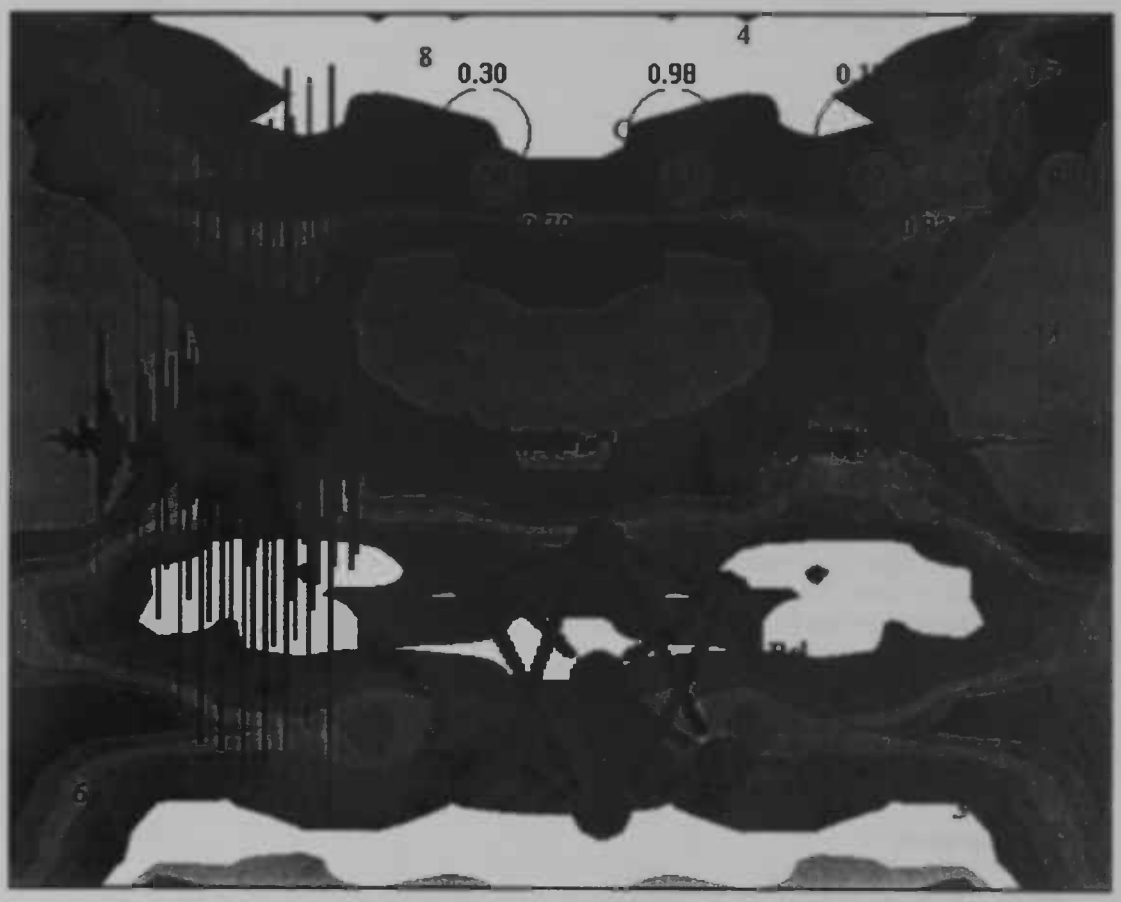
Department of Computing Science

Hidden Markov Models and Multilayer Perceptrons in a practical speech recognition application using the TMS320C50



Author
Martijn van Veelen

August 1997

Supervisors
prof.dr.ir L.Spaanenburg
dr.ir. J.A.G. Nijhuis

Co-operation
drs. M.H. ter Brugge
ing. R.Sytsma

Hidden Markov Models and Multilayer Perceptrons in a practical speech recognition application using the TMS320C50

Author
Martijn van Veelen

Student Technical Computing Science
Intelligent Systems & Signal Processing

Abstract

In this report the development of a speech recognising telephone is described. The speech recognition unit is designed to handle 13 words: the digits 0 to 9 and 3 commands. The TMS320C50, a Digital Signal Processor from Texas Instruments, is the core of the system. It is described briefly, together with the usefulness for speech recognition applications.

FFT-based pre-processing prepares for two techniques that are applied to perform the recognition task: the Multilayer Perceptron and the Hidden Markov Models Both methods are evaluated on basis of usefulness, development time, and performance. For the Hidden Markov Model classifier a tool is implemented. Generation of observation sequences is based on LVQ using a Kohonen Network.

The Multilayer Perceptron and the Hidden Markov Model Classifier takes nearly the same amount of development time. The Multilayer Perceptron performs best on the TMS320C50, it requires less memory and processor time. The Multilayer Perceptron appears to be the best method in this application, though the Hidden Markov Model classifier is realised with more ease.

Supervisors:
prof. dr. ir. L. Spaanenburg
dr. Ir. J.A.G. Nijhuis

Co-operators:
drs. M.H. Ter Brugge (Phd.)
ing. R. Sytsma (Electronics lab)

Department of Computing Science
University of Groningen (RuG)
Address: Blauwborgje 3
P.O. Box 800
9700 AV Groningen
The Netherlands
Tel.: 31 09 (0) 50 363 39 39
Fax.: 31 09 (0) 50 363 38 00

Email: brain@cs.rug.nl

Preface

In September 1996 I started to think about my Master Thesis. I had been a student at the University of Groningen (*RuG*) for exactly three years, and I had completed the majority of courses in two variants of the doctorate program. At that time I was working on a project concerning with the state observer for a motorised bicycle using Fuzzy Logic. This state observer will replace an expensive torque sensor as currently used for the motor control. The state observed is targeted for a 80C51 micro-controller with only 256 bytes RAM available. I found the combination of intelligent techniques, software engineering and hardware implementation quite interesting. I could have made this project the subject of my Master Thesis but contractual details forbade this.

At the end of 1996 I started to do some investigations on the detection and correction of time-related disturbances in batch-oriented production processes. Several Dutch companies are potentially involved in this project. which made this an even more interesting subject. I intend to start on the subject in March 1997. but. due to the many hours I spend on a private project I could not start on time.

From April 1996 till July 1997 I assisted in the graduate course "Technical AI". In this course students have to implement software for a speech recognising telephone using neural networks, subsequently ported to a DSP. The assignment was treated as if it were a commercial project. Being closely involved in designing and implementing the system inclusive a simulation environment for the students to work with during the course. I had not much time for working on a separate master thesis. In June J.A.G. Nijhuis and M. Ter Brugge proposed that I would take this speech recognising system for my Master Thesis. When I read some literature about speech recognition, in April, I kept running into Hidden Markov Models. So I was to make some study of Hidden Markov Models and compare them to Neural Networks as methods for speech recognition. This subject lies exactly in the my of interests: software engineering, intelligent techniques, hardware and practical applications.

Because most of the work on the software had been done, only the Hidden Markov Model and some pre-processing for the DSP had to be implemented. The report you see before you is the result of this work. I hope that this work clears the way for other students using Hidden Markov Models and Digital Signal Processors in their projects, as I found out that the both of them have much to offer.

I would not have been able to finish this thesis in time without the support of several people. I would like to thank the following people. Mark ter Brugge, for his work on the DSP and the pre-processor unit. Roelof Sytsma who has build and integrated all the hardware. My family, for offering a place to relax and looking after me at the stressful periods. My roommates for taking over my household tasks during this project. The students of Technical AI who performed the experiments with the Multilayer Perceptron and all the other people who took the time to help me write and improve this thesis.

Martijn van Veelen

Table of contents

Appendices

1. Introduction

1.1 The assignment, goal and objects

The main goal of this report is to find out which is the better speech recognition method in a commercial product. In this study a speech recognising telephone fulfills the function of the commercial product. As with most scientific research the goal itself is not the only object. One important object of this study is to learn how to apply Hidden Markov Models (HMM). Another is to get some experience with Digital Signal Processors (DSP), the TMS320C series in particular. These objects are met when the goal is achieved. We will present a summary of the problems one runs into applying HMM's, using DSP's and present solutions to these problems[1]. The problems discussed in this report concern both software and hardware implementation.

The techniques applied in this report are in first order very direct and not very fancy. Thus the performance of the system is not optimal, and we will *inter alia* suggest some alternatives that might increase performance. Instead of looking at this project as the realisation of a commercial project, which was the case in the course "Technical AI", it should be looked upon as a feasibility study, evaluating techniques in order to choose the better for a final implementation.

1.2 Requirements

Since we want to realise our system on hardware, using a DSP, it is important to identify what is required for speech recognition. When we decide to use the TMS320C50, ALEA IACTA EST, *See chapter 3*, we will not have an overview of the exact requirements: the chicken and egg problem. We had to make some estimations on the requirements, bases on an analysis of speech, discussed speech in the next chapter. In chapter 3 we will discuss the requirements concerning the hardware. Some aspects that have to be considered are: Computational power, RAM size, Sampling frequency and Accuracy.

1.3 Outline of the target systems

The speech recognising telephone can be used in two ways: (a) using a keypad as with ordinary telephones and (b) by voice. The telephone can store 10 phone numbers in RAM memory, and it stores the last dialled number. The phone recognises Dutch spoken keywords and the number 0 to 9. With the keywords the system can be told to store the spoken number in RAM memory, to recall a number from RAM memory, and to dial the spoken or restored number.

The speech recognition unit is not integrated in the telephone, but contained in a separate unit. The telephone can be connected to this unit, this unit itself is connected to the telephone line, connecting the system to an end-station of the PTT. A display is used to make operation easier, this display is integrated in the SRU. A more detailed description of the hardware is presented in chapter 3.

1.4 Criteria

The criteria for evaluation the two methods for Automatic Speech Recognition (ASR) concern not only the performance of the implemented systems. Although this is indeed a very important criterion, there are many other issues to be concerned, especially those issues that have an influence on the price of the system. After all practical applications must not only service the public, they are developed in order to make a profit too.

Criteria that cover those issues can be divided into two categories: (a) those concerning the components of the system, and (b) those concerning the development costs, which include wages too. The criteria of the first kind that will play a role in this report are expressed in demands on memory (RAM), processor speed (CPU clock

[1] Some, in favour of a more positive view, might want to read "problems" as "puzzles".

frequency), and accuracy (Word length). Those of the second kind are realisation time of components, complexity of the implementation and the complexity of the applied technique itself (how much does it take to understand). These criteria apply to the system as a whole and thus to all its components too, so they are an issue throughout this report. These criteria will be used to evaluate the two techniques in chapter 8 and 9.

1.5 About this report

The next six chapters describe the design and implementation of the system. The line followed in these chapters is the same as the steps in this project in which the ASR has been developed: Analysis of speech, Considering Hardware Issues, Simulator construction, Pre-processing, ASR design and implementation.

In chapter 2 we will look at speech itself. we discuss how can it be described, the important features and data acquisition. chapter 3 is about the hardware, we describe all the components of the system, their performance and how they were integrated. Also we describe how the hardware can be programmed and what parameters were used. In chapter 4 the we present simulation environment used to implement and test the pre-processor, the neural network and the Hidden Markov Model. In chapter 5 we discuss pre-processing: several techniques are presented and the pre-processor as we constructed it (the results from chapter 2 are used here). In chapter 6 and 7 we present the neural network and the Hidden Markov Model as they were implemented. chapters 5 to 7, show a design process which are encountered in many similar projects like the Fuzzy State Observer. First the technique itself is discussed. Then the technique is implemented and tested in Matlab™. When it seems to work properly, it is implemented in C. Finally it is ported to the target systems hardware, which in our case is a DSP. When a working version is realised using a DSP, it would be time to design dedicated hardware, a quite different line of work which is not considered in this report.

In chapter 8 we describe the experiments and evaluate the two techniques, according to the criteria in section 1.4. In chapter 9 we will discuss our findings, try to answer the question "Which is best?", look at some of the problems we encountered and compare our product with some commercial products. We conclude chapter 9 with some suggestions to improve the system.

