Universität Karlsruhe (TH)

Institut für Nachrichtentechnik

Fakultät für Elektrotechnik und Informationstechnik

Insitut für Logik, Komplexität und Deduktionssysteme

Fakultät für Informatik

Arabic Speech Recognition

Diplomarbeit von

JAMAL ABU - ALWAN

Hauptreferent:

Prof. Dr.-Ing. K. Kroschel

Koreferent:

Prof. Dr. Alex Waibel

Betreuer:

Dr.-Ing. Tanja Schultz

angefertigt am

School of Computer Science

Interactive Systems Laboratories

Carnegie Mellon University

Pittsburgh, 15213 PA, USA

July 2001

Ich erkläre hiermit an Eides statt, daß ich die vorliegende Diplomarbeit selbständig und ohne unzulässige fremde Hilfe angefertigt habe.
Die verwendeten Literaturquellen sind im Literaturverzeichnis vollständig zitiert.

Karlsruhe, den 9 Juli 2001 Jamal Abu Alwan

Acknowledgments

This work would not have been possible without the help of a number of people. First I would like to thank Professor Waibel for giving me the opportunity to accomplish this work at the Interactive Systems Labs at Carnegie Mellon University in Pittsburgh.

Special thanks is for Professor Kroschel from the Institute for Communication Engineering in the department of Electrical Engineering and Information Technology at the University of Karlsruhe. Without his permission I would not have been able to do the work in the computer science department, much less complete this work at Carnegie Mellon University.

Many thanks to the supervisor of my thesis, Dr. Schultz, for her excellent advice, guidance and patience throughout the entire project in Pittsburgh. She was always ready to help me at any time.

I would like to thank Sebastian Stüker for his advice which helped me very much.

I would like to thank Kerry Skemp for correcting the English and for input about the linguistic description of Arabic vowels.

Abstract

This work presents the development of a speech recognizer for Arabic as part of the GlobalPhone Project of the University of Karlsruhe.

Modern high Arabic was chosen because of its widespread use: it is the fourth most widely spoken language in the world. The Arabic language has several characteristics which do not exist in many western languages. In particular, Arabic is based on a system of 3-consonant roots which are marked for both inflection and derivation. The writing system closely adheres to the phonetic system, with the exception of the short vowels, which generally are not represented in the writing system. This fact necessitated vocalization (addition of these vowel phones) of the data used for this project.

This paper presents the fundamentals of speech recognition necessary for the development of this Arabic speech recognizer. Topics relevant to this project such as statistics, models and tools are addressed.

The pronunciation dictionary developed for this project is also described. This is a letter-to-sound tool implemented with Perl and requiring both Romanized and vocalized data. This dictionary takes into consideration the context of the phones.

The experiments are carried out according to four systems. There are also four evaluations for the systems.

The highest performance of the Arabic recognizer has a word accuracy of 62.6% for vocalized data and 69.7% for the same data in devocalized form.

The main problem in the work was the small amount of data (4 hours and 27 minutes), which made it difficult to build a very effective language model. However, considering the small amount of data, the results are good. Nonetheless, the results could be improved with certain changes in the project. Suggestions for improvement in future research are provided.

Contents

List of Figures

List of Tables

Chapter 1

Introdution

1.1 Speech Recognition Overview

Computer speech recognition works in a similar way to human speech recognition. In humans, the ear takes the sound in the form of vibration, and the brain decodes the signals and determines meaning. In the basic sense, this is also how computer speech recognition works.

In the first step of speech recognition, there are the sounds that are heard through the signal processing. A speaker says something and the speech recognition engine processes the utterances of the speaker, i. e. digitizes and converts them into spectral representations. These spectral representations of the speakers' spoken speech (words) have been divided into individual frequency components. This spectral representation of the speech is similar to the way the human ear works.

The next step is to encode or translate this spectral representation into phonemes. Phonemes are the basic sounds of the language like /f/ in 'fish' and /m/ in 'mother'.

It is very important to take into consideration other aspects of speech, such as dialects or accents, voice quality, noise conditions, speaking style etc. to distinguish among the individual phonemes.

Much research and progress in speech recognition technology over the past 20 years has gone into developing better statistical modelling techniques, which are able to distinguish more precisely the various speech sounds, even when faced with a wide range of variances.

Finally, the speech recognition application has to search through all possible items that might have been uttered so that it can come up with the most likely match. This search process is driven by grammars, pronunciation dictionaries and language models supplied by software developers.

Speech technology is processing rapidly. Better modelling techniques, increased robustness to accommodate cellular or IP (Internet Protocol) calls, increasing amounts of speaker data (i.e., more and more utterances of words and phrases), and the development of new languages rapidly advance the capabilities of today's speech engines. These advancements mean increasingly sophisticated applications and, more importantly, better experiences for people who use speech services.

1.1.1 Arabic Speech Recognition

In this work, a speech recognition system for Arabic has been developed. This work is part of the GlobalPhone Project of the University of Karlsruhe, Institute for Logic Complexity and Deduction Systems. This project is an answer to the need for multilingual speech recognition systems in our ever-globalizing world. One of the goals of the GlobalPhone Project is to create speech recognizers for a wide variety of languages. Since I am a native Arabic speaker, I was very interested in working on an Arabic speech recognizer for this project.

The data for Arabic already had been collected when I began to work on the speech recognizer. The data consisted of newspaper texts which were read aloud (i.e. dictated). All of these texts were in high Arabic (for a discussion of the various forms of Arabic, see section 2.1). The use of newspaper texts greatly increased careful pronunciation and grammatical accuracy, which makes the speech easier to recognize. Thus, the developed speech recognition system is a dictation system. For the development of the speech recognizer, the Janus Speech Recognition Toolkit was applied [8] (see section 3.3.2).

The main components of creating the Arabic speech recognizer were the development of a pronunciation dictionary, an acoustic model and a language model.

Information about phonemes (on which the pronunciation dictionary is based) is necessary for creating an acoustic model. Each phone will be modelled with a three-state HMM[1]. This topic will be explained in more detail in chapter 3. The language model calculates the statistical probability of word sequences. This is done through language modelling (see section 3.5).

1.2 Purpose of this Work

The goal of this work is to develop a capable automatic speech recognizer for high Arabic. Articles from local newspapers about politics and economics (collected from Arab students from the University of Karlsruhe) were used as a basis for the Arabic speech data.

1.3 Structure of this Work

This work begins with a brief description of the various dialects of Arabic and the reasons for selecting the chosen dialect (Chapter 2). Other topics discussed in this chapter are the Arabic writing system, Romanization and the need for vocalized data. A phonetic analysis of Arabic is also provided. The implications of these topics for this speech recognition system are addressed.

Chapter 3 provides an overview of the fundamentals of speech recognition for this project. Specific topics include preprocessing, Hidden Markov Models

[1]Hidden Markov Models

(HMMs), the Viterbi Algorithm, Language Modelling, Acoustic Modeling and the Janus Speech Recognition Toolkit.

Chapter 4 discusses the pronunciation dictionary. It also addresses the specific problem of the construction of numbers in Arabic.

Chapter 5 the focuses on the speaker database, training and test data, and language model data.

Chapter 6 described the experiments and their results.

Finally, in Chapter 7 the conclusion and future work are presented.

Chapter 2

The Arabic Language

2.1 Selection of Arabic Dialect

The Arabic language belongs to the family of Semitic languages. Arabic is considered to be the most widely spoken language in this family. It is spoken by approximately 220 million people (Pittsburgh Post-Gazette, June) . I would like to give a brief and short introduction about the Arabic language.

The Arabic language contains three forms and they are:

- The classical form of Arabic as it is in the Koran. The Koran Arabic is very difficult to read for most Arabic speakers. It requires full concentration from the reader. Classical Arabic used to be very important in the 9-16 century A.D. because of the influence of Islam in that time from Persia across northern Africa to the Atlantic coast. As result, classical Arabic was a very important lingua franca. There was mutual borrowing between classical Arabic and other languages such as Greek, Hebrew, Aramaic, Turkish, Spanish, Farsi, etc.

- Modern high Arabic, which is the official form of the language used in most Arabic media, was developed from classical Arabic. This form of Arabic is the basis for learning the language in schools and universities. As a result, nearly every Arabic native speaker who learns Arabic in school is able to speak and write in high Arabic in addition to their regional dialect. Since modern high Arabic is mutually intelligible, it is the dialect used among Arabs from different parts of the Arab world when they interact.

- Low Arabic dialects. Most Arabs, however, speak a regional dialect of low Arabic with their families, friends and in their daily lives. The pronunciation differences in these dialects are great. It is very often difficult for an Arab native speaker from the Middle East to understand the dialect of many north African Arabs. Furthermore, regional dialects often contain many foreign words from languages of surrounding countries or regions.

Today modern high Arabic has the status of official language in most Arab countries. Since this is the most widely understood and mutually intelligible form, it was chosen for the speech recognizer in this work.

2.2 The Arabic Phonetic System

All Arabic phonemes (vowels, consonants and diphthongs) can occur in initial, medial and final position in a word.

There are six Arabic vowels, three short and three long. The short ones are called Fatha /a/, Damma /u/ and Kasra /i/ (see table 2.1 for the phonetic symbols and their sound description) . These vowels are written either above the word (Fatha and Damma) or under the word (Kasra). To illustrate this, we can consider the example in which Fatha and Damma are written over the Letter B and the Kasra is written below:

بَ، بُ، بِ

The three long Vowels are Alif /A/, Waw /U/ and Yaa /Y/ and they are written in Arabic from right to left:

أ ، و ، ي

The symbols Waw and Yaa also can represent diphthongs: /aW/ and /aY/, respectively.

Arabic Symbol	Phonemic Symbol	Sound Description
اَ	/a/	short front open unrounded vowel
اِ	/i/	short front closed unrounded vowel
اُ	/u/	short closed back rounded vowel

Table 2.1: The Arabic short vowels

Arabic Symbol	Phonemic Symbol	Sound Description
اَ	/A/	long front open unrounded vowel
و	/U/	long closed back rounded vowel
ي	/I/	long closed unrounded vowel

Table 2.2: The Arabic long vowels

Arabic Symbol	Phonemic Symbol	Sound Description
و	/aW/	diphthong between /a/ and /u/
ي	/aY/	diphthong between /a/ and /i/

Table 2.3: The Arabic Diphthongs

In contrast to the consonants [15, 16, 17, 18, 19], the vowels have no segment duration changes, except that long vowels are about twice the length of short vowels. Vowels have other contextual changes, however, which are described in this section. A vowel is represented by V.

Arabic has 28 consonants. Consonants are represented by C. The durations of intervocalic and initial consonants are about half those of syllable closing or syllable suffix consonants. Aside from this, very few consonants are significantly affected by their context.

Vowel-consonant transitions are designated by TRVC. In medial positions of Arabic words, any of the six vowels can occur before any of the 28 consonants and they are:

ب ، ت ،ث ، ج ، ح ، خ ، د ، ذ ، ر ، ز ، س ، ش، ص ، ض ، ط ، ظ ،
ع ، غ ، ف ، ق ، ك ، أ ، ل ، م ، ن ، ه ، و ، ي

Thus, the possible number of such units is 6 * 28 = 168. Acoustically, the TRVCs include the transitional part from a vowel into neighboring consonant and steady state portions of the vowel and the consonant.

The consonant-vowel transitions are also designated by TRVC . Any of the 28 consonants can occur before any of the six vowels. Thus, the possible number of such units is 28 * 6 = 168. Acoustically, a TRVC unit includes the transitional part from a consonant into a vowel plus steady-state portions of each.

The Arabic diphthongs [16], as shown in table 2.3, are formed with the glides (/aW/ und /aY/) and occur as non-geminate sounds after the short vowel /a/ at the end of a syllabls for type CVC: or as core final consonants for syllables of type CVC: C. The Arabic diphthongs can be defined as monosyllabic gliding sounds that start at the articulation of the regular Arabic vowel /a/ and move into the articulations of slightly centralized versions for either the vowel /i/ or /u/. During this motion a diphthong transists through the sounds for either glide (/aW/ or /aY/), but that sound cannot be maintained.

Arabic Symbol	Phonemic Symbol	Sound Description
ب	/B/	a voiced bilabial plosive
ت	/T/	a voiceless dental nonemphatic plosive
ث	/TH/	a voiceless lingua dental fricative
ج	/J/	a voiced palatal fricative
ح	/H7/	a voicless pharyngeal fricative
خ	/KH/	a voicless velar fricative
د	/D/	a voiced dental non emphatic fricative
ذ	/DH/	a voiced lingua dental
ز	/Z/	a voiced aveolar fricative
س	/S/	a voiceless alveolar non emphatic fricative
ش	/SH/	a voiceless palatal fricative
ص	/SD/	a voiceless alveolar non emphatic fricative
ض	/DD/	a voiced dental emphatic plosive

Table 2.3: The Phonetic Symbols of Arabic Sounds Part I

Arabic Symbol	Phonemic Symbol	Sound Description
ط	/TT/	a voiced bilabial plosive
ظ	/DS/	a voiceless dental nonemphatic plosive
ع	/E3/	a voiced pharngeal fricative
غ	/GH/	a voiced uvular fricative
ف	/F/	a voicless labio dental fricative
ق	/Q/	a voicless uvular plosive
ك	/K/	a voiceless velar plosive
أ	/q/	a glottal stop
ل	/L/	a voiced alveolar lateral
م	/M/	a voiced bilabial nasal
ن	/N/	a voiced alveolar nasal
ه	/H/	a voiceless glottal fricative

Table 2.4: The Phonetic Symbols of Arabic Sounds Part II

2.3 Romanisation

The term 'Romanisation' means the transliteration of a non-Latin script into the Latin script. Romanisation is used in the development of speech recognizers for languages with other writing systems. The Romanisation is necessary because Arabic writing is written from right to left (and, thus, is problematic for speech recognizers) and because most people can not read the letters at all. Thus, all of the Arabic data for this project had to be romanized.

A specific tool was developed for the Romanisation using Tcl script [9].

For the Arabic data that we ordered from LDC[1] there was a need to romanize the data. The LDC corpus contains articles published by the Associated French Press News Agency from 1994 until 2000. The data was decoded in UTF-8 Format and a Tcl script was developed to romanize this format. The LDC had a vocabulary size of 606, 000 and text size of 50 million words.

2.4 The Arabic Writing System

Arabic writing system is different from the writing systems of many other languages because words are written from right to left.

An interesting feature of this writing system is the fact that each Arabic letter has different forms depending on its position in (or outside) a word. For example, the letter called 'Ba' /B/ can be written in three ways:

'Ba' word-initial 'Ba' word-medial 'Ba' word-final.
The following example illustrates these three cases (arab words form right to left) where 'Ba' is the letter which written with point underneath.

