Text+"', '"+textBox5->Text+"')" ;
/************
textBox7->Text="\r\nSending query: "+oleDbDataAdapter1->InsertCommand->CommandText+"\r\n" ;
// send query
oleDbDataAdapter1->InsertCommand->ExecuteNonQuery();
textBox7->Text="\r\nInsertion successful !\r\n" ;
} // end if
else
MessageBox::Show("\r\nAll fields are required..\r\n") ;
} // end try
catch ( Exception ^Exception ) {
MessageBox::Show( Exception->Message, "Error, Word not inserted !",
MessageBoxButtons::OK, MessageBoxIcon::Error );
} // end catch

/* Select query */
private: System::Void button6_Click
(System::Object^  sender, System::EventArgs^  e)
{
String ^my_word="";
foundJustUsed=true;
my_word = textBox4->Text;
findWord=my_word;
try
{
if ( !textBox4->Text->Equals( String::Empty ) )
{
String ^myQuery = "SELECT MorphWord FROM MyTable1 WHERE Word
'"+textBox4->Text+"'">
OleDbCommand ^myCommand = gcnew OleDbCommand(myQuery, oleDbConnection1);
OleDbDataReader ^myReader; // works
myReader = myCommand->ExecuteReader();
myReader->Read();
textBox5->Text = myReader->GetString(0)->ToString();
myReader->Close();
textBox7->Text = "\r\nSending query:" +
"SELECT MorphWord FROM MyTable1 WHERE Word='"+textBox4->Text+"';";
else
textBox7->Text = "\r\nEntering Word is required !\r\n";
} // end try
catch ( Exception ^Exception ) { // catches all errors
MessageBox::Show( Exception->Message, "Word not Found!",
MessageBoxButtons::OK, MessageBoxIcon::Error );
} // end catch

TeleTool player (.wav):

private: System::Void button7_Click
(System::Object^ sender, System::EventArgs^ e) {
this->etPlayPro->SourceFileName =
"D:\SynthesisDatabase\ VOICE\ai,thath2,naa,nan2.wav"; // إثنان
this->etPlayPro->DeviceActive = true;