2. Speech, analysis and sampling issues

In this chapter we will discuss some aspects of speech, and look at the most important features. As we shall see, the distinguishing features of speech are found in the frequency domain. This is not very surprising if one realises that speech, as every kind of making sounds, is based on vibrating air. Hearing or recording sound is just sensing the variations in air pressure.

2.1 Natural speech

Human speech is probably the most important way of communicating, mainly because almost every human being has the ability to speak one or more languages. It is for this reason that speech recognition is of such importance. One might think that communicating with machines by talking would be an improvement over pressing buttons. I do not share this opinion though, because it is in conflict with our desire for machines to operate quickly: pressing buttons is quicker than speaking. However there are many applications for speech recognition to name a few: replacing stenography, operation of computers and other machines by disabled people and automatic post-ordering over telephone lines.

2.2 Views on natural speech

There are many ways to look at speech. On a semantic level we consider the meaning of spoken language. One should realise that human beings can perform any kind of pattern recognition very well due to the fact that they can understand what is meant by that what they observe; machines do not have a notion of meaning, which, for them, is an enormous setback. In this report we will mostly consider the physical aspects of speech, and thus we speak of the speech signal instead of speech. In between these two ways of looking at speech is the grammar. Grammar is a formal aspect of speech and can be described in a mathematical way. Thus it can be used by a machine to perform the recognition task. Using symbolic information to perform (pattern) recognition tasks, is typically an AI approach. It is interesting to note that most speech recognition system, and especially the cheap commercial ones, do not use grammatical information. The use of techniques like neural networks for recognition tasks in combination with feature extraction, as described in this report, typically falls in to the field of CI, computational intelligence; there still seems to be quite a gap between CI and AI.

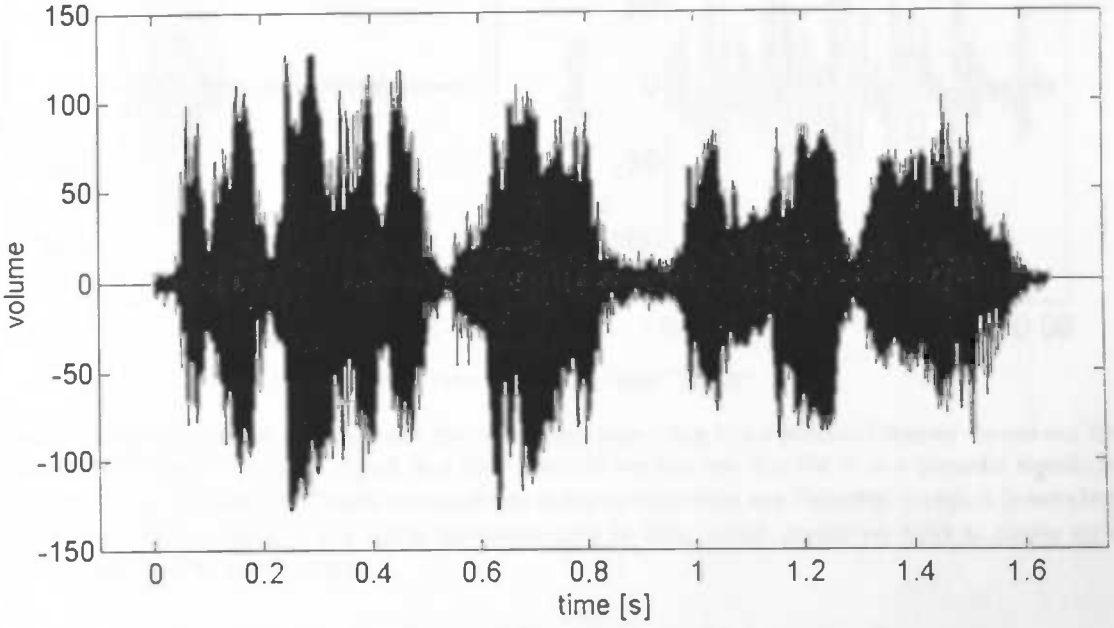


Figure 2-1: Continuous speech "Everybody cool this is a robbery"

We assume the speech signal to represent words. We consider isolated word recognition only, which means: we assume that we can separate words quite easily because their is enough silence in between. This assumption does not hold for spoken sentences. In Figure 2-1 you can see a spoken sentence, speech signals like this are referred to as continuous speech.

Words are composed of phonemes (sounds). A phoneme is considered to be the smallest unit in spoken language, sometimes it is called a phone, but I rather not use that word in this context. For example the word "pen" consists of three phonemes, namely "p", "eh" and "n". Most advanced ASR systems based on continuous speech are designed to recognise around 65 phonemes. In Appendix B. you can find a list of the most common phonemes in the English language, written phonetically. We will use phonetic language in this report to describe phonemes. Phonemes contain about 10 peak frequencies: these frequencies are called Formants. Thus a frequency analysis of the speech signal can give us significant information. After we introduce the words to use in our ASR application we will have a look at the speech signal in the frequency domain, and as expected we will find the unique distinguishing features of speech in that domain.

2.3 The "words" used in this system

The language $\mathcal{L}$ that controls the system contains 13 words or end symbols. Three of these words are control commands the other ten words are the digits '0' to '9'. The keywords are "M", "Bel" and "Opnieuw" which are the Dutch words for "M", "Dial" and "Repeat" This language $\mathcal{L}$ is generated by the grammar G, which is defined by a starting symbol, an alphabet of non-end symbols, an alphabet of end symbols, and a set of transitions. A question mark was introduced to denote an unrecognised command. The transitions, or production rules are numbered 1 - 8. G is defined as in Figure 2-2. The language $\mathcal{L}$(G) is a context-sensitive language, that can be used to improve the ASR system, we will discuss how this property can be used in chapter 9.

The keyword "M" is used for memory control. The word "Opnieuw" will be used to delete the last recognised digit, as can be seen in the grammar production rule 7.. An unrecognised word is denoted as a "?", which is not to be interpreted, shown by rule 9.

G= (S, N, T, P) where,
S = <command>
N = {<command> ; <restore-last> ; <load-number> ; <save-number> ; <dial-number> ; <number> ; digit> }
T = { "M" ; "Bel" ; "Opnieuw"; "0" - "9" ; "?" }
P = {

1.	<command>	→	<restore-last> | <load-number> | <save-number> |
2.	<dial-number>	→	<number> "Bel"
3.	<restore-last>	→	"Bel"
4.	<load-number>	→	"M" <digit>
5.	<save-number>	→	< number> "M" <digit>
6.	<number>	→	<digit> <number> | <digit>
7.	<digit> "Opnieuw"	→	ε[2]
8.	<digit>	→	"0" | "1" | • • • | "9"
9.	"?"	→	ε

}

Figure 2-2: Grammar generating the used language

Table 2-1 gives an overview of the keywords in phonetic symbols Some words can be pronounced in more than one way, this will require some extra states in the Hidden Markov Model we discuss in chapter 7.

[2] A symbol denoting "nothing" or "empty", this symbol is naturally included in all languages.

The minimal set of phonemes that can produce all the words used are: a , a: , b , d , e , ɛ , ə , f, x, i , i: , l , m , n , p , ɔ̃ , R, t , u , v , w , y , z. With the complete set of keywords defined we can start analysing these keywords considering them as continuous signals. The method of acquisition is discussed at the end of this chapter.

Word	Meaning	Phonetically
nul	0	nul
één	1	ein
twee	2	twei
drie	3	dRi:
vier	4	vi:R
vijf	5	vyf
zes	6	zes
zeven	7	zeivən / zəvən
acht	8	a:xt
negen	9	neixən
M	Memory	em
bel	Dial	bɛəl
Opnieuw	Delete	ɔ̃pni:w

Table 2-1 Keyword in phonetic symbols

2.4 The speech signal analysed

Since we are building an isolated word ASR system we will consider only separate words. In Figure 2-3a the word "acht" [a:x] is shown. The issue of separating words is covered in chapter 5.

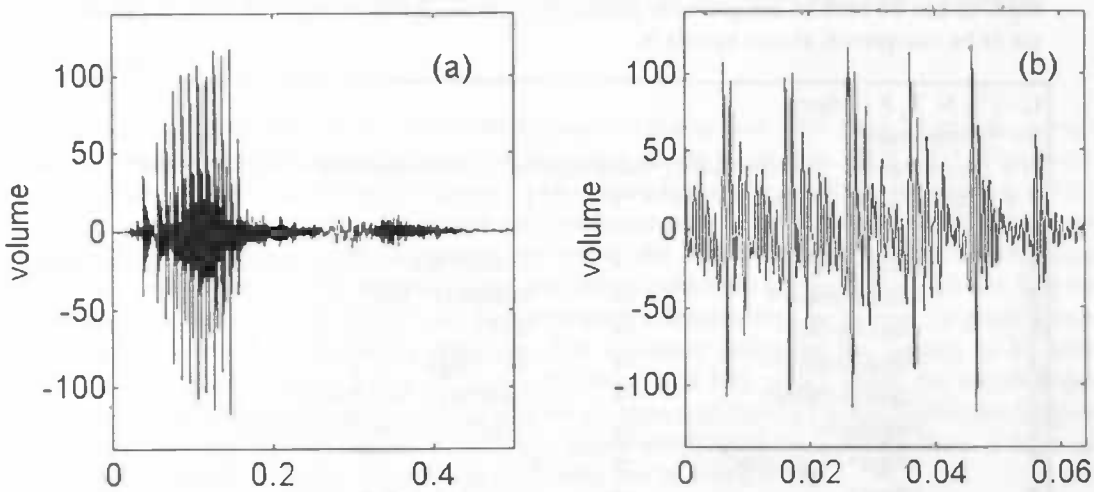


Figure 2-3: Time plot of (a) "acht" (b) "a"

In the figure we can see that the volume is not the same every time, thus it is a potential feature we can use for identification. When we look at the signal in a short interval we can see that the it is a periodic signal. In Figure 2-3b the "a" [a:] is shown. Clearly this phoneme contains more than one Formant, though it is not clear which Formants. The Formants in the signal obviously vary in time, which means we have to derive time dependent signals from the speech signal.

A transformation to the frequency domain enables us to study the Formants, in Figure 2-2a you can see the word "acht" in frequency-domain. One thing we can learn from this plot is that most information is contained in the lower part of the frequency spectrum.

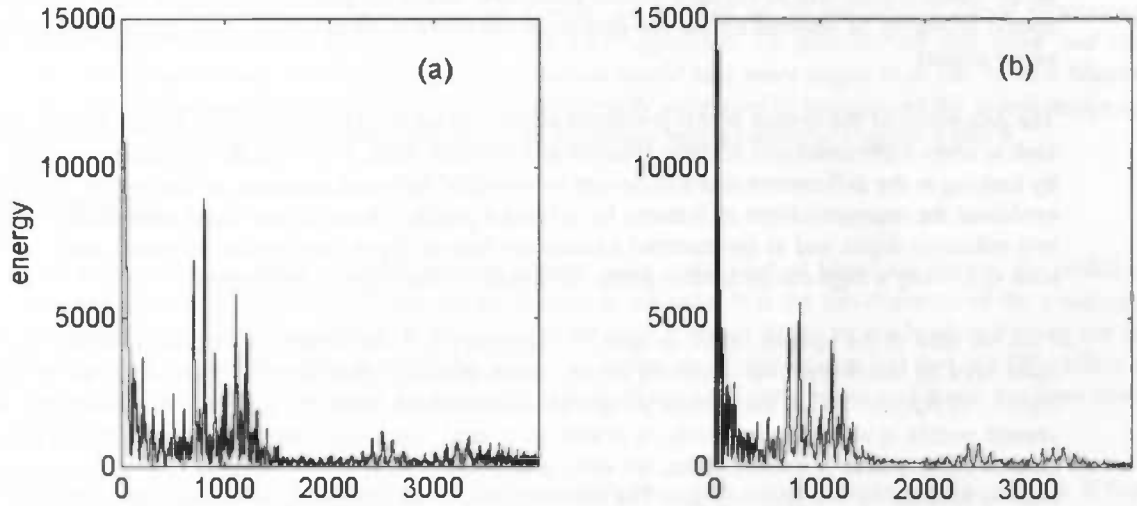


Figure 2-2: Frequency plot of (a) "Acht" (b) "a"

A look at the entire signal in the frequency domain does not give us a clear view on the Formants that produce the phonemes. Therefore we first cut the signal in 6 intervals, and transform them separately. The first interval is shown in Figure 2-2b. The interval contains only the spectrum of the "a" phoneme. In this plot we can see the Formants, the main 4 Formants of this phoneme are approximately : 100, 200, 700 and 1100 [Hz].