بَيت نَبَتَ كَتَب

Another feature of the Arab writing system are the short vowels, which are indicated by diactrics (instead of letters) written above or below the

[1]Linguistic Data Consortium

consonants. An example is:

/RaSaMa/ 'to draw' has the arabic vocalized form

رَسَمَ

As mentioned in the following section these diacritics are often left out entirely, which makes an intermediate step ('vocalization') necessary. The above word without the diacritic is written as follows:

/RaSaMa/ 'to draw' has the non vocalized form

رسم

Words written with diacritics are called 'vocalized'; words without diacritics are referred to as 'devocalized'.

2.5 Vocalization

In addition to Romanisation, another step was needed in order to make the data suitable for the speech recognizer. Namely, the data needed to be vocalized, i.e. the short vowel sounds needed to be included. Arabic is generally written without diacritics (signs which represent the short vowels and can be written above or below a letter).

The data available for this project had already been romanized without vocalization. Thus, vocalization was completed after Romanisation

With the exception of texts for children who are learning to read, Arabic writing is not vocalized. This means that the short vowels are spoken, but not written. Thus, when Arabic texts are Romanized for a speech recognizer, the short vowels must be added. This step is called 'vocalization'.

Although vocalization of the Romanized transcripts is a time-intensive job, without this step it is impossible to develop a functional Arabic speech recognizer. The importance of the vocalization will be discussed further in the following section on Arabic morphology.

2.6 The Morphology of the Arabic Language

The mechanism by which Arabic vocabulary is created is called *Al-ishtiqaq* which means *derivation* in English. This means that from a base form many words can be derived. Most Arabic words are represented by a base from of 3 consonants called a 'root' or 'template', as represented in the following example:

HSB حسب

has many meanings which depend on the vocalization

HaSiBa	'he thought'	حَسِبَ
HaSaBa	'he calculated'	حَسَبَ
HaSBa	'according to'	حَسبَ
HaSaB	'of noble ancestry'	حَسَب

Another example is the root form KTB كتب

KaTaBa	'to write'	كَتَبَ
KaATiB	'author'	كَاتِب
KiTaAB	'a book'	كِتَاب

Each of the above examples changes is derivational, i.e. the syntactic word class is changed (noun, verb, adjective, etc.).

In addition to the derivations based on these roots or templates, there is also inflection in Arabic. Some of the categories marked by inflection are singular/plural, gender, case (nominative, accusative, dative, etc.) and verb tense. A few examples are:

AeaKTuBu	'I write.'	أَكْتُبُ
YaKTuBu	'He writes.'	يَكْتُبُ
TaKTuBu	'She writes.'	تَكْتُبُ
NaKTuBu	'We write.'	نَكْتُبُ

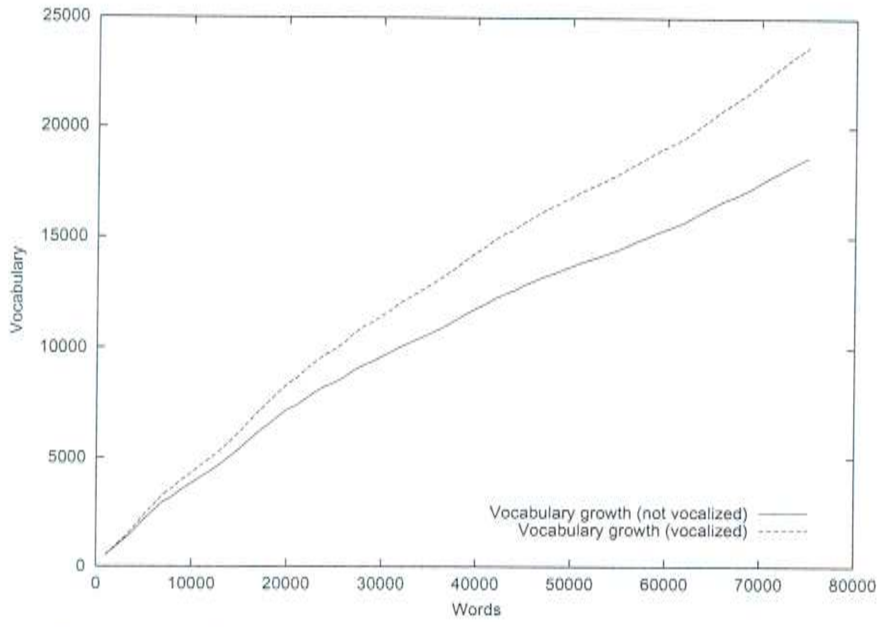

Figure 2.1: Vocabulary growth in vocalized and devocalized corpus

Figure 2.1 depict the running text size and the vocabulary growth for our corpus data in a vocalized and a devocalized form. It is obvious that the vocabulary size for the same corpus in devocalized form has been decreased from 24,000 to 18,500. The reason for this difference is the similarity among derivational roots. In other words, if the data is not vocalized, a root such as

HSB could be recognized as 'he calculated', 'according to', 'of noble ancestry', etc. The probability of accurate recognition is greatly decreased because it is not possible to understand the context, even if the *word error rate* WER (see section 6.1) is very low. If the system recognizes 'he calculated' instead of 'acoording to' the whole meaning of the utterance will be lost.

Thus, the provision of vocalized data is crucial for the acoustics of the speech recognizer. In sections 6.3 and 6.5, the results of the recognition performance for vocalized and devocalized data are shown.

Chapter 3

Fundamentals in Speech Recognition

3.1 The Fundamental Problem of Speech Recognition

Statistics are an essential component of speech recognition. Most modern speech recognizers are based on statistical and pattern recognition methods, especially for LVSCR[1]. For a given unknown speech signal $X = x_1, x_1, \cdots, x_n$ of the statistical recognition system has to find the sequence of words $W = w_1, w_1, \cdots, w_n$ given the parameterised acoustic signal X, i.e it has to find the word sequence $\hat{W}$ that maximize $P(W|X)$.

By using Bayes' formula the desired probability can be formed as follows:

$$\begin{aligned} \hat{W} &= argmax_w\, P(W|X) \\ &= argmax_w\, \frac{P(W) \cdot P(X|W)}{P(X)} && (3.1) \\ &= argmax_w\, (P(W) \cdot P(X|W)) && (3.2) \end{aligned}$$

The term $P(W)$ represents the ***a priori*** probability which observes the word sequence W is calculated by the ***language model*** (LM). The process for

[1]Large Vocabulary Continuous Speech Recognition

finding $\hat{W}$ is called *decoding* and requires solution of a number of a difficult problems.

The term $P(X|W)$ represents the probability of observing the signal X given the signal W. This value is determined by an *acoustic model* (AM) of a speech recognizer. In figure 3.1 there is a description of how a speech recognition system works.

3.2 Preprocessing

This is just a brief introduction of the tasks of the preprocessing. A detailed description about preprocessing for speech recognition is found in [1] and [5]

.

The preprocessing serves to emphasize the significant characteristics of the speech, to suppress interfering noises, and to transfer the speech signal to a normalized representation in which differences among speakers will be equalized. For example, different speakers have vocal tracts of varying lengths, which influence the speech signal. This can be normalized with the help of the *vocal tract length normalization* (VLTN) technique [29]. Other techniques that are used include *linear disciriminant analysis* (LDA) [30] is an efficient technique to reduce the dimension of the feature vector without loss of relevant information.

The first step in preprocessing is to transform the signal into a spectral representation. Most preprocessing systems use a sliding window with a length between 5 ms and 20 ms in order to extract "frames" of samples from the speech waveform. These frames are extracted every 10 to 20 ms. After that the discrete Fourier transform [10] is required to transform a frame of time samples into a spectral representation.

The resulting[2] spectral coefficients can be reduce to 16 values if we have

[2]in our system there are 256 coefficients

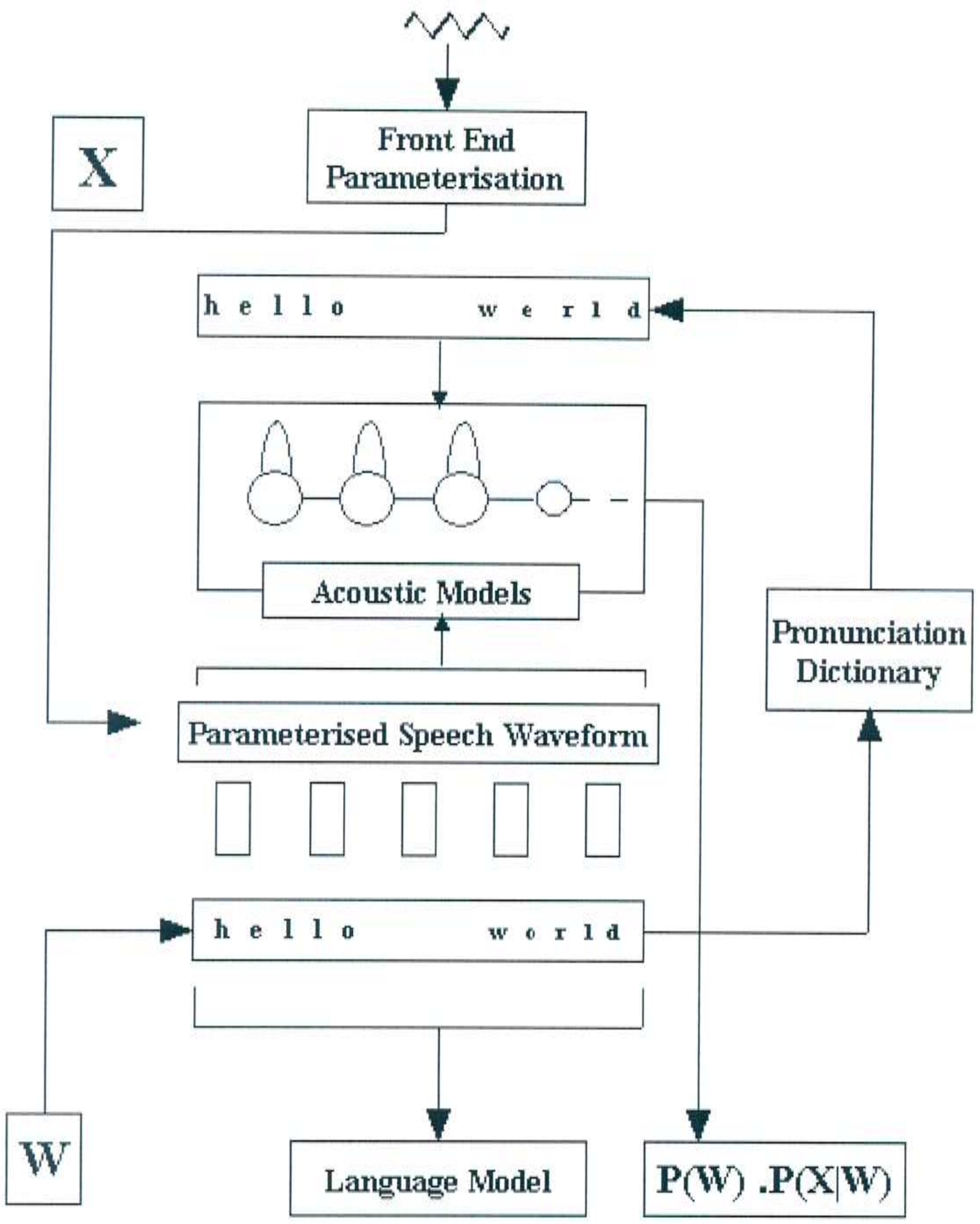

Figure 3.1: Overview of Speech Recognition System

additional knowledge about the sensitivity of the human ear. This mapping results in features such as Mel-scale [4].

3.3 Acoustic Modeling

The purpose of the acoustic model is to provide a method of calculating the conditional probability $P(X|W)$ (see 3.1). This is just a brief introduction of the tasks of the acoustic modeling. A detailed description about acoustic modeling is found in [7, 31, 32].

3.3.1 Hidden Markov Models in speech recognition

Speech is perceived as the chronological sequence of sounds (phones). Thus, the meaning of a word is derived from the sequence of individual phones, for example 'tab' vs. 'bat'. Models of pattern recognition in which such a chronological sequence is considered are called Hidden Markov-Models (HMM) [2, 3] which are used in the most state-of-art systems.

In such models, conditions are possible with condition-dependent distributions. In speech recognition, conditions are often modelled after components of a word, for example a phoneme. Thus:

A Hidden Markov Model is a five-tuple $\lambda = (S, P, O, A, \pi)$ consisting of:

- The set of states $S = \{s_1, s_2, \cdots, s_N\}$
- The set of emission probability distributions (densities), $P = \{p_1, p_2, \cdots, p_n\}$ where $p_i(x)$ is the probability of observing x when the system is in state s_i
- Observation alphabet O
- The matrix of state transition probabilities: $A = (a_{ij})$ where a_{ij} is the probability of state s_j following s_i

- The initial probability distribution $\pi = \pi_j$, $1 \leq j \leq N$ of the j being the first state of a state sequence

In Figure 3.1 a simple HMM with 2 states $S1, S2$, two self loops to remain in the current state and their state transition probabilities distribution is represented.

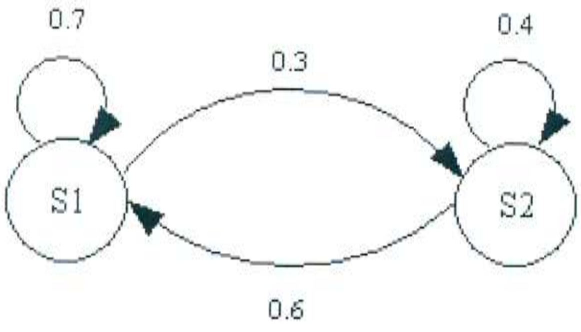

Figure 3.2: Simple HMM with 2 states

A HMM has definite model characteristics. A characteristic is determined by how many states can be employed to model a phoneme. Usually three states or more can be employed for each phoneme.

Since the speech signal differs between the borders and the center of a phoneme, each phoneme model can be represented by a three state left-to-right HMM as three frames roughly correspond to the average length of a phone unit. These three states represent the beginning, the center and the end segment of the phoneme. Each state has a self-loop to remain in the current state and a transition to the next state and sometimes there is a transition skipping one or more states. Figure 3.2 shows a possible realization for a phone where $a_{11} = 0.7$, $a_{12} = 0.3$, $a_{21} = 0.6$ and $a_{22} = 0.4$ are possible state transition probabilities. A word which contains three phones can be modelled with 9 states where every 3 states model exactly one phone.