Since we want to look at the spectrum of the speech signal, but not loose all information of the signal in time, we need some way of representing the spectra as a function of time. Plotting them as above does not result in a very clear figure. One way of representing the spectra is the spectrogram. Instead of plotting a line we use colours to represent the energy in a certain frequency-band. A row of frequency-bands of the spectrum of the signal in a certain interval is called a time-slice. The spectrogram consists of a matrix containing time-slices. In **Error! Reference source not found.** a representation of the spectrograms of the words "acht" and "zeven". High energy is represented as the white and low energy as the black. You can see that the [❖] phoneme in "zeven" (time-bands 28-29) contains much energy in high frequencies whereas the [⊠] phoneme in "acht" contains no energy in high frequencies at all

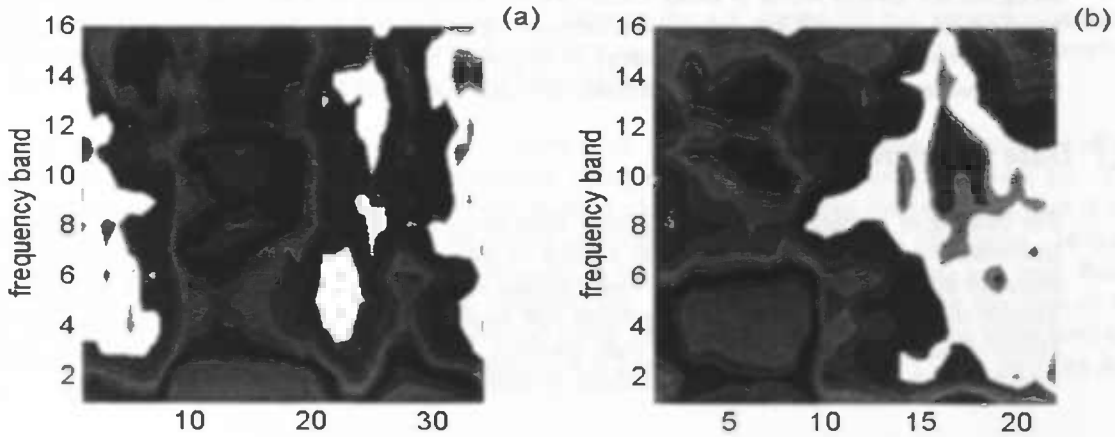


Figure 2-3 Spectrograms of (a) "zeven" and (b) "Acht"

2.5 Distinguishing features

Any classifier implemented on a computer that has to be acceptably fast, is highly dependent on the applied pre-processing technique. Therefore we have to analyse our data and derive methods to extract identifying features. Failing to do so will result in poor performance of the ASR system. Although we should not only look at what we humans can recognise but also at the fruit of information theory, we do use our "performance" on the classification task to evaluate feature extraction. When we aim to improve the ASR system our attention should primarily be focused on the pre-processor discussed in chapter 5, a more profound data analysis has a major impact.

The properties of the speech signal presented above can be used to identify the words. In this section we will look at some representations of these features and evaluate them. We evaluate the representations of the signal by looking at the differences and similarities in words of different speakers. In the course "Technical AI" we evaluated the representations of features by solving a puzzle where the ten digits were shown on a screen and two unknown digits had to be matched against the shown digits, the number of errors made and the time it took to identify a digit can be used to judge the quality of the represented feature.

First we look at the speech signal in time. In Appendix C.1. the described puzzle is shown. You will find it quite hard to match the two unknown digits, which indicates that something more has to be done with the signal. In Appendix C.2. the sample puzzle has to be solved, from the spectrum of the signals. Although the second puzzle is solved more easily, it is still quite hard. Both representations show a lot of data, thus some kind of compression is needed. What we want is something of both representations in a compressed form. Our need is fulfilled by the spectrogram. The spectrograms of the digits are shown in Appendix C.3., clearly this puzzle is not hard to solve. Since we can solve the last puzzle quite fast, we expect it contains sufficient information for a Artificial Neural Network (ANN) and a Hidden Markov Model to classify the words. In chapter 5, we will discuss how we can implement a feature detector that derives spectrograms from the speech signal. Beside the features discussed here, there are several other features commonly used in the field of speech recognition. Many pre-processing techniques are based on zero-crossing counts, auto-correlation functions (ACF) and the degree of voicing, all of these are a form of analysis in the frequency domain.

2.6 Sampling speech

Human speech contains practically no formants with frequencies higher than 4000 [Hz]. Moreover, as we can see in Figure 2-4a-b, most information is found in frequency bands lower than 2000 [Hz]. The sampling theorem of Shannon states that a sampling frequency of twice the highest frequency in the signal is sufficient to guarantee that no information is lost. Telephone companies use a sampling frequency of 8000 [Hz], which corresponds with the sampling theorem of Shannon; we will do the same. The quantisation resolution used in advanced speech recognition systems is about 16 bits per sample, telephone companies use 12 bits. The lower the resolution, the more distorted the signal gets. In the frequency domain this can be seen by growing energy levels in the higher part of the frequency spectrum. The minimum resolution for humans to be able to recognise the spoken words is about 5 bits. We use a resolution of 8 bits, which makes the sampled speech much easier to store in RAM, where each memory address contains a multiple of bytes. a low-pass frequency-band filter ensures that we have no energy in frequency bands above 4000 [Hz]. Surely there is more to be said about sampling speech; a good introduction in this matter can be found in [4].

2.7 Data acquisition

The training of a neural network with rather large amount of weights requires an enormous amount of data. Unfortunately we did not have such an amount of data available. Our speech database contains samples of about 40 different speakers. We have five different versions of each word from each of those 40 speakers. Though this might seem a lot, it is probably still insufficient to obtain a speaker independent ASR system. Moreover the database contains only male speakers. There is however a large database of speech samples available on the Internet: the TIMIT database. Unfortunately it contains only English samples.

3. Hardware, description of the target platform

We have been working with the DSP for almost 6 month, starting out with simple experiments on the DSP itself, then continuing by adding other hardware components to the system and finally testing. After the hardware was all put together we started writing assembly routines for the basic tasks of the hardware like: sampling, interrupt handling and display control. Finally we finished up by adding the signal processing tasks itself, which were written in C, except for the FFT algorithm. To describe all this work and the experience we gained during the development of the system would take more pages than can be in a Master Thesis. In this chapter we will describe only that what is strictly necessary to understand the grand design of the ASR system. Something more will be said about the hardware requirements in chapter 8 and 9.

3.1 Requirements

Whenever a prototype has to be constructed from scratch using some kind of hardware device, the problem arises that the exact hardware requirements are not known in advance. It is the development of the prototype itself that gives rise to certain requirements on the hardware. Still a choice has to be made when starting out to build the ASR system, and we chose the TMS320C50. Our choice was based on that fact that this DSP offers a large instruction set especially designed for signal processing like speech recognition and that it has been used for similar applications in the past.

The starter kit we used has on-board hardware for audio sampling, with variable sampling frequencies. It has 10,000 words of on-chip memory, which can easily be extended by mounting some extra memory chips on the board. The processor speed operates at 40 MHz a second and offers a hardware implemented multiplier which can perform a multiplication in one instruction cycle. This makes it a very good platform for signal processing, because signal processing involves real-time filtering, requiring millions of multiplications per second. One other important feature of this DSP kit is the layout of the hardware and the thorough documentation of the hardware layout which makes it relatively easy to add hardware components to the system. Altogether this DSP kit offers all the features necessary to build a signal processing system. However one drawback is the floating point limitation: the TMS320C50 is a fixed-point processor, and no on-board floating-point unit is present. A good alternative may be the TMS320C3x which is a family of floating-point DSP's with almost the same instruction set as the TMS330C5x family to which the chosen DSP belongs.

3.2 Overview of the hardware

Clearly the core of the system is the TMS320C50. However, several other components are required to make the system operational. One basic component is a telephone, which is used as a microphone and speaker, and its keypad which can be used for input by scanning for DTMF tones. The telephone is connected to the audio input and output of the DSP through the AIC chip. The AIC chip contains both a DA- and an AD-converter, and it can sample as well as generate sound with several different sampling frequencies. The AIC chip can be accessed directly from the DSP by memory-mapping, meaning the AIC digital I/O bus appears as memory addresses on the DSP. Sampling rates are realised by generating timer interrupts. Different timers for sampling and generating audio can be used as interrupt 5 and 6 are reserved for timer interrupts.

We used a 2 line display (16 character per line, memory of 80 characters) to give feedback to the user of the ASR system, The display has its own microprocessor which can communicate through an 8-bit bus. The display processor can be accessed through memory-mapped addresses too, though the communication is not direct. Several latches are needed to control the timing variations between the DSP and the display. In order to be able to detect whether the phone is off or on hook, an interrupt generating mechanism was built. Based on the change in the audio level of input, two interrupts are generated, depending on the direction of the change (interrupt 3 or 4). The drawback of this method is that a sufficiently large change in the audio level can be caused by yelling in the microphone. Some circuitry was constructed to access the PTT-line from the AIC chip, the access is controlled by a relays.

3.3 The TMS320C50 digital signal processor

This section describes the DSP very superficially, for a full description read [16]. The TMS320C50 architecture looks much like the architecture of an ordinary processor. It has several registers for arithmetic operations among which are two accumulators (ACC(B)), 8 auxiliary registers (AR0-7), a multiplier (TREG0) resp. multiplicand register (PREG) and several status register to account for overflow and sign modes. Calculations on auxiliary registers can be performed in parallel with arithmetic operations on the accumulators, though auxiliary registers may be used for other purposes than addressing only.

The memory is divided into pages; the current page is pointed to by a data page pointer (DP). The entire memory is to be configured manually, so that sufficient text and data memory is available for program, data and the stack. The memory configuration (memory map) is used when assembly code is linked, so that the loader knows which piece of code to put where. Six different addressing modes are supported, among which are an addressing mode that provides a way to move entire blocks in memory with only two instructions and circular buffer access, and instructions to access memory-mapped peripherals directly without changing the data page pointer.

The DSP uses pipe-lining for faster execution. Each instruction stays in the pipeline for four instruction cycles. First the instruction is fetched from memory, then the instruction is decoded, so that it can be interpreted by the micro-program[3], at the third instruction cycle the memory access to get data from RAM is performed, and finally the instruction is actually executed and stored in RAM if necessary . There at most four instructions in the pipeline, at each instruction cycle each instruction is passed one step through the pipeline. This way execution of a program can be done four times as fast, though some instructions do not benefit from pipe lining, especially interrupts cause extra overhead due to the pipeline. All instructions are coded in one or two 16-bit words. An instruction contains both an operand and an operator.

3.4 Interrupt handling

Interrupts are generated by a voltage change on one of the designated pins of the DSP. This voltage change causes a flag to be set in one of the registers of the DSP. The pending interrupt can be found in a status register. All interrupts have a certain priority, which tells the DSP in what order the interrupts should be dealt with. Once an interrupt is being processed it can not be interrupted by any other interrupt until it terminates. Only one of each interrupt can be registered, meaning a new interrupt is not registered when an interrupt of the same number is already pending. The code to process an interrupt can be found from the interrupt vector: an ordered list of two instructions per interrupt number. These two instructions can be of any kind though mostly they contain a call to a so-called interrupt service routine. The interrupt vector can be filled and changed dynamically (run-time) or it can be loaded in advance with a piece of code by defining the interrupt vector as a special section in the memory map. The address of the interrupt vector is stored in a status register. This register must be set before interrupts are enabled. Before the instructions in the interrupt vector are executed, some registers like the program counter (PC) are stored in shadow registers. This explains why only one interrupt can be processed at the time: there is only one set of shadow registers. Most interrupts can be masked, which means they are ignored.

As mentioned before we have to deal with 4 interrupts. A fifth interrupt (interrupt 2, non maskable) is used for communication with a PC using an RS-232 connection. There are two interrupts corresponding to the actions of picking up the phone (interupt 4) and putting it on hook again (interrupt 3). There is no way of knowing whether the phone is on hook or off hook. Interrupt 5 and 6 correspond to the timer interrupts for the AD and DA converter in the AIC chip, whatever is currently on the data bus connecting the AIC chip and the DSP is passed one way or the other depending on the interrupt. The timer interrupts are set in such a way that the sample frequency (AD conversion) is 8000 Hz. The DA conversion uses a different frequency. This timer is set to 10077 Hz (we use this frequency to generate DTMF tones).

[3] Micro-code is the hardware coded program in the core of the processor performing basic operations on the registers.

3.5 The telephone line

Since the telephone is not connected to the PTT line but to our hardware we needed some additional circuitry to access the PTT line. Because the PTT line contains high voltage power (110 V), we could not just plug it in to our DSP; instead a transformer is used to separate both circuits. In order to open the PTT line a relays has to be set. The relays can be set through a memory mapped address. While the relays is connected, the phone line is open and thus the audio channels of the PTT line and the AIC-chip are connected. Because the PTT company closes the connection automatically if no number has been dialled after 10 seconds, we have to wait until the number to dial is known before we make a connection. This has the effect of missing the acknowledge signal[4] when the phone is picked up. No circuitry was developed to receive phone calls, an additional interrupt may be sufficient to implement this feature.

3.6 The display

The display can be accessed by first setting up communication through a memory mapped register. There are three bits (lines) to be set before data can be passed: RS, R/W and E. The timing and order in which the bits are set is very strict. You can read all about it in the Hitachi HD44780U documentation available on the Internet. Depending on the setting of these bits there are three possibilities: an instruction or data may be passed to the display processor or data may be read from the display processor. The DSP has to wait for the display processor before data may be passed. To prevent short circuits, latches are used to pass the communication signals and a buffer is used to store data, which can go either way. Communication with the display is rather expensive due to the fact that the display processor takes microseconds to process commands and data whereas the DSP only take about 50 nanoseconds to execute instructions.

The memory-mapped address to write data to the display differs from the address at which data is received. The underlying hardware is different for the two operations and thus two physically different address were required. Because the communication with the display requires such strict timing, one of the first things we did was writing three assemble routines to perform the communication task for writing an instruction, writing data and reading data.

3.7 Outline of the hardware

Now that all the hardware components have been presented we show how it is all connected. A representation of the hardware is shown in Figure 3-1 a full description of the Circuitry is shown in Appendix D.

[4] Dialling tone

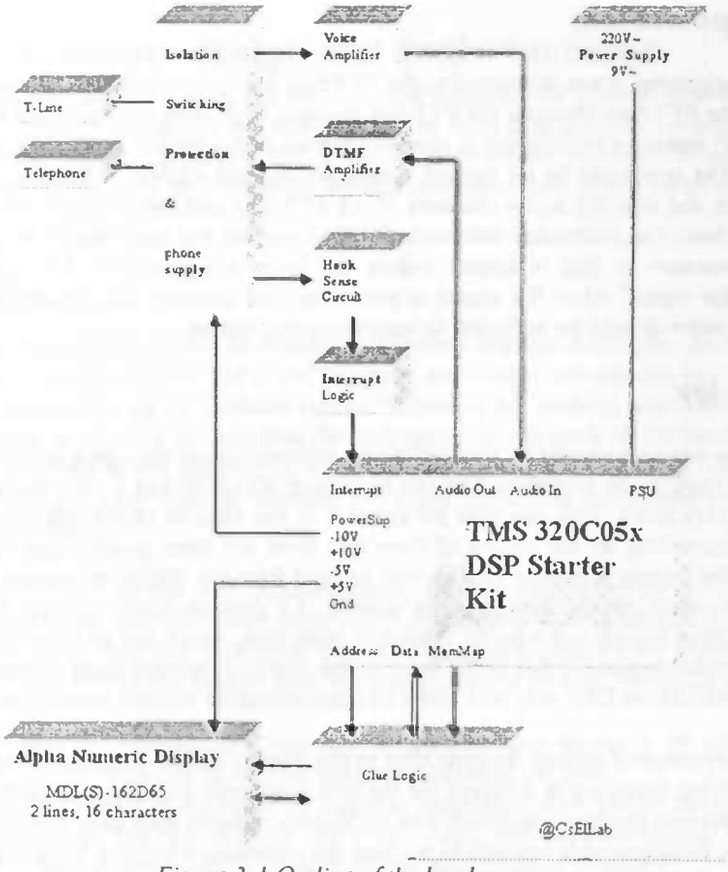


Figure 3-1 Outline of the hardware

3.8 Programming

The process of generating hexadecimal code for the DSP consists of several steps. Assembly code produces object code. For all modules, that contain data to be available in a C module a C-style header file is provided. All variables available in C are preceded with an underscore in assembly code. C-code is compiled to assembly code after which it is assembled and again object code is produced. After all the object code is generated, it is linked. The linker uses the memory configuration defined in the memory map. This map defines the pages in which the memory should be partitioned. Each page contains one or more sections; each piece of code, block of variables and initial data goes into a section. We used three different pages: the default pages for text (page 0) and data (page 1) and a third page for the interrupt vector (page 2). The standard sections for program code, global stack, global variables and constants were defined, and for certain pieces of code and data, required to be in one page to increase performance, some additional sections were defined.

Our goal was to write assembly code dealing with low-level communication, interrupts and parts that should be executed very fast in assemble and C code for the most complex parts and parts that should not be interrupted by timer interrupt. The assembly code generated by a C-compiler is lots more expensive measured in time and memory than hand-written assembly code. It takes much knowledge about C, assembly and compilers to write efficient C-code. Optimising C-code automatically, with the TMS320C5x optimizer, does barely any good. Moreover one has to be extremely careful with data shared among C and assembly code when using an optimizer. Saving space and time in C can be done effectively by choosing loop variables and pointers very carefully. However C-code will never use the extremely large instruction[5] set to its full extent.

[5] Yes, there is much to be said in favour of the philosophy behind the RISC processor

The integration of C code and assembly objects requires some overhead on the following points:

1. The stack pointer (**AR1**) and the frame pointer (**AR2**) used in assembly code generated by the C compiler, have to be stored and restored by the assembly routines.

2. Calls made to assembly routines from C routines using the **asm** statement have to store and restore all used registers on the software stack.

3. The TMS320C50 maintain a 8-level stack of the return addresses of the callers. There is no notification when this stack is overloaded. To prevent overload of the stack, the top of the hardware stack should be stored and restored by the called assembly routine.

3.9 Limitations, an overview

There are several limitations of the hardware to be taken into account. First of all, the cycle time of the main processor, the DSP, is 50 ns, which allows a maximum of 20 mips[6]. This limitation determines mainly how many instructions can be executed per sample for real-time pre-processing. The pre-processing for one sample must be done before the next timer interrupt by the sampling process occurs otherwise samples are lost. The time necessary to perform the recognition task itself only determines the minimum required interval of silence between spoken words, although this time should be small enough for the ASR to be of practical use is not limited by real-time requirements.

A second limitation, which is however easier to solve if problems arise, is the amount of available RAM. The starter kit we used had approximately 10 K words RAM available, but mounting extra RAM chips onto the board is quite easy and very well described in the documentation of the TMS320C5x Starter kit. The 10K words can be addressed separately though byte access is not possible. Thus using 8-bit sampling accuracy takes some extra code for storage.

During the development of the system one is constantly solving a puzzle to fit the code and the data into memory and make the code fast enough. One is constantly looking for balance between efficient data storage and required code. The two limitations determine the possible numerical accuracy and eventually the accuracy and speed of the ASR system as a whole. Therefore accuracy and efficiency are the main criteria for the evaluation of all components in the system.

[6] Million Instructions Per Second

4. Simulation environment, design and implementation

In this chapter we describe the design, implementation and usage of the simulation environment that was written to run an a PC to develop and test the ASR system The main goal of this simulation environment was not to test the system for real-time problems but to test the functionality of all the components, which are described in the next three chapters. It is however a functional equivalent of the hardware described in the previous chapter.

4.1 Why simulating ?

We put rather a large amount of time into developing a Windows-oriented simulation environment. Although there is a C-compiler available for the TMS320C50, there are several reasons not to implement the ASR directly on the target system. During the graduate course "Technical AI" students had to design and implement parts of the system. Because we were not sure that the ASR would finally work on the hardware and there was only one DSP board available it was decided to implement a simulation environment. These were the main reasons; however there are more general reasons for using simulation environments like the one we implemented. To name a few.

1. We could start out with a design and implementation without knowledge about the DSP's assembly language and its limitations. With the rather large instruction set, this is a big advantage.
2. The hardware thus barely has any debugging possibilities, except probing the electric circuit. With a simulation running on a PC we can benefit from the possibilities of IO and debugging using the Microsoft developer studio.
3. We were able to develop software and hardware in parallel, thus saving a lot of time, and putting everybody to work instead of having to wait for each other to finish the work.
4. We could ignore the limitations of the hardware to get an operational ASR system running on a PC.
5. Simulation allows for quick and easy to implement experiments, with the experimental results in an easy to use and clear form. These results could be interpreted very quickly using Matlab™ .

Thanks to Microsoft Developer studio, we were able to implement an functional simulation environment with a nice looking user interface quite quickly. It makes one wonder why students are not educated more thoroughly in the use of RAD tools like this.

4.2 Requirements on the simulation environment

The reasons to implement a simulation environment induce some requirements. To be able to see what is happening during the execution of the simulation we need some sort of output screen to print debugging messages and other information. The simulation environment has to be able to support any kind of pre-processing and recognition method; thus the code has to be easy to extend and its structure must be very clear and easy to understand. Furthermore the simulation environment must support some standard data format for audio to perform tests. Also, naturally, the simulation environment must be a very close approximation of the hardware described in the previous chapter; especially the interrupts have to act the same.
We want the simulation environment to make the hardware, therefore it should contain a user interface which offers ways of interaction like a telephone and a display.
Last but not least we want the code written to simulate an ASR system in this simulation environment to be portable to the target system. One has to be able to compile the written code with the TMS320C5x C-compiler.

4.3 Outline of the simulation environment

Although the Windows-oriented user interface part, which was written using the MDS Application Wizard with the Microsoft Foundation Classed, is a very interesting part of the simulation environment. It is not of any importance for the ASR system so we will not describe its design and implementation.