Three basic problems are associated with a HMM :

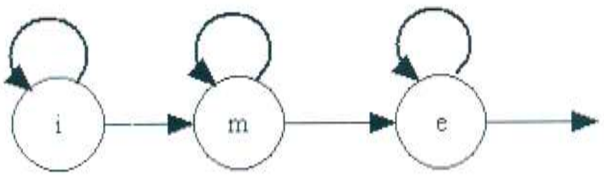

Figure 3.3: Possible HMM model for one phone

- The optimization problem
  How with an observation O the model parameter can be determined so that the model corresponds better to the observations.
- The evaluation problem
  How probable it is that a given characteristic sequence O from model λ is produced.
- The decoding problem
  Given the observation sequence $O = O_1, O_2, \cdots O_T$ and a model λ, how do we choose the corresponding state sequence $Q = q_1, q_2, \cdots q_T$ which is optimal in some meaningful S.

3.3.2 Language Modeling

The words which a speech recognizer can recognize constitute its vocabulary and are assembled in a dictionary. This dictionary additionally contains the description of the composition of the words from phonemes. This dictionary can also contain so-called nonsense words for the modelling of spontaneously spoken effects such as stuttering, stammering and breathing. In order to support the speech recognizer a model for the expected expressions (language model) is applied.

The language model computes the probabilities of a given sequence of words $W = w_1, w_2, \cdots, w_n$ and is given by:

$$P(W) = P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w1 \cdot w_2) \cdots P(w_n|w_1 \cdot w_2 \cdots w_{n-1}) \quad (3.3)$$

According to this equation, the language model considers the word history $w_1, w_2, \cdots, w_{i-1}$.

How does one attain a statistical language model? Let us begin with the fact that in the LDC corpus (see 2.3) there are approximately 606,000 vocabularies and that most of these appear very rarely. For a vocabulary size of M there are M^{i-1} different histories. In case of Arabic LDC corpus, we will have $606,000^{i-1}$ different histories. The estimation of probabilities necessitates an enormous quantity of word sequences. The most common language modelling approach is called *3-gram* model. In this model, the histories are considered equivalent if they end in the same two words.
This model is given by

$$P(W) \approx \prod_{i=1}^{n} P(w_i|w_{i-2}, w_{i-1}) \quad (3.4)$$

Even when only a limited history of three words (*trigram* language model) or two word (*bigram* language model) are considered, the necessary data quantity for a certain estimation is enormous. For this reason, various procedures were developed to estimate unseen *trigram*- or *bigrams*, in which, for example, an interpolated probability is applied or a so-called *back-off* probability is used. A statistical language model can then be interpreted from one word model to another word model (bigram) as a transition probability in a HMM.

For more discussion about language modelling see [11] and [12]

3.4 Speaker Adaptation

Speaker adaption is a technique used to adjust the acoustic models of a speaker independent recognition system to a specific speaker in order to increase the recognition performance. The speaker adaptation is devided into supervised and unspervised class. The real textual transcription in the case of supervised adaption is known and unknown in the case of unsupervised adaptation. For more information about speaker adaptation please see [14]

3.5 The Janus Speech Recognition Toolkit

In order to accomplish this work, it was necessarily to have a speech recognition toolkit. The Janus Recognition toolkit (JRTK) was chosen for this. This toolkit is embedded into a TCL/TK interpreter and has been developed at the Interactive Systems Labs located jointly at Carnegie Mellon University and the University of Karlsruhe. The script language Tcl/TK allows easy programming of graphical user interfaces. This makes looking at data structures and controling the recognizer easy. For more information about Janus please see [21] [22] [23] and [24].

Chapter 4

Pronunciation Dictionary

4.1 Description of the Problem

In order to develop a capable automatic speech recognizer, we need to create a pronunciation dictionary.

The pronunciation dictionary specifies the finite set of words that may be output by the speech recognizer, and gives at least one pronunciation for each vocabulary unit (i. e. phone sequence).

In order to begin the development of a pronunciation dictionary, the following things are necessary:

- Romanized Data: Arabic as well as some other languages (Chinese, Japanese, etc.) use non-Roman characters that are very difficult to process with traditional software. Since all software and hardware were designed to handle only the Roman Alphabet, it is often much easier to translate the language characters into Roman characters. For more information about Romanization of the data, see section 2.3.
- fully vocalized transcriptions. Since the short vowels in Arabic (see section 2.2) are written either above or under the word, there is a need to complete these short vowels in the transcriptions. Unfortunately, it

is very difficult to find vocalized Arabic text data because Arabic native speakers are able to read texts without the vowels. Thus, the vowels must be added into the Arabic text data. For more information please see section 2.4.

4.2 Realization of the Dictionary

The pronunciation dictionary (described in section 4.1) for this work is a letter-to-sound tool which provides the user with the pronunciation of an Arabic word derived from the Roman script.

According to the results of earlier work, we know that in order to enable a speech recognizer to function, we need at least a small portion of spoken data which are written first in the Arab script and then systematically transcribed into Roman script. The Romanized version includes representation of the vowels.

In this work, many problems were encountered in the transcription. The main problem was erroneous correspondence of symbol to phoneme. The phonemic inventory of Arabic was incorrectly described as containing more than one phoneme for a single phoneme that has different spellings. A parallel example from English is the phoneme /f/ which can be written as 'f', 'ff', 'ph' or 'gh', but is still only one phoneme. The Arabic data contained a different phonemic representation for each spelling difference. Thus, it was very important to remove information from the transcription that was not needed or should have been ignore by the recognizer. Although it is possible to text-process the transcriptions while training, it is preferable to have transcriptions that are easy to read by both humans and computer.

The tool has been implemented with the programming language Perl because it is very suitable for such purposes. The implemented script contains two important parts:

- Transformation of the transcription units into phones.

This part contains the phonetic transcription of the actual units, i.e. the transcription of the Roman form into a pronunciation form. Due to the variety of sounds in Arabic and the fact that there are several letters in Arabic that have no equivalent in English, we were required in the Roman form as well as in the pronunciation form to give these Arabic letters (phones) two Roman letters. This is very clear to see in the subphones. For Example Sd, Tt, Te, are phones and their transcription is SD, TT and TE. In some cases we were even compelled to make a combination with a digit like 3 or 7, which has been transcribed into E3 and H7.

- phonological changes. This part contains some grammatical rules, which are changeable at any time. Due to the great number of rules in Arabic (some of which are very rare), it is impossible to consider all these rules in these tools.

The main problem in producing the pronunciation dictionary is that the pronunciation in many cases depends on the phone that comes before or after, i.e. the context of a phone. Sometimes it even depends on the preceding word.

This problem was already partially solved because, during the vocalization of the speech data, we accentuated these unpronounced phones with brackets. Because of this these brackets must be considered in the script.

We search for these words in the main part of the script and, if they exist, we let the program progress to the subroutine, which will be called 'Variant'. In this subroutine we remove the brackets and apply the pronunciation and consider two cases:

- Pronunciation with consideration of the phones within the brackets.
- Pronunciation without consideration of the phones within the brackets. If the word appears more than one time, we register the version number in the dictionary.
- A combined form of the first two cases

Here is an example for clarification: The word is: AL-MiNTaQa(Te): Pronunciation in the form that is in our dictionary according to Janus Format will be:

- {AL-MiNTaQaTe} : A L M i N T a Q TE
- {AL-MiNTaQaTe(2)} : A L M i N T a Q a

The 2 means that there are two different pronunciations for this word. In many cases we have even two pairs of brackets in one word. That means two phones are not pronounced.

Let us take another example for the same word (A)L-MiNTaQa(Te): with 2 pairs of brackets.
In this case, we will have 4 different registrations for this word in the dictionary and they are as follows:

- {AL-MiNTaQaTe} : A L M i T a Q a TE
- {AL-MiNTaQaTe(2)} : A L M i N T a Q a
- {AL-MiNTaQaTe(3)} : L M i N T a Q a TE
- {AL-MiNTaQaTe(4)} : L M i N T a Q a

In this way, we cover all different variations in the pronunciation of these words.

For example, the phone which is represented in Arabic by the letter *Ta Murbota*

ة

is a form of /t/ that comes at the end of word. It will be pronounced only if it is vocalized, i.e. if it is followed by a vowel.

To complete the dictionary it was very important to consider several important letter-to-phoneme rules. The rules can be classified as: omission, definite article assimilation, special letters and diacritic symbol conversion, long vowel generation.

The omission rules apply to letters which appear in writing without having any phonemic significance, for example, the letter /A/ of the Arabic *Alif* when it appears at the end of a word before *Tanwin Fatha* /an/.

The definite article assimilation rules apply to the Arabic letters for the definite article "AL" which always occur word-initially. These rules are called *Alif Lam Ashamsiya* and *Alif Lam Al-Qamariya* rules.

Arabic letters can be classified as *Shamsi* or *Qamari* symbols. There are 14 Shamsi and they are:
T, Th, D, Dh, R, Z, S, Sh, Sd, Dd, D , L and N (Arabic letters from right to left)
ت ، ث، د، ذ، ر، ز، س، ش، ص، ض ، ط ، ظ ، ل ، ن
and 14 Qamari which are the rest 14 letters in the alphabet.

The Shamsi letters include coronal sounds which are produced with the tongue blade. The Qamari letters include all sounds which are produced either at the far front or far back of oral tract.

The rules for Shamsi and Qamari are as follows:
If a word begins with a Shamsi sound and is preceded by the definite article, the /L/ of the definite article is assimilated to the sound of the word that begins with Shamsi sound. The phone /l/ thus disappears. This is illustrated in the following example:
{AL-ShShaMS} A **SH** SH a M S
The 'SH' is a shamsi sound.

On the other hand, if the word begins with a Qamari sound, the /l/ of the definite article will not be assimilated. This is illustrated in the following example:
{AL-BaYT} A L **B** a Y T
The 'B' is a Qamari sound.

The gemination symbol Shedda indicates that the consonant over which it appears should be repeated. All Arabic consonants can be repeat themselves and are pronounced twice. This is illustrated in the following example:
{FaLaSTtiNiYYiN} F a L a S TT i N i **YY** i N

The 'YY' is a double Phones represents the assimilation of Y .

The diacritic symbols Fatha, Damma and Kasra (Short vowels) are used for sounds of the short vowels /a/, /u/ and /i/ respectively. Also the diacritic symols *Tanwin Fatha*, *Tanwin Damma* and *Tanwin Kasra* are symbols for /an/, /un/ and /in/ and can be pronounced as the phoneme sequences /A/ /N/, /U/ /N/, and /I/ /N/, respectively.

The long vowel generation rules are:

- /A/ is produced when the letter symbol for Alif is preceded by diacritic symbol for the short vowel /a/ and is not followed by any short vowel.
- /U/ is produced when the letter symbol for Waw is preceded by diacritic symbol for the short vowel /u/ and is not followed by any short vowel.
- /I/ is produced when the letter symbol for Waw is preceded by diacritic symbol for the short vowel /i/ and is not followed by any short vowel.

These are the rules which have been considered during the development of the dictionary.

Future work may include the extension of the letter-to-phoneme- rules in order to make this tool more powerful.

4.3 Construction of Numbers in Arabic

Due to the quantity of numbers in the transcripts it was very important to consider them in the dictionary. Otherwise, there would be too many errors. That means that our dictionary should be extended to contain the pronunciation of numbers.

The construction of numbers in Arabic is similar to German. For example, the number 21 in Arabic is said 'one-and-twenty'. The construction itself is fairly simple, but realizing a tool for the numbers is very difficult for the following reasons:

- the numbers in Arabic are inflected according to their grammatical status, for example, 30 can be read as ThaLaThiYN or ThaLaThuWN, ThaLaThuWNa, ThaLaThiYNi.
- There are also masculine and feminine forms for numbers. This also complicates the realization of the tool.
- Although our transcriptions are in high Arabic, many speakers read the numbers in their own dialect, which makes the development of a tool for numbers difficult and inefficient.

Due to these complexities, I developed a tool which produces the Roman form of digits and numbers. I considered in this tool numbers with between 1 and 6 digits.

The program was implemented with the programming language Tcl/Tk. It can easily be extended to work with numbers longer than 6 digits. In addition, it can give the Roman form of floating-point numbers.

If we run this program separately, the output will be in roman form. In order to obtain the pronunciation form, we have to execute it using the first program without the need to change anything. In this way, we guarantee that changes in the transcription will be done one time in the first program only without the need to do changes in the second one.

Changes are easy to make in this program. The program can be reconstructed very quickly in order to fulfill the needs of other Arabic dialects.

Chapter 5

The Recognition System

5.1 Speaker Database

For the development and evaluation of the Arabic speech recognizer in this work we used the collected *GlobalPhone* database [25, 26, 27]. This database consists of 4 hours and 27 minutes of speech for the training and 83 minutes for testing by 44 native Arabic speakers. Each speaker read a few articles from Arabic newspapers. These articles were about national and international politics and economics. The speakers normally speak Tunisian and accented Palestinian dialects, but they read the articles in high Arabic. The dialect of the speakers was noticed in the speech data even that they spoke high Arabic.

For the recording of the speech data, a microphone was connected to the DAT-recorder. The speech data were recorded on a stereo with a sampling rate of 48 KHZ. After the transfer to the hard disk this was downsampled to 16 KHZ, 16bit.

Table 5.1 provides us with some information about the speaker, such as sex and dialect. This information is important for defining the training, test and cross-validation sets. All these sets should represent the speakers of Tunisian and Palestinian dialects according to their percent coverage from the total number of speakers.

Number of Speakers	44
Female	26
Male	18
Dialect : Tunisien	27
Dialect : Palestinian	17
Utterances	2897
Text Size	69k
Vocabulary Size	27k

Table 5.1: Speaker's Description

5.2 Training and Test Data

For the development of a speech recognizer we need to divide the speech data into a *training set*, *test set* and a *cross-validation set*. The division is done with 8, 1, 1, i.e. 80% training, 10% test, and 10% cross-validation. All these sets must be disjunct.

Thirty six of the speakers were used for training the acoustic models. They spoke a total of 2,213 utterances with a vocabulary size of 22,650. The four test speakers spoke a total of 381 utterances with a vocabulary size of 5,425 and OOV[1] rate of 32.63%.

The remaining four speakers for the cross-validation set spoke a total of 303 utterances with a vocabulary size of 4,042 words and an OOV rate of 32.02%.

For the experiment (see chapter 6) we tested the recognizer for the vocalized and non-vocalized data. Figure 5.1 shows us the vocabulary growth for the corpus. Figure 5.2 shows us the same non-vocalized corpus.

[1]Out Of Vocabulary means words in text files which are not covered by words in the train corpus

	Training Set	Test Set	Cross-Validation Set
Number of Speakers	44	4	4
Utterances	2213	381	303
Vocabulary Size	22,650	5,421	4042
OOV rate	–	32.63%	32.02%
Total utterances	2897		
Total vocabulary	27,134		

Table 5.2: Description of acoustic database

5.3 Language Model Data

To build a good speech recognizer we need a language model (LM) in addition to the acoustic model.

In order to build a good language model, it is important to have a sufficient quantity of data to reduce the OOV. In this work, the OOV of 32.63% for 22k vocabulary items is extremely high, and there was a need to collect adequate data to reduce the OOV. For example in English we have an OOV smaller than 0.5% with 65k vocabulary size.

Due to the fact that the GlobalPhone data was not sufficient to build a good language model, there was a need to find data from other sources.

However, this need could not be met because it was impossible to find vocalized Arabic data. None of the Arabic newspapers or internet sites use vocalized data. Through our experience on the GlobalPhone project, we discovered that vocalization by experts of Arabic texts is a huge task which requires a lot of time and money.

We managed to find non-vocalized Arabic data from LDC (see chapter 2.3). The purpose of this data was to build a language model from a big corpus with a large vocabulary size in order to reduce the high OOV rate. The high OOV is very clear to see in figure 5.1 which shows us the self- and

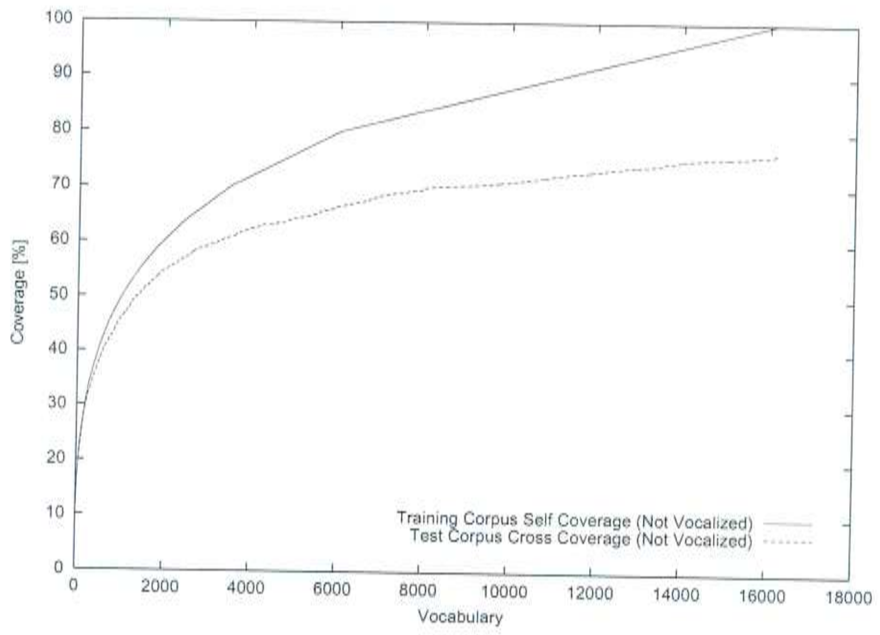

Figure 5.1: Self- and cross- coverage in the vocalized corpus

cross- coverage[2] Figure 5.2 shows us the coverage for the same devocalized corpus and figure 5.3 shows us the self- and cross-coverage for the vocalized and devocalized corpus together.

- language model I with training data and vocabulary of all corpus data. In this case, OOV = 0.
- language model II with training data and vocabulary of all training data. In this case, OOV = 32.63% .
- language model III with the non vocalized NewWire corpus and vocabulary of all devocalized training data, OOV = 21.75%

All the details for the three language models are represented in table 5.3 and table 5.4

[2]Coverage of the vocabulary in the text corpus itself in the first case and in the second case the coverage with a test corpus.

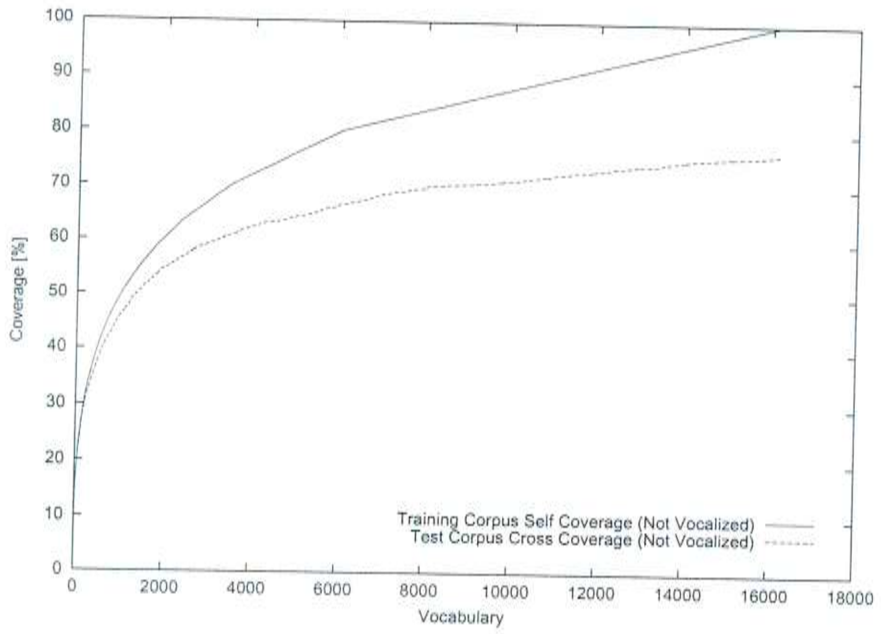

Figure 5.2: Self- and cross- coverage in the devocalized corpus

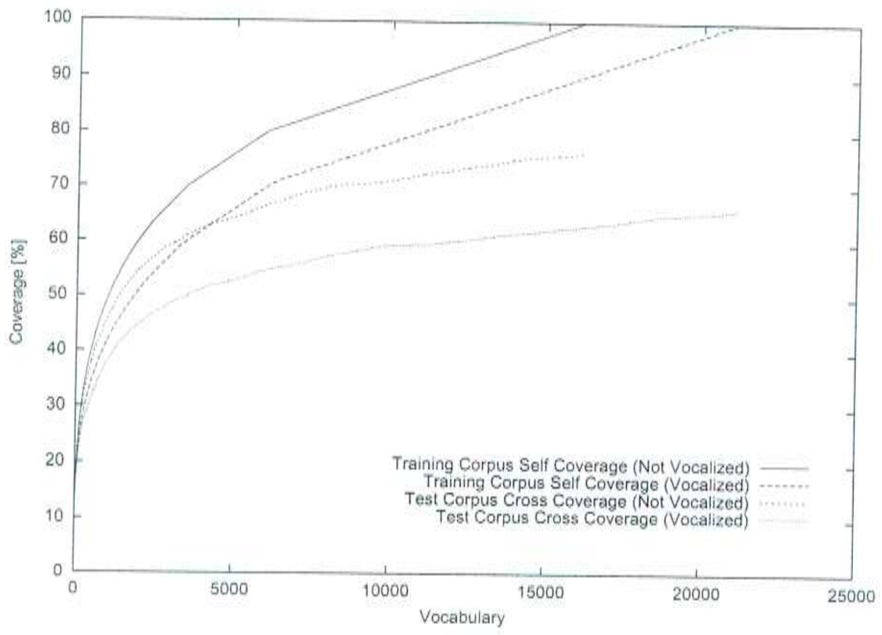

Figure 5.3: Self- and cross- coverage in the vocalized and devocalized corpus

LM	Corpus	Vocabulary
LM I	training data	training- and test data
LM II	training data	training data
LM III	NewsWire	training data

Table 5.3: Language Modell facts

	Language model I	Language model II	Language model III
Corpus size	$75k$	$75k$	57.5 Millions
1-gram used	82.56%	65.1%	42.51%
2-gram used	14.6%	26.2%	30.77%
3-gram used	2.842%	8.701%	26.72%
Perplexity	5078	816.4	2241
OOV	0%	32.63%	21.75%

Table 5.4: facts about language model

If we compare the results of the three language models, we can conclude that the OOV in the second language model is high. In contrast, the language model III OOV is ca. 12% lower because the corpus in this model is devocalized and because there is more words to cover the test corpus. The perplexity which indicates the average word branching factor, i.e. the average number of words that can follow the current word, is in all three language models high. The explanation for the high perplexity is because Arabic very rich in vocabulary. It is also important to mention that the NewsWire data is devocalized and, for this reason, we devocalized the training data in order to have all the data in devocalized form.

It is apparent from Table 5.4 that the 1-, 2-gram calculation in the first and the second language models is very high; in the third language model it is much lower. Furthermore, the 3-gram calculation in the first two language models is very low and much higher in the third language model. The higher the 3-gram calculation, the better the language model is. If the 1-gram is high, this indicates that the language model is not good.

The fact that language models I and II have a high 1-gram means that Arabic is a very rich language in vocabulary. This also indicates that the corpus in these two language models is small. The way improve this result is to supply more data.

These results demonstrate that, in order to achieve good recognition results, we need an adequate acoustic model, a good language model, and sufficient data.

Chapter 6

Experiments and Results

In this chapter the prerequisites to the experiments are discussed. Then the experiments are described. Finally, the results are presented.

6.1 Recognition Performance

In order to determine the recognition performance in our system, there are three different kinds of errors which can occur:

- substitution error: a spoken word is falsely recognized, i.e. substituted by another word.
- deletion error: a spoken word was not recognized at all, i.e deleted.
- insertion error: the recognition system recognizes a word which was not spoken at all.

Based on the above mentioned types of error we define the *word error rate* (WER) as:

$$\mathrm{WER} = 100 \cdot \frac{substitution+deletion+insertion}{number\ of\ words\ in\ reference\ sentence}$$

The following example illustrates the concepts of word error rate.

```
Ref: This great machine can recognize speech

Hyp: This grape machine     wreck nice peach
```

We have one deletion error (*can* was not recognized) and three substitution errors (*great* was recognized as *grape*, *recognize* was recognized as *wreck* and *speech* was recognized as *peach*) and one insertion error (*nice*). The word error rate (WER) is :

$$\text{WER} = \frac{3+1+1}{6} = 100 \cdot \frac{5}{6} = 83.33$$

The recognizer has a word error rate of 83.33% or a ***word accuracy*** WA (WA = 100−WER) of 16.67%.

6.2 Bootstrapping

As mentioned previously, this work is a part of the GlobalPhone Project, which has the goal of the development and realization of a multilingual speech recognition system.

In order to integrate the Arabic speech recognizer in the multilingual system, we used the same preprocessing of the speech signal as in the multilingual system. Table 6.1 shows us the Arabic phones and their mapping in the multilingual system.

The fact that we are developing a recognition system for a new language means that we do not have any parameter configuration. For this reason, we used the weights of the multilingual speech recognizer in order to initialize the system. The weight files in Janus are codebooks and distributions, which are used for Gaussian mixtures.

Using the initial weights, we calculated the first labels for the system. This has the advantage of storing the Viterbi alignment paths in files instead of recomputing them each time. Writing the first labels enables the system to be trained.

Arabic phone	Multilingual phone	Arabic phone	Multilingual phone
AR_B	M_b	AR_TH	M_ts
AR_KH	M_x	AR_DH	M_th
AR_TT	M_t	AR_J	M_j
AR_H7	M_h	AR_DD	M_D
AR_R	M_r	AR_RR	M_r
AR_Z	M_z	AR_SH	M_S
AR_SHSH	M_S	AR_SD	M_s
AR_SDSD	M_s	AR_S	M_s
AR_SS	M_s	AR_T	M_t
AR_DS	M_z	AR_D	M_d
AR_DD	M_d	AR_E3	M_a
AR_GH	M_g	AR_F	M_f
AR_Q	M_kh	AR_K	M_k
AR_L	M_l	AR_LL	M_l
AR_M	M_m	AR_MM	M_m
AR_N	M_n	AR_NN	M_n
AR_H	M_h	AR_W	M_v
AR_aW	M_y	AR_Y	M_j
AR_aY	M_i	AR_YY	M_j
AR_AE	M_Q	AR_A	M_a
AR_AA	M_a	AR_q	M_Q
AR_a	M_a	AR_u	M_u
AR_i	M_i	AR_+QK	M_+QK
AR_+hGH	M_+hGH		

Table 6.1: Initializing the Arabic phones

6.3 Context Independent System

In the context independent (CI) system, the modelling of the phone in different contexts is independent and does not change.

The system has been evaluated four times. The first evaluation was completed with a language model based on Arabic transcripts only, and the connected words with '_' were treated as if they were one word. In addition, homophones[1] were not considered. The second evaluation was also with the language model based on Arabic transcripts, but connected words with '_' were separated. Homophones were not considered in this evaluation either. The third evaluation similar to the second evaluation, but the homophones were considered. The fourth evaluation was the same as the third one, but the data was devocalized.

In the first system, AR0, we initialized the weights of the multilingual system (see 6.2). In the second system, AR1, we trained with 4 iteration after calculating an LDA and initializing the Gaussian distributions with the *k-means* algorithm. The idea behind this algorithm is simple. A set of example vectors is clustered into a few classes iteratively such that a distortion measure keeps being minimized. Every class has its mean vector. In a k-means iteration, every example vector is assigned to the class with the closest mean vector. After that every mean vector is replaced by the average of all vectors that have been assigned to it. In the third system, AR2, we used supervised adaptation ([redacted]) which is employed to improve the 'labeling' ([redacted]) of the training utterances. This technique is referred to as *label-boosting*.
Tables 6.2 and 6.3 represent the results for the CI system.

[1]Words that sound the same but are spelled differently and have different meanings like 'eight' and 'ate'.

System	Training action	WA	WA
		(1st evaluation)	(2nd evaluation)
AR0	Initialized by ML	10.2%	10.2%
AR1	CI System 4iter	40.0%	40.0%
AR2	CI System labelboost	40.6%	40.7%

Table 6.2: Recognition results for the first and second evaluation

System	Training action	WA	WA
		(3rd evaluation)	(4th evaluation)
AR0	Initialized by ML	10.2%	10.2%
AR1	CI System 4iter	40.0%	40.0%
AR2	CI System labelboost	41.6%	48.9%

Table 6.3: Recognition results for the third and fourth evaluation

6.4 Context Dependent System

In the context dependent system (CD) each phone is modelled according to the context using decision trees that are generated by questions about the phonetic context. These are represented in table 6.4 and 6.5.

The pronunciation of a phoneme can change depending on the neighboring phonemes, i.e. context. For this purpose a collecting of all contexts that can be modeled with the given task should be done. This means that the boundary characteristics must be examined.

The consideration of these aspects improves the modelling of the phoneme. We call context dependent models *Polyphones*. This, in turn, makes the recognition results better.