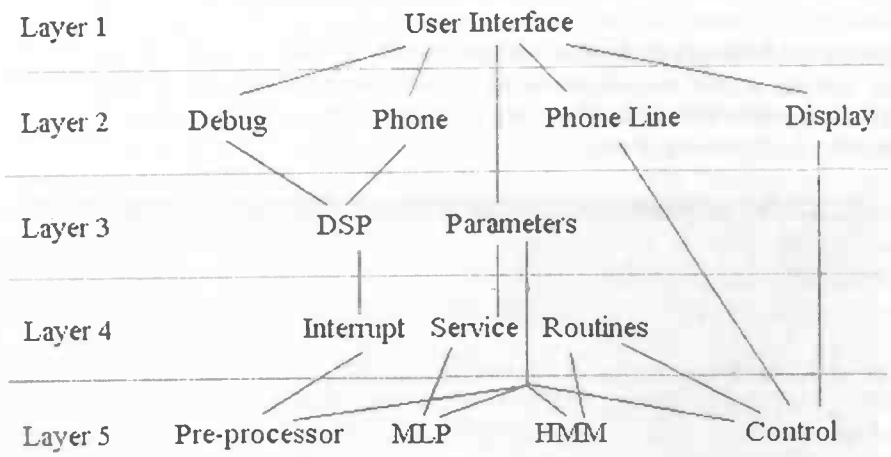


Figure 4-1 Structure of the simulation environment

The global structure of the simulation environment is shown in figure 4-1. The simulation environment is split up into five layers. The five layers are defined as follows:

Layer 1:	The Windows part of the Simulation Environment, visual realisation
Layer 2:	Peripherals, passing data from User interface to DSP simulation
Layer 3:	The hardware system simulator and its parameters
Layer 4:	Interrupt service routines, similar to that of the hardware system
Layer 5:	Supporting code

The entire user interface is the one and only component in layer 1. It is described in the next section. The second layer contains all the peripherals: telephone, the display and the Phone line. The debugging information also passes through this layer. The Third layer contains the core of the DSP simulator; it simulates the interrupts described in the previous chapter and passes data between the peripherals and layer 4 in both directions, also the parameters associated with the peripherals reside in this layer. In the fourth layer the interrupt service routines are implemented. In the simulation environment these are just stubs which are to be implemented to complete the simulation. The fifth layer contains nothing initially: it is reserved for modules and routines associated with pre-processing, recognition and general control. In the fourth and fifth layer the pre-processor and the core recognition, a Multilayer Perceptron or a Hidden Markov Model Classifier, described in the next three chapters are to be implemented.

4.4 The user interface

The user interface shown in Figure 4-2 contains three main areas. The area on the right called "Simulation" shows the debugging and other information that can be extended in the Interrupt Service Routines Part. At the bottom of this area a button with the caption "Running" is attached; this button starts the simulation itself, it corresponds to supplying power to the hardware or turning the ASR system on. On the left a so-called progress bar is shown; this bar gives an indication about the number of samples, from a wave file (*.wav), that are send to the DSP simulator.

The upper left area shows information about the loaded wave file, namely the sample frequency, accuracy and the size of the file. On top of the screen the loaded file is shown behind the word "Filename". Wave files can be loaded by selecting "Open" in the "File" menu or by clicking the "Map" icon. With the "Play" button the loaded wave file samples are sent so the DSP simulator. This corresponds to speaking through the phone.
In the lower left area the Phone is shown with a 16 character, 2-line display. A wave file containing DTMF samples is attached to each of the buttons on the keypad. These samples are passed to the DSP simulator when a button is pressed. At the bottom left of this area two buttons are attached. The upper one with the caption "Sound" can be used to turn sound on and off. When the sound is turned off, the wave files are not played

back when the play button is pressed nor are the wave files attached to the keypad. However DTMF tones generated by the Control routines, yet to be implemented, are played back. Turning sound on/off can be done too by using the button in the toolbar with the blue-red speaker on it. At the bottom, the lower button with the caption "Active" is used to pick up the phone and hang up. This can be achieved too by using the "Phone" button in the toolbar. Before the Phone is picked up power must be supplied. So the "Active" button should be pressed after the "Running" button.

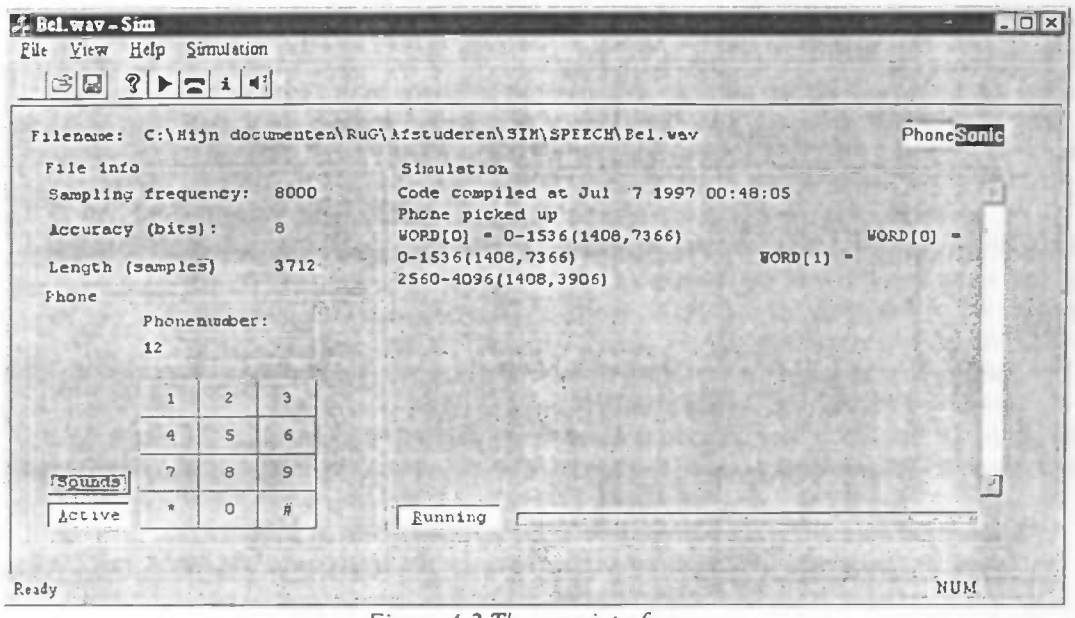


Figure 4-2 The user interface

All these actions cause interrupts that are simulated by the DSP simulator discussed in the next section. All these interrupts, which we call messages in the simulation environment, can be send manually too. In the menu "Simulation", there are four options: "Start" and "Stop" act like the "Running" button, "Settings" and "Messages". With the "Settings" options the simulation parameters will be shown. After selecting the "Messages" option the dialogbox in Figure 4-3 appears. One can select the message to send; this is the interrupt to simulate. The block interrupt is generated after a fixed number of samples have been sent. The number is a simulation parameter. Furthermore there are three messages that can be used for debugging the control part of the ASR system; these all have prefix "Test". When sending samples, the number of samples and a noise percentage can be modified. This way the noise sensitivity of the ASR system can be tested.

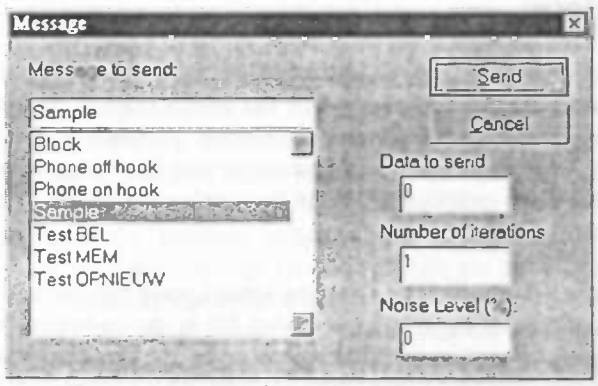


Figure 4-3 Dialogbox for sending messages

4.5 Implementation, and simulation parameters

The DSP simulator is implemented as a handler that processes messages sent by the User Interface: it can tell the User Interface what to do by returning messages to it. The sample data, debug text and display text are variables that can be accessed by the user interface. Sample data is passed to the Interrupt Service routine as a value parameter. The DSP simulator is initialised when the "Running" button is pressed. At every message received by the Message Handler it calls an Interrupt Service Routine. The Interrupt Service Routines are: **ProcessSample(sample)**, **ProcessBlock()**. **ProcessFirstSemiBlock()**, **PickUpPhone()** and **OffHookDetect()**. The routine **ProcessFirstSemiBlock()** was necessary to make the pre-processor work properly. Three testing routines were added later on to test control: **TestBEL()**, **TestMEM()** and **TestOPNIEUW()**. These correspond to the spoken commands dial, memory and delete, *see chapter 2*.

The display can be addressed like the real thing. There are three routines available for the display object: **DisplayWriteInstr(instruction)**, **DisplayWriteData(data)**, **DisplayReadData(data)**. The parameters correspond to the 8 bit instruction format of the Hitachi HD44780U. These three routine have the same interface as the three assembly written routines to access the display processor from the DSP on hardware.
The routines operate on two strings, one for each line on the display. These two strings are accessible by the User Interface. We will not go into the implementation details for the display because is not of importance for the Recognition Task of the system.

The phone line and the relays to access this phone line in hardware are simulated by three routines. **OpenAudioOut()** set the relay so that the connection to the phone line is made; **CloseAudioOut()** closes this connection. The routine **AudioOut(sample)** sends an audio sample to the phone line. This last routine is redundant on the hardware, because the audio input and output are on the same single circuit, as is common in telephone applications. The incoming speech and outgoing DTMF generated samples are directly passed to the phone line. The routine **OpenAudioOut()** simply opens a file called *audioout* which is closed by **CloseAudioOut()**. The routine **AudioOut(sample)** writes the sample to the file.

The parameters can be found in a separate file. The parameters can be modified to meet the requirements of the ASR system but there are parameters which should not be changed without modifying the User Interface. for example the sample frequency. Most parameters have more to do with the pre-processor and the recognition method. Those will be discussed in the next three chapters. Important parameters for the DSP simulation itself are the sample frequency (SAMPLE_FREQ) to check if the loaded wave file is valid, and the frame rate (FRAME_RATE), which is the number of samples between each block messages.

Debug information can be stored in a string called **DebugText**. Some routines are available to manipulate this string, one can easily add text or integers to this string. It can be cleared and initialised with a default string that contains the date of compilation. Furthermore one can use any default file operation using the stdio C library, which comes in handy when debugging algorithms that deal with large amounts of data.

To make the code portable one must use the following construction:

#ifdef SIMULATION <simulation code> **#else** <DSP code> **#endif**

The parameter **SIMULATION** should be defined in the parameter file. If the DSP code does not contain statements that are unknown in ANSI C the code can be compiled without the SIMULATION parameter defined. This is shown in the "Settings" dialogbox.

For the storage of samples and other data, a file declaring all large data structures is available, the sample buffer is already declared it is called **samples_buf[SAMPLES_BUF/2]**. The parameter **SAMPLES_BUF** should be defined in the parameter file. Because the DSP has no byte addressing the array is an array of words. It is assumed that two samples are stored in one word; the routine **ProcessSample** should take care of this. Efficient implementation of the simulation data structures and accessing them properly makes the code easier to port to the hardware.

4.6 Comparing the simulation with the target system

The simulation environment closely approximates the hardware but there are a few discrepancies. The block message used in the simulation environment is not available on the DSP. On the DSP, the Interrupt Service Routine that corresponds to **ProcessSample** takes care of storing the samples and computes the index of a sample in the sample buffer, which is used to store samples. A polling routine is required to check if the end of a block is reached and the last block should be processed after this limit is reached.

The functionality of the interrupt handling is the same for all interrupts. However the timing of these interrupts, the processing speed and the available memory are different in the simulation environment. Therefore all Interrupt Service Routines are written in assembly for the DSP, which makes them faster and smaller. Polling techniques are used to handle interrupts properly in the DSP. The simulation environment does not support parallelism, so the real-time tests must all be performed on the hardware.

By the routines described above the handling of peripherals is the same as on the DSP. The described routines are written in assembly for the DSP, but their interface to C is the same as the routines used in the simulation environment. The memory limitations that exist on the hardware are not implemented in the simulation environment, one should however use efficient data structures and take the limitations of the DSP into account, otherwise code can not be adjusted to work on the DSP.

Altogether the simulation environment offers exactly what we wanted: a way to develop the ASR system without having to deal with the timing and memory limitation nor with the extensive instruction set of the TMS320C5x.

4.7 Finishing the systems control

The remainder of this report deals only with the recognition task including pre-processing. Therefore we will describe the control unit here. The control unit takes care of proper initialisation and termination on the interrupts that occur when the phone is picked up and when it is put on hook again. This includes turning the display on and off and putting an acknowledgement on the first line of the display using the described display routines. The control unit is implemented in C code that can be compiled with the Microsoft developer Studio and the TMS320 C5x Optimising C compiler.

The main task of the control unit is to parse the grammar described in Figure 2-2 and to execute the recognised commands. Because the grammar is context sensitive a state variable is used. It indicates whether a phone number is entered either through the use of the keypad or through speech or it is to be retrieved or stored in the phone number memory. The memory functions are implemented by using a two dimensional array containing space for 10 phone numbers and 1 line for the last dialled number. The phone number is put on the second line of the display when a phone number is being recognised. When a command is recognised, the associated action is put on the first line of the display. When the delete command is recognised, the last digit is removed from the phone number and the digit is removed form the display. When the "Bel" (dial) command is recognised the routine **OpenAudioOut()** is called and a Boolean **isOpenAudio** is set to indicate that the recognition task is completed and a connection is made. The DTMF tones are generated by calling a routine **ToggleDTMF()**. This routine reads the phone number and generates the DTMF tone for each digit. On the DSP this routine is implemented in assembly and has the same C interface. When the interrupt for putting the phone on hook again occurs, the Boolean **isOpenAudio** is inspected and if necessary the routine **CloseAudioOut()** is called. Because the interrupt service routines are all written in assembly for the DSP version, the main program polles Boolean variables to detect certain events. This requires a different approach than the method described above. The sampler timer interrupt is disabled when the end of a word is found and enabled again when the classification is complete. The DTMF generator timer interrupt is enabled when the number is to be dialled after the connection is made. This interrupt is disabled after the number has been dialled. Then the main program simply waits for the phone on hook interrupt to occur, whereafter the connection is closed.

5. Preprocessing, feature extraction for speech data

In chapter 2. we discussed some features of the speech signal. This chapter is about the technique used to extract those features. We will discuss the mathematical theory of the technique and its implementation in the simulation environment as well as the hardware. The reason for so much attention to the pre-processor is that it has great influence on the achievable quality of recognition and, because the bigger part of the development time is spent on designing and testing a pre-processor in most projects where some intelligent technique is applied.

5.1 Techniques for feature detection and related issues

There is a wide range of techniques available for feature extraction of the speech signal. The most prominent ones are Linear Predictive Coding (LPC) and Fast Fourier transforms (FFT). The LPC technique is based on finding the coefficients of a linear predictive filter. This filter can be expressed as:

$$\hat{s}_n = \sum_{i=1}^{p} a_i s_{n-i}$$

where a are the filter coefficients and s the signal. The standard estimator notation is used. The next sample is estimated using the weighted sum of the last p samples. The coefficients are usually found by the auto-correlation method or the covariance method, this method is described shortly in [17]. The popularity of this method can be explained by the low computational cost compared to FFT-based techniques. A comparison of pre-processing techniques can be found in [18].

The FFT is just an efficient method for computing the Discrete Fourier Transform (DFT) denoted by:

$$X(e^{2\pi i \frac{f}{f_s}}) = \sum_{n=-\infty}^{\infty} x(n) e^{-m 2\pi \frac{f}{f_s}}$$ were, f_s denoted the sampling frequency.

As The FFT method is used in this application, we discuss it further in the next section. The resulting vectors form a spectrogram, which is also discussed in the next section.

Both methods described above form the core of a pre-processor unit. Other issues that have to be reckoned with are:

Separating speech segments:	finding the start and end of each spoken word in the speech signal,
Windowing:	some kind of filtering to limit the range to which the FFT is applied
Normalisation:	scale the resulting vectors to fixed size vectors
Compression:	compress the resulting spectrogram to a fixed number of time slices
Vector Quantisation:	map each time-slice to a so-called code book vector.

Sometimes a filter is applied to stretch the signal in the lower part of the frequency spectrum and to compress it in the higher part. This corresponds to the features we discussed in chapter 2. One commonly used filter is the Mel-frequency bank denoted by the formula below:

$$m = 1125 \log(0.0016 f + 1)$$

However we did not use this filter The issue of vector quantisation is discussed in chapter 7, because it only applies to the pre-processing for the Hidden Markov Model. The general idea is that we have a so-called code book of vectors to represent different classes, each input vector falls into one class. After classifying an input vector it can be treated as a symbol. This is necessary for the chosen Hidden Markov Model type, which uses discrete probability distributions instead of continuous multivariate probability density functions. This choice is explained in chapter 7.

5.2 Pre-processor definitions

First some definitions. We consider a frame to be a vector containing 128 samples, corresponding to 16 [ms] of sampled sound. Thus using the z operator we can define a frame as:

$$F(n) = [x * z^{128n-127}, x * z^{128n-126}, \ldots, x * z^{128n}] = [x(128n-127), x(128n-126), \ldots, x(128n)]$$

A block is considered to be a vector containing 256 consecutive samples, but, unlike frames consecutive blocks overlap. The block is defined by:

$$B(n) = [x * z^{128n-255}, x * z^{128n-254}, \ldots, x * z^{128n}] = [F(n-1), F(n)]$$

The volume in a block, which we will use to separate spoken words is defined as follows:

$$V(n) = \sum_{i=0}^{255} B_i(n) = \sum_{i=0}^{127} F(n-1) + \sum_{i=0}^{127} F(n) = \sum_{i=0}^{255} x(128n-i)$$

There are two constants defined to separate spoken text from background noise. These are v_{trig} and v_{norm} resp. the trigger level and the norm level. The trigger level is the volume in a frame that definitely contains speech signal. The norm level is the minimal volume in a frame containing speech signal. A third constant N_{gap} is the maximum number of blocks containing a volume below the norm level which may be contained in a series of frames belonging to the same spoken word. This corresponds to the amount of silence between the syllables in one keyword and we will refer to this constant as the maximum gap. Now we have to make the assumption that either no speech has been detected yet or we know where the last spoken word ended, which ever is the case. We assume k_0 is the point after which no more than one keyword is stored in the buffers. Furthermore we assume at k_0 there have been a sufficient number of frames containing silence. Once we have a way of spotting spoken words these assumptions hold for known k. Now we give a formal definition of the method to find the spoken keyword, by defining two indices k_{first} and k_{last} so that

$$W = [F(k_{first}), F(k_{first}+1), \ldots, F(k_{last})]$$

is exactly the sequence of samples containing one keyword:

$$k_{first} = \min\arg(V(k) \geq v_{trig} \wedge k > k_0)$$

$$k_{last} = \max\arg(\forall i \in [k - N_{gap} + 1, k] : k > k_{first} : V(i) \geq v_{norm})$$

Now that we know where the keyword begins and ends, we might consider storing the entire speech signal representing this keyword, however it would require a very large buffer to contain all the samples. Instead we store only a small amount of samples and perform an FFT every time a new frame is filled. This requires the computation of 62.5 FFT's per second. We will look more closely at the computational load of the pre-processor later. To be able to calculate the FFT while new samples can still be stored in the buffer, we store three frames in the buffer corresponding to 384 samples. The oldest two frames together form a block to which the FFT is applied. One problem has to be solved yet, and this is the problem of the infinity of the signal in the time domain, which is required (theoretical) to perform an FFT. The problem is partly solved by considering the signal contained in one block to be periodic, thus we define $B_i(n) = B_{i \bmod 256}(n)$. However this causes some problems with continuity at the edges. Hence some kind of windowing filtering is required. Whatever filter we use it must preserve the total energy contained in the signal when all the blocks are considered. One common filter applied to solve this problem is the n-point Hamming window, defined by the function $H_i(n) = 0.54 - 0.46\cos(2\pi \frac{i}{n-1})$, which looks a bit like the Gaussian probability density function. A 256-point FFT is applied to the inner product of the last filled block and the 256-point Hamming window. Were not the complex result but the distance in the complex plane is considered, leaving only 128 frequencies to consider, the resulting vector is then compressed to 16 frequency bands which form one time slice, expressed by the formula:

$$T_f(n) = c \cdot L\left(\sum_{k=f*8+1}^{8(f+1)+1} {}^{10}\log\left|\sum_{j=0}^{255} B_j(n) H_j(256) e^{i2\pi kj}\right|\right)$$

The log function is necessary to extract sufficient information from the lower part of the frequency spectrum. Note that the DC-component, corresponding to, $k = 0$ is not calculated. It is not of interest, because it is already measured by the volume $V(n)$. The factor c is just a scaling factor so that all the values fall into the range [0..255]. The function L is a hard-limiter with minimum $f_{\min}$ and maximum $f_{\max}$. The index j is used instead of i so that it is not confused with the imaginary $i = \sqrt{-1}$. In our implementation all values stored in buffers are either bytes, words, integers or longs; only intermediate results are stored in floats. Also the values in the time slice do not correspond to the real frequencies; they differ a factor $\frac{256}{8000}$. The butter-fly FFT algorithm has been described and proven by many authors in the field. In [2] you can find a very use full and well-explained version.

At the moment the end of a spoken word is detected, the spectrogram formed by the time slice $T(k_{first})\ldots T(k_{last})$ is compressed to a 256-point spectrogram. After compression the spectrogram is scaled or normalised in order to use the full byte range. For the compression each time slice is multiplied by a weight factor defined by

$$w(k,t) = \begin{cases} r(t+1) - r(t) \Leftarrow n_{first}(t) = n_{last}(t) \\ 1.0 - (r(t) - n_{first}(t) \Leftarrow k = n_{first}(t) \wedge n_{first}(t) \neq n_{last}(t) \\ r(t+1) - n_{last}(t) \Leftarrow k = n_{last}(t) \wedge n_{first}(t) \neq n_{last}(t) \\ 1.0 \Leftarrow n_{first}(t) \neq n_{last}(t) \wedge k \neq n_{first}(t) \wedge k \neq n_{last}(t) \end{cases}$$

where,

$$r(t) = \tfrac{t}{16}(k_{last} - k_{first}) \qquad n_{first}(t) = \lfloor r(t) \rfloor \qquad n_{last}(t) = \begin{cases} \lfloor r(t+1) \rfloor \Leftarrow t < 15 \\ k_{last} - 1 \Leftarrow t = 15 \end{cases}$$

This definition deserves some explanation. It takes care of some edge problems. The first option occurs only when time slices need to be stretched, something that will not occur very often. The next two options compensate for the first and last intervals which will be shorter than the intervals in the middle of the sequence. The last option occurs in the default case where a whole number of time slices are compressed to one time slice.

With the previous definition we can now formalise the method of calculating the compressed spectrogram:

$$S_f(t) = \frac{\sum_{k=n_{first}(t)}^{n_{last}(t)} w(k,t) \cdot T_f(k)}{\sum_{k=n_{first}(t)}^{n_{last}(t)} w(k,t)}$$

$S_{\min}$ and $S_{\max}$ are used to scale the spectrogram within the byte range, they are the minimum and maximum value in the spectrogram rounded to integers. The scaled spectrogram is obtained by:

$$\tilde{S}_f(t) = \left\lfloor \frac{255 \cdot (S_f(t) - S_{\min}(t))}{S_{\max}(t) - S_{\min}(t)} \right\rfloor$$

The last step, which is only performed when we use the Hidden Markov Model, is vector quantisation. Assuming we have successfully determined a number of codebook vectors representing $N_{classes}$ different classes, then each time slice in the spectrogram has to match with one of the code book vector. Let m_i denote

the different codebook vectors and C_k the different classes, where each class is defined by a number of codebook vectors. The observation sequence is generated as follows. First the codebook vector is found using the Euclidean norm:

$$m(t) = \arg\min(m \in \bigcup_{k \in [1, N_{classes}]} C_k :: \|m - \tilde{S}(t)\|)$$

Than the number of the class to which the resulting vector belongs is assigned to the next symbol in the observation sequence:

$$o(t) = \arg(k \in [1, N_{classes}] :: m(t) \in C_k)$$

5.3 Implementation and analysis with Matlab™

Matlab is a very powerful tool for building and testing signal processing components. It has a huge amount of built-in signal processing tools and allows fast vector operations. Also the graphical tools included in Matlab are very handy to visualise intermediate results. This makes Matlab a good environment for designing and implementing signal processor prototypes.