PHONES	@ SIL B TH KH DH TT J H7 DD R RR Z SH SHSH SD SDSD S SS T DS D E3 GH F Q K L LL M MM N NN H W aW Y aY YY A AA I U a u i q +QK +hGH
SILENCE	SIL
NOISES	+hGH +QK
NOISES	+hGH +QK
CONSONANT	B TH KH DH TT J H7 DD R RR Z SH SHSH SD SDSD S SS T DS D E3 GH F Q K L LL M MM N NN H W Y YY
BILABIAL	B M MM
LABIODENTAL	F
ALVEODENTAL	TH DH TT Z SD SDSD S SS T D N NN
ALVEOLAR	R L N SD S SDSD SS DD Z RR L LL
VELAR	KH K Y YY
UVULAR	GH Q
PHARYNGEAL	H7 E3 SH SHSH
GLOTTAL	H q
PLOSIVE	B T D Q K q
NASAL	M MM N NN
TRILL	R RR
FRICATIVE	TH KH DH J H7 Z SH SHSH S SS E3 GH F H Y
EMPHATIC-PLOS	TT DD
EMPHATIC-FRIC	SD SDSD
LATERAL	L LL
SEMICONS	W Y
STIMMHAFT	B DH DD R RR Z D E3 GH L LL M N NN W Y YY a AA i AA I A U u
VOICED	B DH DD R RR Z D E3 GH L LL M N NN W Y YY
UNVOICED	TH KH TT J H7 SHSH SD SDSD S SS T F Q K H

Table 6.4: Questions about phonetic context I

VOWEL	a A AA i I u U
ROUND	u U
UNROUND	a i I A AA
FRONT-VOW	a A AA i I aY
BACK-VOW	u U aW
CLOSE-VOW	i I u U
OPEN-VOW-UNR	a A AA
FRONT-VOW-UNR	i I
BACK-VOW-ROU	u U
LONG	A AA U I
SHORT	u i a
NASALIZED	aW aY u U a A AA i I

Table 6.5: Questions about phonetic context II

6.4.1 Polyphones

Polyphones are marked phonemes that represent a phone with information about its right and left contexts. A polyphone is still a single phone. It is just modeled depending on its context. Polyphones should not be confused with sequences of phones.

In our system we considered two kinds of polyphones: (1) triphones which model a phoneme where the context +1 to the right and −1 to the left are considered and (2) quintphones where the context +2 to the right and −2 to the left are considered. In the Janus toolkit (JRTK) there is a restriction that only one phone from the successor word and one from the previous word can be considered.

We will see later that the context modelling with triphones and quintphones raises the recognition performance of our system. This is because the modelling of the phones is more exact.

System	Training action	WA (1st evaluation)	WA (2nd evaluation)
AR3a	CD System 1.1cb700	55.8%	55.9%
AR3b	CD System 2.2cb1500	60.0%	60.3%
AR3c	AR3b on old phoneset	57.9%	58.1%
AR4a	AR3a with VTLN	57.3%	57.5%
AR4b	AR3b with VTLN	61.0%	61.3%

Table 6.6: Recognition results for the first and second evaluation

6.5 Experiments with Context Dependent System

The experiments with the context dependent system (CD system) represent in 4 different systems. In system AR3a we used triphones for the modelling. The number of triphones chosen after clustering is 700. The original number of triphones in the training data is 62000. In System AR3b we used quintphones for the modelling of the phones. Their total number after clustering is 1500. System AR3c used the same triphones, but with an old phoneset that was changed after completing corrections in the phonetics. The last two systems are AR4a and AR4b, which are the same as the systems AR3a and AR3b with following changes:

- In these systems the VLTN (see 3.2)technique was used.
- We computed a new LDA, initialized the codebooks of our context dependent system by extracting samples, ran kmeans, and did 4 iteration training.

The evaluation was done 4 times as in the CI systems (see 6.3)

Tables 6.6 and 6.7 represent the results for the CD system

System	Training action	WA (3rd evaluation)	WA (4th evaluation)
AR3a	CD System 1.1cb700	57.0%	64.1%
AR3b	CD System 2.2cb1500	61.6%	68.7%
AR3c	AR3b on old phoneset	59.2%	66.2%
AR4a	AR3a with VTLN	58.6%	66.0%
AR4b	AR3b with VTLN	62.6%	69.7%

Table 6.7: Recognition results for the third and fourth evaluation

6.6 Discussion of the Results

The best result that our Arabic recognizer achieved was in the 3rd evaluation: 62.6% for the vocalized data. This means that the word error rate was 37.4%. This result was performed in System AR4b, which uses quintphones for modelling the phones and the VLTN technique. System AR3a, which uses triphones with VLTN, achieved 4% less than AR3b. The reason for the difference is modelling. In system AR3b we modelled with quintphones, which is more exact than modelling with triphones.

The results for the systems AR3a and AR3b are lower because we did not use the VLTN technique.

Part of the reason for the relatively low recognition performance are the transcripts. In many cases, the same word was transcribed differently. Other cases include not recognizing the definite article (in arabic AL), which is added before a word and problems with short vowels and numbers. The short vowels at the end of a word can characterize the grammatical inflection of the word (marked by the suffixes: nominative /u/, accusative /u/ and dative /i/). These were in many cases not recognized. That means that the entire word until the phone before the vowel was recognized.

The following example shows us some of these cases:

```
REF: wqal ALNNAIB AL3RBYY 3zmy bsharte aenn MJALA alaemn lys
```

```
AEFDDLA mkan mkan ltttbyq ALMSA almsawate byn al3rb walyhwd
ALAESRAIYLYYYN
```

```
HYP: wqal NNAIB AL3RBY 3zmy bsharte aenn MJAL alaemn lys AEFDDL
mkan mkan ltttbyq MWSU  almsawate byn al3rb walyhwd ALAESRAIYLYYTE
```

Words written in capital letters were not recognized. The third word in the REF. ALNNAIB was recognized as NAIB. This means that AL (the definte article) was not recognized. The word MJALA was recognized as MJAL. The same problem occurred with AEFDDLA, which was recognized as AEFDDL.

These recognition problems in this example are due to the inflection. The reason for these problems was our language model, which included all test words. In order to start testing, we chose to include all test words at the beginning, and to build another language model later with sufficient data.

As soon as we received the LDC data, we started building a new language model. However, we were unable to finish due to a lack of time.

To annulate (from the example) this kind of errors we need a very good language model and a lot of training data. Also, consistent transcripts should be available. One more important point should be noted about the result in the 4th evaluation (69.7%). This result is higher than the result that was achieved in the 3rd evaluation because we devocalized our data. The reason for the better performance in this evaluation is that the inflection disappeared when we devocalized the data. When words such as 'kataba' and 'kitab' are devocalized, both will become 'ktb'. In this case it easier to recognize 'ktb' which might be originally 'kataba' or 'kitab'. In other words, there is less information and fewer choices of possible words for the speech recognizer to recognize. Therefore, there is also a lower chance of error.

Chapter 7

Conclusions

7.1 Conclusions

The Arabic speech recognizer developed in this work indicates that we still have to make a lot of changes for improvement. The main problem in this work was the data itself.

The highest result that has been achieved for vocalized data is (62.6%); for the same data devocalized the result is 69.7%. These results are good, if we take into consideration the following explanations:

- We have a small amount of speech data: only 447 minutes (4 hours and 27 minutes) for training.
- That fact that our corpus was not big, it was not possible to build a very good language model. With an improved language model it would be possible to reduce the OOV and have better results.

Our results are good considering that our language model was deficient. We expect that a new language model with sufficient data will have better results.

7.2 Future Work

In order to achieve better recognition performance for Arabic speech recognition, a number of measures can be taken:

- " *Vocalization* "solutions to be found.
  In this work we had a problem finding vocalized data. Such data is very important to achieve good recognition results because the vowels which are added during the vocalization are spoken and are very important for improving the acoustics. The development of tools or methods to do the vocalization automatically would be a crucial step to improve the efficiency and effectiveness of an Arabic speech recognition system.

- "*More Data*"
  To improve the recognition performance there is a need to collect more speech data for the acoustic models. This means not just improving the acoustic models, but also increasing speaker robustness by learning more about the "new speakers". In addition, more speech data will make it possible to build better qualitative language models

Bibliography

[1] Alex Waibel and Kai-Fu Lee. *Reading in Speech Recognition.* Morgan Kaufmann Publishers, Inc., San Mateo 1990

[2] Ernst Guenter Schukat-Talamazzini. *Automatische Spracherkennung.* Vieweg, Braunschweig/Wiesbaden 1995

[3] L.R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. In Alex Waibel and Kai-Fu Lee, *Reading in Speech Recognition.* Morgan Kaufmann Publishers, Inc., San Mateo, 1990.

[4] L.R. Rabiner and R.W. Schafer.*Digital Representations of Speech Signals.* In Alex Waibel and Kai-Fu. Lee, *Reading in Speech Recognition.* Morgan Kaufmann Publishers, Inc., San Mateo, 1990.

[5] L.R. Rabiner and R.W. Schafer.*Digital Processing of Speech Signals.* Prentice-Hall, Inc., New Jersey, 1978.

[6] F. Jelinek. Self-Organized Language Modeling for Speech Recognition. In Alex Waibel and Kai-Fu Lee, *Reading in Speech Recognition.* Morgan Kaufmann Publishers, Inc., San Mateo, 1990.

[7] Steve Young. *Large Vocabulary Continuous Speech Recognition: a Review.* Cambridge University Engineering Department UK, 1996.

[8] M. Finke, I. Rogina, M. Woszczyna, M. Westphal and T. Sloboda.*The JanusRTK Tutorial.* Interactive Systems Laboratories, Pittsburgh, PA, USA and Karlsruhe 1993-1997.

[9] O. Karboul. *Die hocharabische Sprache und Romanisierung ihrer Schrift.* Studienarbeit an der Fakulät für Informatik, University of Karlsruhe, May 1999.

[10] E.O. Brigham. *FFT: schnelle Fourier-Transformation.* R. Oldenburg Verlag, 1995.

[11] F. Jelinek, R.L. Mercer and S. Roukos. *Advances in Speech Signal Processing*, pages 651-700. M. Dekker Publishers, New York, NY, 1991.

[12] R. Kneser and H. Ney. Improved Backing-Off for M-Gram Language Modeling. In *Proceeding of the International Conference on Acoustics, Speech and Signal Processing Advances*, volume 1. IEEE Signal Processing Society, IEEE, New York, NY, May 1995.

[13] L.R. Rabiner and S.E. Levison. Isolated and Connected Word Recognition-Theory and Selected Applications. 1981. In Alex Waibel and Kai-Fu. Lee, *Reading in Speech Recognition.* Morgan Kaufmann Publishers, Inc., San Mateo, 1990.

[14] C.J. Leggetter and P.C. Woddland. Maximum Likelihood Linear Regression for Speaker Adaption of Continous Density Hidden Markov Models. In*Computer and Language*, 9(2): 171-185, April 1995.

[15] S.H. Al-Ani. *Arabic Phonology.* The Huge: Mouton, 1970.

[16] Yousif A. El-Imam. An Unrestricted Vocabulary Arabic Speech Synthesis System. In *IEEE Transaction on Acustics Speech And Signal Processing.* Vol. 37 No. 12, December 1989.

[17] Ossama Essa. Using Prosody in Automatic Segmentation of Speech. In *Proceeding of the ACM 36th Annual Southeast Conference.* April 1998.

[18] S.H. Al-Ani and J.Y. Shammas. *Phonology and Script of Literary Arabic.* Institute of Islamic Studies. McGill University. Motntreal, 1971.

[19] E. McCarus and R. Rammuny. *A Programmed Course in Modern Literary Arabic Phonology and Script.* University of Michigan. Department of Near Eastern Studies. Ann Arbor, 1974.

[20] John K. Ousterhout. *Tcl and the Tk Toolkit.* Addison Wiesley Publishing Company, Massachusetts 1995

[21] M. Finke, J. Fritsch, P. Geutner, K. Ries and A. Waibel. The JanusRTK Switchboard/Callhome 1997 Evaluation System. In *Proceedings of the LVCSR Hub5-e Workshop*, May 1997.

[22] A. Lavie, A. Waibel, L. Levin, M. Finke, D. Gates, M. Gavalda, T. Zeppenfeld and Z. Puming. Janus III: Speech-to-Speech Translation in Multiple Languages. In *Proccedings of the International Conference on Acoustics, Speech and Signal Processing*, voulme 1. IEEE, IEEE Comput. Soc. Press, Los Alamitos, CA, April 1997.

[23] A. Lavie, A. Waibel, L. Levin, D. Gates, M. Gavalda, T. Zeppenfeld, P. Zhan and O. Glickman. Translation of Conversational Speech With Janus II. In *Proccedings of the International Conference on Spoken Language Processing, ICSLP 96* , voulme 4. IEEE, IEEE, New York, NY, October 1996.

[24] M. Woszczyna and M. Finke. Minimizing Search Erros due to Delayed Bigrams in Real-Time Speech Recognition Systems. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, volume 1, IEEE, IEEE, New York, NY, May 1996.

[25] T. Schultz and A. Waibel. Development of Multiligual Acoustic Models im the GlobalPhone Project. In Proceedings of the First Workshop on Text, Speech, and Dialogue(TSD). Masaryk University, September 1998.

[26] T. Schultz and A. Waibel. Multiligual and Crossligual Speech Recognition. In *Proceedings of the DARPA Broadcast New Workshop*,IEEE, February 1998.

[27] T. Schultz and A. Waibel. Experiments towards a Multi-Language LVCSR Interface. In *Second International Conference on Multimodal Interfaces (ICMI 99)*, Januray 1999

[28] K. Ries, B. Suhm and P. Geutner. *Language Modeling in JANUS*. internal documentation. Interactive System Laboratories, Pittsburgh, PA, USA and Karlsruhe 1995.

[29] P. Zahn and A. Waibel. *Vocal Tract Length Normalization for Large Vocabulary Continuous Speech Recognition*. Technical Report CMU-CMT-97-150, Carnegie Mellon University, Center for Machine Translation, May 1997.

[30] M. Westphal. *Dimensionalitätsreduktion von Sprachsignalen mit statistischen und neuronalen Methoden*. Master's thesis, University of Karlsruhe, January 1994.

[31] T. Schultz. *Multilinguale Spracherkennung Kombination akustischer Modelle zur Portierung auf neue Sprachen*. Dissertation, Universität Karlsruhe (TH), Juli 2000.

[32] M. Woszczyna. *Fast Speaker Independent Large Vocabulary Continuous Speech Recognition*. Dissertation, Universität Karlsruhe (TH), February 1998.

[33] I. Rogina. *Parameterraumoptimierung für Diktiersysteme mit unbeschränktem Vokabular*. Dissertation, Universität Karlsruhe (TH), Juni 1997.

Appendix A

A.1 Tools

A.1.1 Dictionary implementation

```
#!/usr/local/bin/perl5
#-----------------------------------------------------------
# create Arabic dictionary (syllable- or word-based)
#-----------------------------------------------------------

############################################################
# RCS header:
#-----------------------------------------------------------
#
# $Id: dict.pl,v 1.3
#
############################################################

# Note1:
# We need to execute each V-KV substitution twice

# Note2:
# Creation of a pronounciation for a unit includes its
# neighbourhood i.e.  we append the last  syllable of
# the predecessor unit to the front and the first syllable
# of the successor unit to the end of our unitand create
# the pronounciation for that compound being able to
# resolve assimilations.

############################################################
# sub usage
#-----------------------------------------------------------
# show the user help on how to use the script
```

```
############################################################

sub usage {

    print <<END
------------------------------------------------------------
Usage: $0 [OPTION] <corpus_files ...>
create a dictionary for the words in corpus_files ...
We pay respect to phonological variations, too!

Options:

  --unit UNIT               UNIT is one of {syllable, word}
                            default: "word"
  --single-line LINE        calculate pronounciation of single
                             line, print it and exit
  --out FILE                store resulting dictionary in FILE
                            default: "dict"
  --comment-signs STRING    STRING contains chars that mark
                            comment lines that shall be ignored
                            default: ";#"
  --help                    display this help and exit
------------------------------------------------------------
END
}


############################################################
# sub my_read
#-----------------------------------------------------------
# enables reading from gzipped files
############################################################

sub my_read {

    local (*handle,$file) = @_ ;

    if (!-e $file && -e "$file.gz") {
        $file .= ".gz";
    }

    if ($file =~ /\.gz$/) {

        # gzipped file

        return(open(handle,"gunzip -c $file |"));
    } else {

        #standard open

        return(open(handle,$file));
    }
}


############################################################
# sub phones
#-----------------------------------------------------------
# create phonetic transcription of the actual unit
############################################################


sub phones {

    local($phones) = @_;

    # ***** consonants

  #$phones =~ s/([0-9][0-9])([0-9][0-9])/$1 $2/g;

    $phones =~ s/uWTh/uW Th/g;
    $phones =~ s/uW7/uW 7/g;
    $phones =~ s/uWKh/uW Kh/g;
    $phones =~ s/uWDh/uW Dh/g;
    $phones =~ s/uWSh/uW Sh/g;
    $phones =~ s/uWSd/uW Sd/g;
    $phones =~ s/uWDd/uW Dd/g;
    $phones =~ s/uWTt/uW Tt/g;
    $phones =~ s/uWD~/uW D~/g;
    $phones =~ s/uW3/uW 3/g;
    $phones =~ s/uWGh/uW Gh/g;

    $phones =~ s/aWTh/aW Th/g;
    $phones =~ s/aW7/aW 7/g;
    $phones =~ s/aWKh/aW Kh/g;
    $phones =~ s/aWDh/aW Dh/g;
    $phones =~ s/aWSh/aW Sh/g;
    $phones =~ s/aWSd/aW Sd/g;
    $phones =~ s/aWDd/aW Dd/g;
    $phones =~ s/aWTt/aW Tt/g;
    $phones =~ s/aWD~/aW D~/g;
```

```
$phones =~ s/aW3/aW 3/g;
$phones =~ s/aWGh/aW Gh/g;

$phones =~ s/iYTh/iY Th/g;
$phones =~ s/iY7/iY 7/g;
$phones =~ s/iYKh/iY Kh/g;
$phones =~ s/iYDh/iY Dh/g;
$phones =~ s/iYSh/iY Sh/g;
$phones =~ s/iYSd/iY Sd/g;
$phones =~ s/iYDd/iY Dd/g;
$phones =~ s/iYTt/iY Tt/g;
$phones =~ s/iYD~/iY D~/g;
$phones =~ s/iY3/iY 3/g;
$phones =~ s/iYGh/iY Gh/g;

$phones =~ s/aYTh/aY Th/g;
$phones =~ s/aY7/aY 7/g;
$phones =~ s/aYKh/aY Kh/g;
$phones =~ s/aYDh/aY Dh/g;
$phones =~ s/aYSh/aY Sh/g;
$phones =~ s/aYSd/aY Sd/g;
$phones =~ s/aYDd/aY Dd/g;
$phones =~ s/aYTt/aY Tt/g;
$phones =~ s/aYD~/aY D~/g;
$phones =~ s/aY3/aY 3/g;
$phones =~ s/aYGh/aY Gh/g;



$phones =~ s/aYa/a Y a/g;
$phones =~ s/aYi/a Y i/g;
$phones =~ s/aYu/u Y u/g;
$phones =~ s/iYa/i Y a/g;
$phones =~ s/iYu/i Y u/g;
$phones =~ s/iYi/i Y i/g;
$phones =~ s/uYu/u Y u/g;
$phones =~ s/uYi/u Y i/g;
$phones =~ s/iY/892/g;    #Long Vowel like ee in meet
$phones =~ s/aY/858/g;    #diphtong
$phones =~ s/aWa/a W a/g;
$phones =~ s/aWi/a W i/g;
$phones =~ s/aWu/a W u/g;

$phones =~ s/uWi/u W i/g;
$phones =~ s/uWu/u W u/g;
$phones =~ s/uWa/u W a/g;
```

```
$phones =~ s/iWa/i W a/g;
$phones =~ s/uW/882/g;     #long vowel like oo in boot
$phones =~ s/aW/958/g;     #diphtong


$phones =~ s/SS3/SS 3/g;
$phones =~ s/MMTe/MM Te/g;
$phones =~ s/SdSdKh/SdSd Kh/g;
$phones =~ s/LLDd/LL Dd/g;
$phones =~ s/LLTt/LL Tt/g;
$phones =~ s/TTAe/TT Ae/g;
$phones =~ s/BBan/BB an/g;
$phones =~ s/7Sh/7 Sh/g;
$phones =~ s/L-/L -/g;
$phones =~ s/BB/119/g;
$phones =~ s/B/ B /g;
$phones =~ s/un7/un 7/g;
$phones =~ s/inKh/in Kh/g;
$phones =~ s/TTSh/TT Sh/g;
$phones =~ s/WWTe/WW Te/g;
$phones =~ s/SSTe/SS Te/g;
$phones =~ s/RR3/RR 3/g;
$phones =~ s/A~Sd/A~ Sd/g;
$phones =~ s/A~Th/A~ Th/g;
$phones =~ s/LLDh/LL Dh/g;
$phones =~ s/LLGh/LL Gh/g;
$phones =~ s/LL3/LL 3/g;
$phones =~ s/L-ThTh/841/g;
$phones =~ s/ThThan/ThTh an/g;
$phones =~ s/ThThun/ThTh un/g;
$phones =~ s/ThThin/ThTh in/g;
$phones =~ s/ThTh/118/g;
$phones =~ s/Th/120 /g;
$phones =~ s/SdSdGh/SdSd Gh/g;
$phones =~ s/SdSd7/SdSd 7/g;
$phones =~ s/A~Kh/A~ Kh/g;
$phones =~ s/NNSh/NN Sh/g;
$phones =~ s/KhKhan/KhKh an/g;
$phones =~ s/KhKhun/KhKh un/g;
$phones =~ s/KhKhin/KhKh in/g;
$phones =~ s/KhKh/116/g;
$phones =~ s/-Kh/2001/g;
$phones =~ s/Kh/121 /g;
$phones =~ s/L-DhDh/846/g;
$phones =~ s/DhDhan/DhDh an/g;
$phones =~ s/DhDhun/DhDh un/g;
```

```
$phones =~ s/DhDhin/DhDh in/g;
$phones =~ s/DhDh/115/g; #double phones that should be considered
$phones =~ s/Dh/122 /g;
$phones =~ s/ZZTe/ZZ Te/g;
$phones =~ s/YYTe/YY Te/g;
$phones =~ s/DDTe/DD Te/g;
$phones =~ s/Te/ T /g;
$phones =~ s/L-TtTt/855/g;
$phones =~ s/TtTtan/TtTt an/g;
$phones =~ s/TtTtun/TtTt un/g;
$phones =~ s/TtTtin/TtTt in/g;
$phones =~ s/TtTt/114/g;
$phones =~ s/Tt/124 /g;
$phones =~ s/L-Sh/L Sh/g;
$phones =~ s/JJan/JJ an/g;
$phones =~ s/JJun/JJ un/g;
$phones =~ s/JJin/JJ in/g;
$phones =~ s/-JJ/158/g;
$phones =~ s/JJ/112/g;
$phones =~ s/J/ J /g;
$phones =~ s/-7/148/g;
$phones =~ s/7/146 /g;
$phones =~ s/WWDd/WW Dd/g;
$phones =~ s/L-DdDd/901/g;
$phones =~ s/Dd/127 /g;
$phones =~ s/L-RR/848/g;
$phones =~ s/RRan/RR an/g;
$phones =~ s/RRun/RR un/g;
$phones =~ s/RRin/RR in/g;
$phones =~ s/RR/908/g; #double phones that should be considered
$phones =~ s/R/ R /g;
$phones =~ s/L-ZZ/849/g;
$phones =~ s/ZZan/ZZ an/g;
$phones =~ s/ZZun/ZZ un/g;
$phones =~ s/ZZin/ZZ in/g;
$phones =~ s/ZZ/99/g;
$phones =~ s/Z/ Z /g;
$phones =~ s/L-ShSh/851/g;
$phones =~ s/ShShan/ShSh an/g;
$phones =~ s/ShShun/ShSh un/g;
$phones =~ s/ShShin/ShSh in/g;
$phones =~ s/ShSh/909/g; #double phones that should be considered
$phones =~ s/Sh/125 /g;
$phones =~ s/L-SdSd/852/g;
$phones =~ s/SdSdan/SdSd an/g;
$phones =~ s/SdSdun/SdSd un/g;
```

```
$phones =~ s/SdSdin/SdSd in/g;
$phones =~ s/SdSd/801/g; #double phones that should be considered
$phones =~ s/Sd/126 /g;
$phones =~ s/L-SS/850/g;
$phones =~ s/SSan/SS an/g;
$phones =~ s/SSun/SS un/g;
$phones =~ s/SSin/SS in/g;
$phones =~ s/SS/802/g; #double phones that should be considered
$phones =~ s/S/ S /g;

$phones =~ s/LLAe/LL Ae/g;
$phones =~ s/LLDD/LL DD/g;
$phones =~ s/L-TT/742/g;
$phones =~ s/TTan/TT an/g;
$phones =~ s/TTun/TT un/g;
$phones =~ s/TTin/TT in/g;
$phones =~ s/TT/804/g; #double phones that should be considered
$phones =~ s/T/ T /g;
$phones =~ s/L-DD/845/g;
$phones =~ s/DDan/DD an/g;
$phones =~ s/DDun/DD un/g;
$phones =~ s/DDin/DD in/g;
$phones =~ s/DD/805/g;
$phones =~ s/L-D~D~/854/g;
$phones =~ s/D~/128 /g;
$phones =~ s/D/ D /g;
$phones =~ s/-3/151/g;
$phones =~ s/3/129 /g;
$phones =~ s/-Gh/156/g;
$phones =~ s/Gh/422 /g;
$phones =~ s/FFan/FF an/g;
$phones =~ s/FFun/FF un/g;
$phones =~ s/FFin/FF in/g;
$phones =~ s/FF/98/g;
$phones =~ s/F/ F /g;
$phones =~ s/QQan/QQ an/g;
$phones =~ s/QQun/QQ un/g;
$phones =~ s/QQin/QQ in/g;
$phones =~ s/QQ/97/g;
$phones =~ s/Q/ Q /g;
$phones =~ s/KKan/KK an/g;
$phones =~ s/KKun/KK un/g;
$phones =~ s/KKin/KK in/g;
$phones =~ s/KK/96/g;
$phones =~ s/K/ K /g;
$phones =~ s/L-NN/859/g;
```

```
 $phones =~ s/NNan/NN an/g;
 $phones =~ s/NNun/NN un/g;
 $phones =~ s/NNin/NN in/g;
 $phones =~ s/L-LL/856/g;
 $phones =~ s/LLan/LL an/g;
 $phones =~ s/LLun/LL un/g;
 $phones =~ s/LLin/LL in/g;
 $phones =~ s/LL/906/g; #double phones that should be considered
 $phones =~ s/L/ L /g;
 $phones =~ s/MMan/MM an/g;
 $phones =~ s/MMun/MM un/g;
 $phones =~ s/MMin/MM in/g;
 $phones =~ s/MM/904/g; #double phones that should be considered
 $phones =~ s/NN/905/g; #double phones that should be considered
 $phones =~ s/M/ M /g;
 $phones =~ s/N/ N /g;
 $phones =~ s/H/ H /g;
 $phones =~ s/WWan/WW an/g;
 $phones =~ s/WWun/WW un/g;
 $phones =~ s/WWin/WW in/g;
 $phones =~ s/WW/005/g; #double phones that should be considered
 $phones =~ s/W/ W /g;
# $phones =~ s/aW/a W /g;
# $phones =~ s/W/ U /g;
 $phones =~ s/YYan/YY an/g;
 $phones =~ s/YYun/YY un/g;
 $phones =~ s/YYin/YY in/g;
 $phones =~ s/YY/902/g; #double phones that should be considered
 $phones =~ s/Y/ Y /g;

 $phones =~ s/<-/ < -/g;
 $phones =~ s/->/ - >/g;
 $phones =~ s/</ /g;
 $phones =~ s/>/ /g;
 $phones =~ s/-/ /g;
 # ***** Alif and Hamzeh

 $phones =~ s/-Ae/- Ae/g;
 $phones =~ s/Ae/ | /g;    # Hamzeh over or under Alif
 $phones =~ s/A~/142/g;    # Wasla
 $phones =~ s/A/ A /g;     # Alif without Hamza

 # the vowel after Hamzeh is Important
 # if the vowel is Fatha, then the Hamzeh
 #is over Alif and if the
 # vowel is Kasra then the Hamzeh is under Alif.
```

```
    $phones =~ s/E/ | /g;   # isolated Hamzeh
    $phones =~ s/I/ | /g;    # Hamzeh on a chair
    $phones =~ s/O/ | /g;    # Hamzeh on Waw
    $phones =~ s/U/ A /g;    # Alif Maksura ********************

        # ***** vocales


    $phones =~ s/an/a N/g;   # Tanwin Fatha
    $phones =~ s/un/u N/g;   # Tanwin Damma
    $phones =~ s/in/i N/g;   # Tanwin Kasra
    $phones =~ s/a/ a /g;    # Fatha
    $phones =~ s/u/ u /g;    # Damma
    $phones =~ s/i/ i /g;    # Kasra

    $phones =~ s/_//g;



    # ***** spaces ...

    $phones =~ s/(\A +| +\Z)//g;
    $phones =~ s/ {2,}/ /g;

    # **** Arabic phones (within ML-context)
    $phones =~ s/^/AR_/g;
    $phones =~ s/ / AR_/g;

    # *** syll delimiter
    $phones =~ s/ *AR_- */-/g;


    return($phones);
}

############################################################
# sub phonologicalChanges
#----------------------------------------------------------
# apply phonological changes that do occur at syllable boundaries
############################################################

sub phonologicalChanges {

    local ($word) = @_;
```

```
# ****** need a space in the beginning and in the end!
    $word =~ s/892/I/g;
    $word =~ s/858/aY/g;
    $word =~ s/882/U/g;#   long vowel like oo in boot
    $word =~ s/958/aW/g;
    $word =~ s/96/K/g;
    $word =~ s/97/Q/g;
    $word =~ s/98/F/g;
    $word =~ s/99/Z/g;
    $word =~ s/112/J/g;
    $word =~ s/114/TT/g;
    $word =~ s/115/DH/g;
    $word =~ s/116/KH/g;
    $word =~ s/118/TH/g;
    $word =~ s/119/B/g;
    $word =~ s/120/TH/g;
    $word =~ s/121/KH/g;
    $word =~ s/122/DH/g;
    $word =~ s/124/TT/g;
    $word =~ s/422/GH/g;
    $word =~ s/125/SH/g;
    $word =~ s/126/SD/g;
    $word =~ s/127/DD/g;
    $word =~ s/128/DS/g;
    $word =~ s/129/E3/g;
    $word =~ s/141/AE/g;
    $word =~ s/142/AA/g;
    $word =~ s/149/an /g;
    $word =~ s/144/un /g;
    $word =~ s/145/in /g;
    $word =~ s/146/H7/g;
    $word =~ s/148/H7/g;
    $word =~ s/151/E3/g;
    $word =~ s/156/GH/g;
    $word =~ s/158/J/g;
    $word =~ s/841/TH/g;
    $word =~ s/742/T/g;
    $word =~ s/845/D/g;
    $word =~ s/846/DH/g;
    $word =~ s/848/RR/g;
    $word =~ s/849/Z/g;
    $word =~ s/850/SS/g;
    $word =~ s/851/SHSH/g;
    $word =~ s/852/SDSD/g;
    $word =~ s/854/DS/g;
    $word =~ s/855/TT/g;
    $word =~ s/856/LL/g;
    $word =~ s/859/NN/g;
    $word =~ s/901/DD/g;
    $word =~ s/902/YY/g;
    $word =~ s/904/MM/g;
    $word =~ s/905/NN/g;
    $word =~ s/906/LL/g;
    $word =~ s/908/RR/g;
    $word =~ s/909/SHSH/g;
    $word =~ s/801/SDSD/g;
    $word =~ s/802/SS/g;
    $word =~ s/804/T/g;
    $word =~ s/805/D/g;
    $word =~ s/1110/TE/g;
    $word =~ s/005/W/g;
    $word =~ s/2001/KH/g;


#$word = " " . $word . " ";
#$word =~ s/ {2,}/ /g;

return($word);
}


############################################################
#
# MAIN
#
############################################################

### initialize variables and process command line arguments

# variables

$silence_phone  = "SIL";
$fragment       = "#fragment#";
$fragment_phone = "AR_+QK";
$noise          = "#noise#";
$noise_phone    = "AR_+hGH";



if ($#ARGV < 0) {
    &usage;
    exit -1;
}

# use this unit as default
$unit = "word";

# single-line mode is off by default
$single_line = 0;

# file to store resulting dictionary
$dict = "dict";

# default comment signs
$comment_line = "#;";

# parse command-line arguments

while ($#ARGV >= 0 && $ARGV[0] =~ /^--/) {

    if ($ARGV[0] eq "--help") {

&usage;
exit -1;

    } elsif ($ARGV[0] eq "--unit") {

        # set unit as specified

shift(@ARGV);
        $unit = shift(@ARGV);

if ($unit ne "word" && $unit ne "syllable") {

    &usage;
    exit -1;
}

    } elsif ($ARGV[0] eq "--out") {

        # set output file name as specified

shift(@ARGV);
        $dict = shift(@ARGV);

    } elsif ($ARGV[0] eq "--single-line") {
        # set line to create as specified

shift(@ARGV);
        $line = shift(@ARGV);
$single_line = 1;

    } elsif ($ARGV[0] eq "--comment-signs") {

        # ignore lines starting with one of these chars

shift(@ARGV);
        $comment_line = shift(@ARGV);

    } else {

# invalid option

&usage;
exit -1;
    }
}

$comment_line = "[".$comment_line."]";



# if user wants phonological transcription of one single line
if ($single_line) {

    $line =~ s/$noise//g; # no #noise# !! needed
    $line =~ s/(\A +| +\Z)//g;
    $line =~ s/ {2,}/ /g;


    # this is needed for merged units! we just pretend that merged units
    # behave like two in-word  neighbour syllables
    $line =~ s/_/-/g;

    if ($line eq "" || $line =~ /^$comment_line/) {
exit;
    }

    $line =~ s/ ([^\-])/$1/g;

#    $prono = &phones(&phonologicalChanges($line));
```

```
     $prono = &phonologicalChanges(&phones($line));

    $prono =~ s/\-/ /g;

    print "$prono\n";
    exit;
}


print stderr "===========================================\n";
print stderr " silence: $silence_phone\n";
print stderr "fragment: $fragment --> $fragment_phone\n";
print stderr "   noise: $noise --> $noise_phone\n";
print stderr "--------------------------------------------\n";
print stderr "         unit: $unit\n";
print stderr "  output file: $dict\n";
print stderr "comment-signs: $comment_line\n";
print stderr "===========================================\n";

@corpus_files      = @ARGV;

# check whether files exist

foreach $file (@corpus_files) {
    if (!-r $file) {
print "ERROR: corpus-file \"$file\" is not readable!\n";
exit -1;
    }
}

open(OUT, ">$dict") || die "ERROR: Failed to open dictionary
 file \"$dict\"\n";


foreach $file (@corpus_files) {

    my_read(IN, $file) || die "ERROR: Failed to open corpus-file
    \"$file\"\n";

    print stderr "Processing $file ... ";

    while ($line = <IN>) {

        #$cnt++;
        #print stderr "$cnt ";
chop($line);
```

```
$line =~ s/$noise//g; # no #noise# !! needed
$line =~ s/(\A +| +\Z)//g;
$line =~ s/ {2,}/ /g;

if ($line eq "" || $line =~ /^$comment_line/) {
    next;
}

if ($unit eq "syllable") { $line =~ s/\-/ -/g;}

# process line ...

@units = split(/\s+/, $line);

$pred = "";
$word = shift(@units);
while ($word ne "") {

            if ($word =~ /\(/) {
         $word =~ s/\(/</g;
 $word =~ s/\)/>/g;
     }

    $word_orig = $word;

    $succ_orig = shift(@units);
    $succ = $succ_orig;

    $pred =~ s/_/-/g;
    $pred =~ s/^.*\-([^\-]+)$/$1/g;

    $succ =~ s/_/-/g;
    $succ =~ s/^\-//g;
    $succ =~ s/^([^\-]+)\-.*$/$1/g;

    # concatenate neighbour syllables and actual unit
    $word = $pred."-".$word."-".$succ;
    $word =~ s/_/-/g;
    $word =~ s/(\A\-|\-\Z)//g;
    $word =~ s/\-{2,}/-/g;

    # ***** create phonetic transcription, but first process
                #phonological changes
#     $prono = &phones(&phonologicalChanges($word));
    $prono = &phonologicalChanges(&phones($word));
    # ***** remove multiple spaces
    $prono =~ s/(\A +| +\Z)//g;
    $prono =~ s/ {2,}/ /g;

    #if ($pred ne "") { $prono =~ s/^[^\-]*\-(.*)$/$1/g; }
    #if ($succ ne "") { $prono =~ s/^(.*)\-[^\-]*$/$1/g; }
    if ($pred ne "") { $prono =~ s/^[^\-]+\-//g; }
    if ($succ ne "") { $prono =~ s/\-[^\-]+$//g; }


    # insert janus tag stuff
    if ($unit ne "syllable") {

$prono =~ s/\-/ /g;

$prono =~ s/^([^ ]+)(.*) ([^ ]+)$/{$1 WB}$2 {$3 WB}/g;
$prono =~ s/^([^ ]+)$/{$1 WB}/g;
    }

    $table{"{$word_orig}\t{$prono}"} = "1";

    # unit for next iteration is the actual successor
    $word = $succ_orig;

    # predecessor for next iter. is last syllable actual word
    $pred = $word_orig;
}
    }

    print stderr "done\n";
    close(IN);
}

# create dictionary now

print OUT "{$noise}\t{$noise_phone}\n";
print OUT "{$fragment}\t{$fragment_phone}\n";
print OUT "{(}\t{$silence_phone}\n";
print OUT "{)}\t{$silence_phone}\n";
print OUT "{SIL}\t{$silence_phone}\n";
print OUT "{\$}\t{$silence_phone}\n";

$count      = 1;
$last_unit  = "";

foreach $key (sort keys %table) {
    @key = split("\t", $key);
    $unit = $key[0];

    if ($last_unit eq $unit) {

$count++;
$key =~ s/}\t/($count)}\t/;

} else {

    $count = 1;
}

print OUT "$key\n";

$last_unit = $unit;
}
print OUT "{<Geraeusch>}\t{{AR_+hGH WB}}\n";
close(OUT);
exit;
```