The Matlab prototype of our system consists of a set of files. Each file performs one step in the feature extraction. The first step loads a wave file and stores it in a vector. Each file has access to the pre-processor parameters which are stored in a file called **globals.m** This file contains the parameters:

v_{trig} -TRIG_LEVEL, v_{norm} -NORM_LEVEL, N_{gap} -MAX_GAP , $f_{\min}$ -CUT_SPEC_LO, $f_{\max}$ -CUT_SPEC_HI

The input vector is split into a number of smaller vectors using TRIG_LEVEL, NORM_LEVEL and MAX_GAP. These parameters should be adjusted to fit the recording hardware as good as possible; furthermore the sampled speech must contain enough energy to be useful.

The spectrogram is calculated in five steps. In the first step a time-slice is computed by performing an FFT after windowing and applying the Hamming filter. In the second step the number of frequency bands is compressed to NOF_FREQ_BANDS. In the third step the values in the time-slices are scaled and hard limited using CUT_SPEC_LO and CUT_SPEC_HI. In step four the number of time bands is compressed to NOF_TIME_BANDS after this step a 16x16 spectrogram has been computed. In the last step the values in the spectrogram are scaled to benefit from the full byte range. These five steps correspond to the definitions described above. The five steps are performed by five Matlab functions which can be found in Appendix E.1. In Figure 5-1 the computation of a spectrogram is visualised.

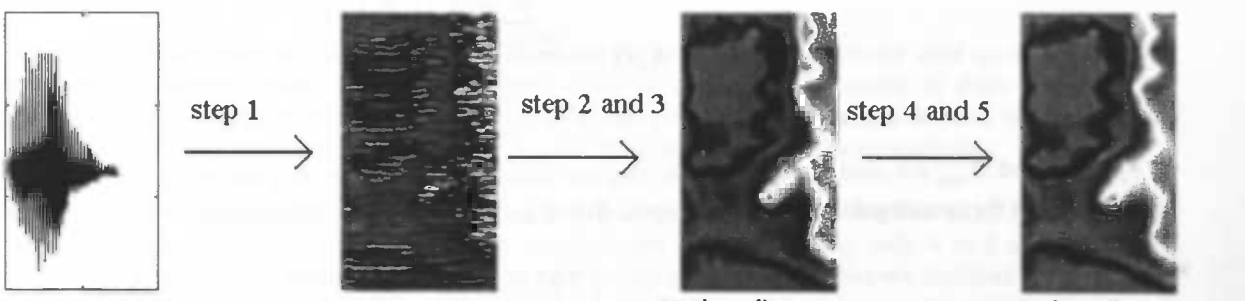


Figure 5-1 Computation of spectrograms

The spectrograms shown in chapter 2 were generated using Matlab and the described pre-processing algorithms. The tests described in chapter two, which are shown in Appendix C show that this pre-processing results in spectrograms containing sufficient information. In the next sections the implementations in the simulating environment and on the hardware are discussed; the Matlab pre-processor is used to measure the quality of the pre-processor implementations.

5.4 Simulation: implementation and results

The parameters in the simulation environment are the same as the parameters in the Matlab pre-processor described above. Some new parameters, dealing with buffer sizes were added. Source code of the discussed algorithms can be found in Appendix E.2. In the simulation samples are processed by the routine **ProcessSample()**. In this routine the samples are put in the sample buffer. An index (**pSample**) is maintained that corresponds to the number of samples in the buffer, not to the number of words in the buffer. At odd indices samples are stored in the Most Significant byte of each word; this is done by shifting the sample eight bits to the left and adding it to the word that contains the previous sample in the Least significant byte. The volume of each block (**level[i_level]**) of 128 samples is calculated by summing all the levels of the samples stored in that block, where **i_level** is the index of such a block. In the routine **ProcessBlock()** the total volume of the samples in the last block of 256 samples (**total_level**) is calculated by adding the levels of the last two blocks of 128 bytes. There are three blocks of 128 stored in the buffer, so that one block can be filled with new samples while the other two blocks can be used for pre-processing.
In **ProcessBlock()** the new time slice is calculated by performing a Butterfly FFT to the last block of 256 samples. At the end of the FFT the time slice is compressed to values and hard-limited, this correspond to step 2 and 3 in the Matlab pre-processor. The resulting time slice is stored in a spectrogram buffer (**spectro_buf[i_spectro]**), this is a circular buffer with index **i_spectro**. The 16 (byte) values in each time slice are packed to 8 words, like the samples in the sample buffer. The size of the spectrogram buffer is determined by the parameter **SPECTRO_BUF**. It should be large enough to contain each word in the language presented in chapter 2, with a sample rate of 8000 Hz and a maximum word length of 1500 ms. This size must be about $8000*1.5/128 \approx 94$.

The total volume and the three parameters **TRIG_LEVEL**, **NORM_LEVEL** and **MAXGAP**, stored in the designated file, are used to detect silence. When the start of a word is found, the volume is greater than **TRIG_LEVEL** and the start of the word was not found yes (**i_spectro == SPECTRO_BUF**), the start of the word in the spectrogram buffer is marked by setting an index (**i_norm**) to the current index of the spectrogram buffer (**i_spectro**). If a gap of silence (more than **MAX_GAP** blocks) occurs **i_norm** is reset to **SPECTRO_BUF**. At the end of the word, detected when **MAX_GAP** blocks of 128 samples with a volume less than **NORM_LEVEL** have been read, the spectrogram is computed in two steps in the routine **NormSpectro()**. This routine takes two parameters: the start of the word (**i_norm**) and the number of time slices to include. Each frequency band is compressed to 16 time slices, corresponding to step 4, by the routine **Compress()** which takes three parameters: the frequency band ranging from 1 to **NOF_FREQBANDS**, the start of the word (**i_norm**) and the number of time slices to included. **Compress()** puts the resulting time slices in a two-dimensional array **spectrogram[NOF_TIMEBANDS][NOF_FREQBANDS]**. After all the frequency bands are compressed, they are scaled to the full byte range, corresponding to step 5 at the end of **NormSpectro()**.

When the Hidden Markov Model Classifier is used, the spectrogram is translated to 16 observation symbols in the routine **GetSymbols()**. For each time slice the closed codebook vector is found using the Euclidean distance. The codebook vectors are stored in the two-dimensional array **CodeBook[CODEBOOK_SIZE][NOF_FREQBANDS/2]**. The codebook vectors, containing byte values, are compressed to half it's length like the time slices in the spectrogram buffer.

The quality of the pre-processor implemented in the simulation environment is judged by the quality of the generated spectrograms. When the pre-processors implemented in the simulation environment and Matlab are applied to the same speech sample, the mean squared error is 0.000 for all generated spectrograms.

5.5 Implementation on target platform

The pre-processor implementation on the TMS320C50 differs at several points from the Simulator implementation. These difference are found in sampling, control, the FFT and the compression of the time slices to an 16 dimensional vector. The differences between the two implementations concerning control have been discussed in the previous chapter.

The interrupt service routine **ProcessSample()** was implemented in assembly because is should be very fast to save time for the computation of the time slices. The use of C interrupt service routines is supported by the compiler but it results in slow code, which is caused by the extra overhead to save registers on the stack and the inefficient assembly code that is generated by the compiler. The assembly deals with some extra scaling of the received samples to select only the significant bits. We do not discuss the assembly algorithm itself because it requires some knowledge about the assembly language that is specific for the TMS320C50 and about the communication with the AIC chip. The functionality of the assembly routine is the same were only buffer access, volume updates and index computation is concerned.

The Butterfly FFT was fully implemented in assembly. Our code was based on the Butterfly FFT available on the Texas Instruments Internet site *http://www.ti.com*, written by Manfred Christ. Some adjustments had to be made:

1. The memory map was adjusted. The original code was written for a TMS320C5x simulator that had a different memory architecture.

2. The buffers used by the original where all on one memory page and directly accessed without updating the data page pointer. The code was modified so that the buffer could be on any data page and data page pointer updates were added.

3. The original code was written to be included in assembly programs or not included in any program at all. The input and output buffers and the call to the main FFT routine were made accessible from C code, header files were added to make the interface to C simple and clear.

The square root, to compute the absolute value of the complex value that result from the FFT, and the log function used in the computation of the time slice proved to take too much processor time, mainly due to the numerical approximation algorithm for the log function included in the math library for the TMS320C5x C-compiler. Instead the mean of the sums of real and imaginary part of each value was used for each frequency band. Although this is a significant difference with the original method, the resulting spectrograms look alike.

5.6 Performance

The accuracy of the sampled data on the DSP is equivalent to the Sound Recorder used on the PC for the simulation environment, when the Sound Recorder is set to a sampling frequency of 8000 Hz and 8 bit accuracy. The AIC chip has a programmable SCF chip that performs a low-pass filter. It is set to a cut-off frequency of 4000 Hz, which corresponds to the Nyquist frequency at the chosen sample rate.

The assembly Butterfly FFT that was initially used has an accuracy of 16 bits. The 256 point FFT requires 8 stages, *See* [2,4]. With 256 multiplications and additions per stage, eventually only 4 to 5 significant bits remain. This proved to be insufficient for both speech recognition methods, thus we replaced the algorithm with its 32 bit variant. A setback of this replacement was the increase in both the number of instructions and the amount of data memory required.

The quality of the pre-processor was again measured by comparing the resulting spectrograms with that of the Matlab version. On first sight the spectrograms look the same; the spectrograms of both version for all thirteen keywords are shown in Appendix F.

Apart from some difference in the volume of the input signal resulting in less distinctive spectrograms, most distortion occurs in the lower part of the frequency spectrum. This is caused mainly by the absence of the log function in the DSP version, log function stretches the lower part of the frequency spectrum.

5.7 Discussion

At the point where the pre-processor was operative a lot of out time had been used. The resulting pre-processor took the greater part of the 10K words available memory and we had to admit that more time should have been spent in selecting proper pre-processing techniques. Our choice was mainly based on our signal processing experience and the available documentation. Decreasing the code size and the required processor time may be achieved by applying the LPC technique with cepstral coefficients which is applied more often in the field of speech recognition though this techniques performs little worse, *See* [18].

The pre-processing alone proved to require intermediate use of float arithmetic. The TMS320C50 is not a very good choice from this perspective because the floating point operations have to be imported from a math library. This requires a huge amount of processor time and memory. The TMS320C3x has an on-chip floating point unit that can perform floating point operations in one instruction cycle, which makes is more appropriate choice for the system we build. One advantage of not using the floating-point DSP is however that one is forced to implement code using integers as much as possible which requires less data memory. Another advantage is the price of the TMS320C5x DSPs which is much lower than the price of the TMS320C3x DSPs.

The adjustments that had to be made to the Butterfly FFT assembly version took much time. This seems to be a general problem with "stolen" code, even with C code. It explains why one takes pride in stealing code and making it work. The remark *Stolen with pride* seems to apply very well in this situation. The lack of documentation may well prevent others from using one's code.

6. The Multilayer Perceptron

In this chapter we discuss the use of a Multilayer Perceptron and its implementation which is the same for both the simulation environment and the TMS320C50. The theoretical background is discussed very briefly because it has already been applied and described by so many others in the field of signal processing and intelligent systems, especially at our department. For those interested in the theoretical background of the MLP one is referred to the open literature.

6.1 The Multilayer Perceptron

Multilayer Perceptrons are used for a wide range of classification and prediction problems. A search for its applications on the Internet gives an enormous list of links, which shows the popularity of this kind of Artificial Neural Network. However the MLP is not used very often for speech recognition as the basic recognition technique, more often a combination of the MLP and a Hidden Markov Model is used. These Hybrid models and their history are briefly discussed in [12]. Some of the theory of these models is described in [13]. However, we had a lot of experience with MLP for pattern recognition task dealing with two dimensional input data, i.e. drs Ter Brugge (RuG) has implemented a system for car number plates recognition and much research was done in the field of character recognition (OCR). The spectrograms produced with the pre-processor described in the previous chapter are especially fit for pattern recognition using the MLP, thus we gave it a try. In the rest of this section we will briefly discuss the MLP. For a full description see [12], a good summary of the MLP presented in [12] is given in the Master Thesis of drs. H. Stevens (RuG).