A.1.2 Creating pronunciation variants for words within brackets

```
#####################################################################
# sub Variant                                                       #
#-------------------------------------------------------------------#
# create phonetic transcription for words with brackets             #
#                                                                   #
#####################################################################

sub Variant {

    $line =~ s/$noise//g; # no #noise# !! needed
    $line =~ s/(\A +| +\Z)//g;
    $line =~ s/ {2,}/ /g;


    $firstbracket1pos = index($line, "(");
    $firstbracket2pos = index($line, ")" , $firstbracket1pos + 1);
    $secondbracket1pos = index($line, "(" , $firstbracket2pos + 1);
    $secondbracket2pos = index($line, ")" , $secondbracket1pos + 1);


#if the word contains one pair of bracket, the value of
#the secondbracket1pos is -1 (because second pair is not existing)
#the index function return the value -1

    if ($secondbracket1pos eq "-1") {


    $line =~ s/_/-/g;


# this is needed for merged units! we just pretend that merged
# units behave like two in-word neighbour syllables

    if ($line eq "" || $line =~ /^$comment_line/) {
    exit;
    }

    $line =~ s/ ([^\-])/$1/g;

    $prono1 =~ s/\-/ /g;
    $prono2 =~ s/\-/ /g;
    $line1 = $line;
    substr($line1, $firstbracket1pos, $firstbracket2pos + 1 -
           $firstbracket1pos) = "";
# $line1 =~ s/-//g;

        $LINE = $line;
    $LINE =~ s/\(//g;
    $LINE=~ s/\)//g;



    $line2 = $line;
    substr($line2, $firstbracket1pos, 1) = "";
    substr($line2, $firstbracket2pos -1 , 1) = "";


    $prono1 = &phones(&phonologicalChanges($line1));
    $prono1 = &phonologicalChanges(&phones($line1));
    $prono2 = &phones(&phonologicalChanges($line2));
    $prono2 = &phonologicalChanges(&phones($line2));
    $prono1 =~ s/\-/ /g;
    $prono2 =~ s/\-/ /g;
    $prono1 =~ s/^([^ ]+)(.*) ([^ ]+)$/{$1 WB}$2 {$3 WB}/g;
    $prono1 =~ s/^([^ ]+)$/{$1 WB}/g;
    $prono2 =~ s/^([^ ]+)(.*) ([^ ]+)$/{$1 WB}$2 {$3 WB}/g;
    $prono2 =~ s/^([^ ]+)$/{$1 WB}/g;

    print "$line1\n";
    print "$line2\n";
    print "{$LINE}$prono1\n";
    print "{$LINE(2)}$prono2\n";

#else means that the word contains 2 pairs of bracket, that means
#the value of the secondbracket1pos is not -1

exit;   } else {

 # this is needed for merged units! we just pretend that merged
 # units behave like two in-word neighbour syllables



   if ($line eq "" || $line =~ /^$comment_line/) {
   exit;
   }

   $line =~ s/ ([^\-])/$1/g;

   $prono1 =~ s/\-/ /g;
   $prono2 =~ s/\-/ /g;
   $prono3 =~ s/\-/ /g;
   $prono4 =~ s/\-/ /g;

   $newline =$line;
   $newline =~ s/\(//g;
   $newline =~ s/\)//g;


   $line1 = $newline;
   $prono1 = &phones(&phonologicalChanges($line1));
   $prono1 = &phonologicalChanges(&phones($line1));

   $line2 = $line;
   substr($line2, $secondbracket1pos, $secondbracket2pos -
          $secondbracket1pos + 1) = "";
   $line2 =~ s/\(//g;
   $line2 =~ s/\)//g;
 $prono2 = &phones(&phonologicalChanges($line2));
 $prono2 = &phonologicalChanges(&phones($line2));

 $line3 = $line;
 substr($line3, $firstbracket1pos , $firstbracket2pos -
        $firstdbracket1pos + 1) = "";

 $line3 =~ s/\(//g;
 $line3 =~ s/\)//g;
 $line3 =~ s/-//g;


 $prono3 = &phones(&phonologicalChanges($line3));
 $prono3 = &phonologicalChanges(&phones($line3));


 $line4 = $line;
 substr($line4, $secondbracket1pos, $secondbracket2pos -
        $secondbracket1pos + 1) = "";

 $newfirstbracket1pos = index($line4, "(");
 $newfirstbracket2pos = index($line4, ")" ,
 $newfirstbracket1pos + 1);
 substr($line4, $newfirstbracket1pos,
                $newfirstbracket2pos + 1) = "";
 $line4 =~ s/-//g;


 $prono4 = &phones(&phonologicalChanges($line4));
 $prono4 = &phonologicalChanges(&phones($line4));

 print "$line1\n";
 print "$line2\n";
 print "$line3\n";
 print "$line4\n";
 print "$prono1\n";
 print "$prono2\n";
 print "$prono3\n";
 print "$prono4\n";

#print "$firstbracket1pos\n";
#print "$firstbracket2pos\n";
#print "$secondbracket1pos\n";
#print "$secondbracket2pos\n";

 exit;
```

```
}
}
```

A.1.3 Counter for arabic numbers

```
# ================================================================
#  JANUS-SR   Janus Speech Recognition Toolkit
#             ----------------------------------------------------
#  Author  :  Jamal Abu Alwan
#  Module  :  counter for arabic numbers
#  Date    :  02.24.01
#
#  Remarks :
#
#=================================================================
if { $argc != 2 || [lindex $argv 1] == "-help"} {
    puts stderr "USAGE: $argv0 'in-file' 'out-file'"
    exit
}
set infilename [lindex $argv 0]
set outfilename [lindex $argv 1]

# ---------------------
#  read text file
# ---------------------


# ---------------------------------------
#  Einheiten fuer Tausende und Millionen
# ---------------------------------------

proc subst1m {no str} {
set ausspr ""
switch $no {
0 {set ausspr ""}
1 {set ausspr "MaLYuWN"}
2 {set ausspr "MaLYuWNaYN"}
3 {set ausspr "ThaLLAThaTeu MaLAYiYN"}
4 {set ausspr "AeaRBa3aTeu MaLAYiYN"}
5 {set ausspr "KhaMSaTeu MaLAYiYN"}
6 {set ausspr "SiTTaTeu MaLAYiYN"}
7 {set ausspr "SaB3aTeu MaLAYiYN"}
8 {set ausspr "ThaMaANiYuTeu MaLAYiYN"}
9 {set ausspr "TiS3aTeu MaLAYiYN"}
}
```

```
if {$no != "0"} {
    append str " "
}
append str $ausspr
return $str
}



# --------------------------
#  Tausende
# --------------------------

proc subst1000 {no str} {
switch $no {
0 {set ausspr ""}
1 {set ausspr "AeaLF"}
2 {set ausspr "AeaLFaYN"}
3 {set ausspr "ThaLLAThaTeu AAaLLAF"} #siehe ob romform richtig ist
4 {set ausspr "AeaRBa3aTeu AAaLLAF "}
5 {set ausspr "KhaMSaTeu AAaLLAF"}
6 {set ausspr "SiTTaTeu AAaLLAF"}
7 {set ausspr "SaB3aTeu AAaLLAF"}
8 {set ausspr "ThaMaANiYuTeu AAaLLAF"}
9 {set ausspr "TiS3aTeu AAaLLAF"}
}

append str $ausspr



return $str
}


# ----------------------
#  Hunderte
# ----------------------
proc subst100 {no no2 no3 str} {
 switch $no {
 0 {set ausspr ""}
 1 {if {($no2 == "0") && ($no3 == "0")} {
        set ausspr "MiIa"
 } else {  set ausspr "MiIa Wa"

   }
 }
2 {if {($no2 == "0") && ($no3 == "0")} {
       set ausspr "MiIaTAN"
   } else {
   set ausspr "MiIaTAN Wa"
   }
  }

3 {if {($no2 == "0") && ($no3 == "0")} {
       set ausspr "ThaLLAThuMiIa"
   } else {
   set ausspr "ThaLLAThuMiIa Wa"
   }
  }

4 {if {($no2 == "0") && ($no3 == "0")} {
       set ausspr "AeaRBa3uMiIa"
   } else {
   set ausspr "AeaRBa3uMiIa Wa"
   }
  }

5 {if {($no2 == "0") && ($no3 == "0")} {
       set ausspr "KhaMSuMiIa"
   } else {
   set ausspr "KhaMSuMiIa Wa"
   }
  }

6 {if {($no2 == "0") && ($no3 == "0")} {
       set ausspr "SiTTuMiIa"
   } else {
   set ausspr "SiTTuMiIa Wa"
   }
  }

7 {if {($no2 == "0") && ($no3 == "0")} {
       set ausspr "SaB3uMiIa"
   } else {
   set ausspr "SaB3uMiIa Wa"
   }
  }

8 {if {($no2 == "0") && ($no3 == "0")} {
       set ausspr "ThaMaANuMiIa"
   } else {
    setausspr "ThaMaANuMiIa Wa"
    }
   }


  9 {if {($no2 == "0") && ($no3 == "0")} {
        set ausspr "TiS3uMiIa"
    } else {
    set ausspr "TiS3uMiIa Wa"
    }
   }
 }
 if {[string length $ausspr]>0} {
  if {$no != "0"} {
     append str " "
  }
  append str $ausspr
 }
 return $str
}



# ------------------
#  Einheiten
# ------------------

proc subst1 {no str} {
 set ausspr ""
 switch $no {
 1 {set ausspr "WA7iD"}
 2 {set ausspr "AeiThNaYN"}
 3 {set ausspr "ThaLLATha"}
 4 {set ausspr "AeaRBa3a"}
 5 {set ausspr "KhaMSa"}
 6 {set ausspr "SiTTa"}
 7 {set ausspr "SaB3a"}
 8 {set ausspr "ThaMaANiYa"}
 9 {set ausspr "TiS3a"}
 }
 append str " "
 append str $ausspr
 return $str
}
```

```
# --------------------
#  Zehner
# --------------------
proc subst10 {no no2 str} {
 set ausspr ""
 if {$no == "1"} {
     switch $no2 {
     0 {set ausspr "3aShaRa"}
     1 {set ausspr "Aea7aDa 3aShaR"}
     2 {set ausspr "AeiThNaA 3aShaR"}
     3 {set ausspr "ThaLLAThaTea 3aShaR"}
     4 {set ausspr "AeaRBa3aTea 3aShaR"}
     5 {set ausspr "KhaMSaTea 3aShaR"}
     6 {set ausspr "SiTTaTea 3aShaR"}
     7 {set ausspr "SaB3aTea 3aShaR"}
     8 {set ausspr "ThaMaANiYaTea 3aShaR"}
     9 {set ausspr "TiS3aTea 3aShaR"}
    }

    if {[string length $ausspr] > 0} {
        append str " "
        append str $ausspr
     }

 } else {
  if {$no == "0"} {
     set ausspr ""
  } elseif {$no == "2"} {
     if {$no2 == "0"}  {
         set ausspr "3iShRiYN"
     } else {
         set ausspr "[subst1 $no2 {}] Wa 3iShRiYN"
     }
  } elseif {$no ==  "3"} {
     if {$no2 == "0"}  {
         set ausspr "ThaLAThiYN"
     } else {
         set ausspr "[subst1 $no2 {}] Wa ThaLAThiYN"
     }
  } elseif {$no == "4" } {
     if {$no2 == "0"}  {
         set ausspr "AeaRBa3iYN"
     } else {
         set ausspr "[subst1 $no2 {}] Wa AeaRBa3iYN"
    }

  } elseif {$no == "5" } {
     if {$no2 == "0"}  {
         set ausspr "KhaMSiYN"
     } else {
         set ausspr "[subst1 $no2 {}] Wa KhaMSiYN"
     }

  } elseif {$no == "6" } {
       if {$no2 == "0"}  {
         set ausspr "SiTTiYN"
     } else {
         set ausspr "[subst1 $no2 {}] Wa SiTTiYN"
     }

  } elseif {$no == "7" } {
       if {$no2 == "0"}  {
         set ausspr "SaB3iYN"
     } else {
         set ausspr "[subst1 $no2 {}] Wa SaB3iYN"
     }
  } elseif {$no == "8" } {
       if {$no2 == "0"}  {
         set ausspr "ThaMaANiYN"
     } else {
         set ausspr "[subst1 $no2 {}] Wa ThaMaANiYN"
     }
  } elseif {$no == "9" } {
      if {$no2 == "0"}  {
         set ausspr "TiS3iYN"
     } else {
         set ausspr "[subst1 $no2 {}] Wa TiS3iYN "
     }

} else { puts "WARNING - wrong value for no $no"
}

if {[string length $ausspr] > 0} {
   if {$no != "0"} {
      append str " "
   }
   append str $ausspr
}

if {($no != "0") && ($no2 != "0")} {
     append "Wa" str
  }
 }
 return $str
}



# -----------------------
#  Filter
# -----------------------

proc filter {in} {
 set temp $in
 regsub -all "  " $temp " " temp
 regsub -all "{ " $temp "{" temp
 regsub -all " }" $temp "}" temp

 return $temp
}


# -------------
#  Body
# -------------

set FPin [open $infilename r]
set FPout [open $outfilename w]

while {[gets $FPin firstword] >= 0} {

set aussprache1 ""



       if { [regexp {^[0-9]([0-9]?)([0-9]?)([0-9]?)([0-9]?)
       ([0-9]?)([0-9]?)([0-9]?)([0-9]?)} $firstword] } {
         puts $firstword
         set length [string first "." $firstword]
         if {$length < 0} {set length [string length $firstword]}
        # set aussprache1 ""
         switch $length {

      6 {
           set aussprache1 [subst100  [string index $firstword 0]
           [string index $firstword 1] [string index $firstword 2]
      $aussprache1]

     set aussprache1 [subst10  [string index $firstword 1]
     [string index $firstword 2] $aussprache1]
     if {([string index $firstword 1] == "0") &&
     ([string index $firstword 2] == "0")} {
    set aussprache1 ""
     } else {    set aussprache1 [subst1m [string index
            $firstword 2] $aussprache1]
     }
     append aussprache1 " AeaLF Wa"
     set aussprache1 [subst100  [string index $firstword 3]
     [string index $firstword 4] [string index $firstword 5]
      $aussprache1]
     set aussprache1 [subst10  [string index $firstword 4]
     [string index $firstword 5] $aussprache1]
 if {[string index $firstword 4] != "1"} {
      set aussprache1 [subst1 [string index $firstword 5]
      $aussprache1]
     }
     }

   5 {
     set aussprache1 [subst10  [string index $firstword 0]
     [string index $firstword 1] $aussprache1]
 if {([string index $firstword 0] == "1") && ([string index
     $firstword 1] == "0")  } {

     append aussprache1 " A~LaF Wa"
     } elseif {([string index $firstword 0] == "1") &&
     ([string index $firstword 1] == "0") &&
     ([string index $firstword 2] == "0") &&
     ([string index $firstword 3] == "0") &&
     ([string index $firstword 4] == "0")} {
        append aussprache1 " A~LaF"
 } else {
     append aussprache1 " AeaLF Wa"}
     set aussprache1 [subst100  [string index $firstword 2]
     [string index $firstword 3] [string index $firstword 4]
      $aussprache1]
     set aussprache1 [subst10  [string index $firstword 3]
     [string index $firstword 4] $aussprache1]
 if {[string index $firstword 3] == "0"} {
        set aussprache1 [subst1 [string index $firstword 4]
         $aussprache1]
     }
```

```
      }


         4 {
          set aussprache1 [subst1000 [string index $firstword 0]
           $aussprache1]

      if {([string index $firstword 1] == "0") && ([string index
            $firstword 2] == "0") &&
      ([string index $firstword 3] == "0") } {

          append aussprache1 " "

      } else {
         append aussprache1 " Wa"}

          set aussprache1 [subst100  [string index $firstword 1]
          [string index $firstword 2] [string index $firstword 3]
           $aussprache1]
          set aussprache1 [subst10  [string index $firstword 2]
          [string index $firstword 3] $aussprache1]
      if {[string index $firstword 2] == "0"} {
       set aussprache1 [subst1 [string index $firstword 3]
           $aussprache1]
          }


   #   if {([string index $firstword 1] == "0") &&
           ([string index $firstword 2] == "0") &&
           ([string index $firstword 3] == "0") }
          { set aussprache1 [subst1000 [string index $firstword 0]
                                  $aussprache1]
     # }




      }

        3 {
         set aussprache1 [subst100  [string index $firstword 0]
         [string index $firstword 1] [string index $firstword 2]
          $aussprache1]
         set aussprache1 [subst10  [string index $firstword 1]
         [string index $firstword 2] $aussprache1]
      if {[string index $firstword 1] == "0"} {
          set aussprache1 [subst1 [string index $firstword 2]
               $aussprache1]
           }
           }

         2 {
           set aussprache1 [subst10  [string index $firstword 0]
           [string index $firstword 1] $aussprache1]
      # if {[string index $firstword 0] != "1"} {
      #   set aussprache1 [subst1 [string index $firstword 1]
      #                   $aussprache1]
      # }
           }

         1 {
           set aussprache1 [subst1 [string index $firstword 0]
                           $aussprache1]

           }
         }
    }
    set aussprache2 ""
    #append aussprache2 "{"21
    append aussprache2 $aussprache1
    #append aussprache2 "}"

    set filtered_aussprache [filter $aussprache2]

  # puts $FPout "{$firstword} $filtered_aussprache"
  puts $FPout "{$firstword} [exec ./dictfuerklammerfreeNeu.pl
  --single-line "$filtered_aussprache"]"


}

close $FPin
close $FPout

exit
```

A.1.4 Procedure for float point numbers

```
#*****************************************************************

#Procedure  for float point numbers
```

```
#************************************************************
set aussprache1 ""
set separation "FASdiLa"
#if {[regexp {\.} $firstword]} { set separation "NuQTta" }
#if {[regexp {\,} $firstword]} { set separation "FASdiLa" }

if {[regexp {[\.,\,]} $firstword]} {

    set i [string first "\." $firstword]
    set vorkomma  [string range $firstword 0 [expr $i - 1]]
    set nachkomma [string range $firstword [expr $i + 1] end]
    set laenge [string length $vorkomma]

    switch $laenge {

    0 { puts "we don't handle this now" }
    1 { set aussprache1 [subst1 [string index $firstword 0] {}]
     append aussprache1 " $separation"
     append aussprache1 " [subst1 [ string index $firstword 2] {}]"
     append aussprache1 " [subst1 [ string index $firstword 3] {}]"
     append aussprache1 " [subst1 [ string index $firstword 4] {}]"
     append aussprache1 " [subst1 [ string index $firstword 5] {}]"
     append aussprache1 " [subst1 [ string index $firstword 6] {}]"
    }

    2 { set aussprache1 [subst10 [string index $firstword 0]
        [string index $firstword 1] {}]
     append aussprache1 " $separation"
     append aussprache1 " [subst1 [ string index $firstword
                              [expr $i + 1]] {}]"
     append aussprache1 " [subst1 [ string index $firstword
                              [expr $i + 2]] {}]"
     append aussprache1 " [subst1 [ string index $firstword
                              [expr $i + 3]] {}]"
     append aussprache1 " [subst1 [ string index $firstword
                              [expr $i + 4]] {}]"
}

   3 { set aussprache1 [subst100 [string index $firstword 0]
         [string index $firstword 1] [string index $firstword 2] {}]
if {[string index $firstword 1] == "0"} {
    append aussprache1 " [subst1 [ string index $firstword 2] {}]"
} else {
         append aussprache1 " [subst10 [string index $firstword 1]
          [string index $firstword 2] {}]"
}
      append aussprache1 " $separation"
      append aussprache1 " [subst1 [ string index $firstword
                                    [expr $i + 1]] {}]"
      append aussprache1 " [subst1 [ string index $firstword
                                    [expr $i + 2]] {}]"
      append aussprache1 " [subst1 [ string index $firstword
                                    [expr $i + 3]] {}]"
}
}
}


 set aussprache2 ""
   # append aussprache2 "{"
    append aussprache2 $aussprache1
   # append aussprache2 "}"

    set filtered_aussprache [filter $aussprache2]

#   puts $FPout "{$firstword} $filtered_aussprache"
    puts $FPout "{$firstword} [exec ./dictfuerklammerfreeNeu.pl
    --single-line "$filtered_aussprache"]"


}
```