The Multilayer Perceptron is a Neural Network with several layers, where each layer contains a number of neurons which are Perceptrons , see [3]. A Perceptron has a number of inputs and one output. The sum of these products is filtered by a so-called activity function. Mathematically the Perceptron can be denoted:

$$y = \varphi(-\theta \cdot w_0 + \sum_{k=1}^{p} w_k \cdot x_k)$$

where y is the output, x the input, p the number of inputs, w the weights, θ a bias and φ the activity function. The Sigmoid function is often used as activity function:

$$\varphi(v) = \frac{1}{1 + e^{-av}}$$

where a is the slope parameter. The first layer in a Multilayer Perceptron is called the input layer and does not contain neurons but just holds an input vector. The last layer is called the output layer. The layers in between are called hidden layers. In Multilayer Perceptrons the output of one layer is offered to the next layer, thus data always flows from the input layer to the output layer. Therefore the Multilayer Perceptron is called a Feed Forward Network. The MLP is trained by offering the input and desired output vector to the network. The difference between the output produced by the network and the desired output is the error. The error is used to adjust the weight in all the Perceptrons starting at the output layer and ending at the input layer. The training algorithm is called error back propagation or simply error back.

The adjustments to the weights depend on the derivative of the squared errors of the output of the layer to the weights of the Perceptron and a learning parameter which determines the percentage of adjustment at each training step. The sigmoid function together with the error back propagation algorithm form the training method which is called Gradient descent. The MLP can be seen as an alternative for the Least Mean Square algorithm described in [12], [8] and [3].

When using the MLP for classification, usually each output neuron corresponds to one class. Thus the desired outputs used for training the MLP, called the training set, have one output fully active (usually 1) and all others non-active (usually -1). After training however the neural network does not produce output vectors

containing exactly two possible values for each output but values in a certain range. To determine the recognised class one has to search the neuron with maximal output (activity level). Usually not only the best neuron is searched but it is also checked if the output level of the best neuron (Absolute level) and the difference between the output of the best and second best neuron (Difference level) are high enough. These are called rejection criteria. What is high enough, has to be determine on basis of the desired quality. High values for the minimal Absolute level and minimal Difference level result in lower recognition rates and less errors .

6.2 Neural Networks for speech recognition

Apart from the MLP there are several other types of neural networks used for speech recognition. Though they are not as popular as Hidden Markov Models the number of publications on Neural Networks for speech recognition is huge. Both supervised as unsupervised learning is used. One example of an unsupervised neural net frequently used as part of a ASR system is the Kohonen network or Self Organising Feature Map (SOFM) which also called Linear Vector Quantisation (LVQ). This type of network was used in combination with Hidden Markov Models, *See Appendix H.* In [19] a K Nearest Neighbor (KNN), which is basically a SOFM, is compared to a recurrent network. Recognition rates of 44.4 % and 70.3 % were achieved resp. In general Recurrent networks and TDNN's[7] are used, which have the advantage of using temporal information compared to a "simple" MLP.

6.3 Implementation

The design and training of the MLP was done using Inter*Act*. The spectrograms are offered directly to the MLP, thus there are 256 inputs (number of time frames multiplied with the number of frequency bands). There are 13 keywords to be recognised, we used one output neuron for each keyword. One Hidden was used. As for the number of hidden neurons, neurons in the hidden layer, several experiment were conducted by the students participating in the course Technical AI and 7 hidden neurons were required to get any performance at all, more than 13 neurons resulted in poor generalisation, the best performance was achieved using 9 hidden neurons. Because 9 hidden neurons used as much memory as 10 hidden neurons in the generated C code for the MLP using 16 bit accuracy, 10 hidden neurons were used. Several experiments were conducted to determine the ideal learning rate and learning momentum, *see* [12], which appeared to be 0.7 and 0.2 respectively. One of the Multilayer Perceptrons that was used is shown in Figure 6-1.

[7] Tapped delay Neural Networks

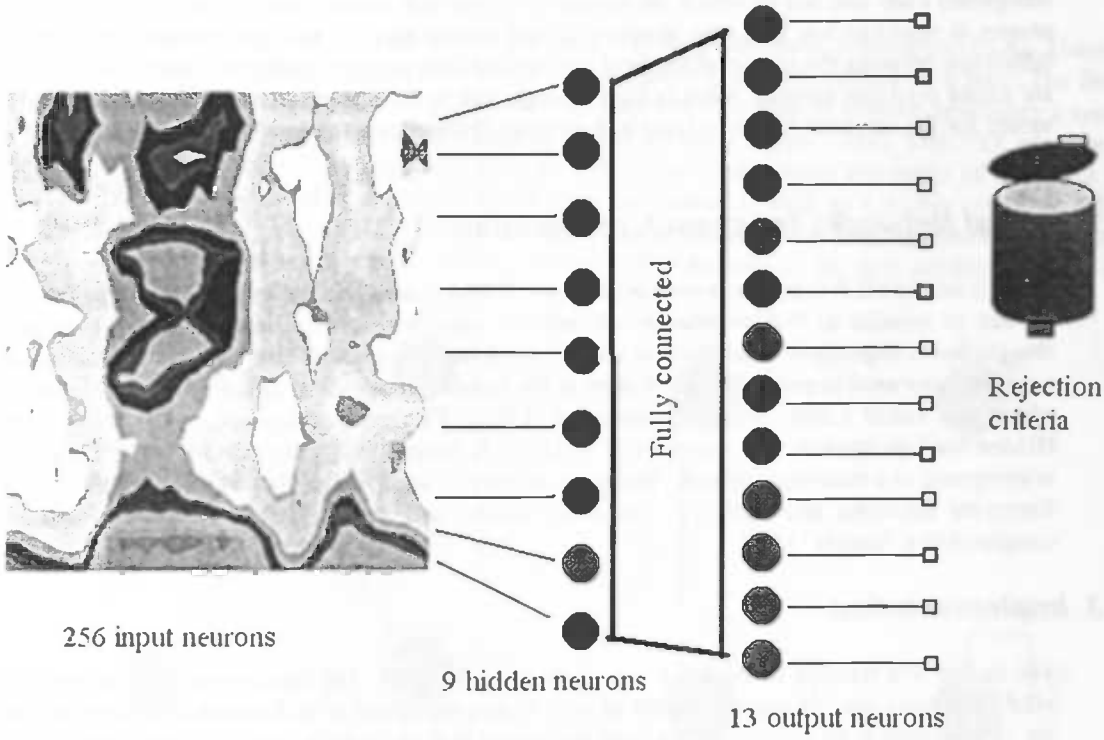


Figure 6-1: The Multilayer Perceptron

In the simulation environment the C code for the MLP generated by InterAct with float arithmetic was used. Nijhuis wrote an additional code generator in InterAct based on compressed inputs. The spectrograms produces by the pre-processor are compressed with respect to the number of frequency bands. Moreover instead of float precision integers arithmetic was used with 16 bit accuracy which required some extra scaling. Although in many publications on the subject of required MLP accuracy it is said that 8 bit accuracy should be sufficient this resulted in huge errors in the calculation of the neuron outputs (600 %) in our experiments. The subject of accuracy requires more research, in our opinion. The source code of the MLP used on the DSP can be found in Appendix G.

Rejection criterion were implemented in a special routine called Recognize(). First the computed spectrogram is offered to the MLP. Than two parameters (THRES_ABS=110 and THRES_DIF 30) ABS_LEVEL and DIFF_LEVEL) were used to determine if the input signal should be rejected. The outputs of the MLP were scaled in the byte range, like the input vector, the spectrogram.

6.4 Discussion

The implementation of the MLP went very smooth. No particular problems, except the accuracy, were encountered. However there are some potential problems. Firstly, there was not very much data available, only 40 test persons, which required repeating the experiments several times. All the test persons are male students at the University, some speaking dialects, therefore recognition problems will probably occur with female speakers and dialects. The problem lies in data acquisition, it is not feasible to acquire a large amount of speech data in short time. Speech data is available for English phonemes, and for Dutch words too but only on a commercial basis. Secondly, the MLP does not benefit from any prior knowledge concerning grammatical features of both the language and the keywords themselves. Using prior knowledge in Neural Networks is a subject of much research at the moment. It is sometimes referred to as the initialisation problem for ANNs. currently ir. W Jansen (RuG) is working on the subject.

7. The Hidden Markov model

7.1 Background

Hidden Markov Models are a very common technique for speech recognition. They became popular in the field of speech recognition in the mid-eighties where a lot of pioneering work was conducted by Juang and Rabiner. They have formalised the model and nowadays most ASR system are based on their work. Although the next definition, stated by Rabiner and Juang (1986), will not be very clarifying we want to give a taste of the model:

A Hidden Markov Model (HMM) is a double stochastic process with an underlying stochastic process that is not observable (i.e. it is hidden), but can only be observed though another set of stochastic processes that generates the sequence of observed symbols.

In the next two sections we will consider Markov Chains, Stochastic Variables and Finite state automatons. After this the definition above will hopefully be more insightful, and we will illustrate it with a simple example using the well known Urn-model approach. The basic idea behind the theory if Hidden Markov Models is that some kind of (observable) data is generated by something that generates data depending on its states, which are not observed. In terms of speech we observe (hear) the sound produced by a voice, but we can not see the vocal cords which produce the sound we hear. The goal we want to achieve is to find a machine that tracks the vocal cords so that we can retrace what was meant by a sound. Although this is a very simplistic view, it can help you to understand the rest of this chapter.

7.2 Markov Chains

Markov Chains are used to model several kinds of stochastic processes, or processed driven by chance. Although a pure probabilistic process, or pure random processes can be considered as the only true random process, there are many other processes that are not random in essence, but chaotic. Chaotic processes are driven by deterministic rules (i.e. the rules of physics), but the amount of influences in the process makes it random in our eyes. An often used example is that of the distribution of ink particles in time when a drop of ink falls into a glass of water. When the space is considered to be discrete, Markov Chains can be used to describe this process.

We simplify our model considering it as follows. There are 5 possible intervals for the ink particles to be in, initially there are N particles. We want to find out the amount of particles in each interval in time: A(t), where A(t) is a 5 dimensional vector. Given the physics of fluids we can say something about the fraction of particles going from one interval to the other. These fractions together form a 5 by 5 transition matrix P, which allows us to say something about A(t) knowing the initial distribution A(0). For the amount A(t) only depends on the ()distribution at A(t-1) and is given by A(t-1) * P. Mathematical interesting is of course the question of convergence and stability, though in is not for our application. These features can be calculated by looking at the Eigen vectors of P, and the diagonal matrix representing P. The most important aspect about Markov Chains is that the distribution at time t is only dependent on the distribution at time t-1.

The number of ink particles in a certain interval is typically expressed using a stochastic variable. If we denote the stochastic variable describing the distribution of particles X and a realisation of this distribution with 1,2..M, then the dependence on previous distributions is described by

$$P(X_t = i | X_{t-1} = j, X_{t-2} = k) = P(X_t = i | X_{t-1} = j)$$

Where i,j and k represent a certain realisation of X. We have introduced the concept of Markov Chains very coarsely. For a more profound description see [3]. In the next section we will have something more to say about them.

7.3 Finite State Automata

Another concept that lies beneath the Hidden Markov Model is the Finite State Automaton (FSA). There are essentially two ways in which the idea of the FSA is used in HMM's, we will discuss both here. The first is that of a language parser or generator. In this case the FSA goes from one state to another generating a symbol that is part of the language. An example of this kind of FSA is shown in Figure 7-1(a). This FSA generate the language which is described by the regular expression a*b(ca+ba*c)*ab* some examples of strings in this language are "ba","abcbaacab" and "abbccaa". Finite states are marked as such by a double circle around the state number. As we will see these kind of FSA's can model the keywords in our language where the length of the vowels can be expressed as the number of recurrent- or self transitions of the state corresponding to that vowel.

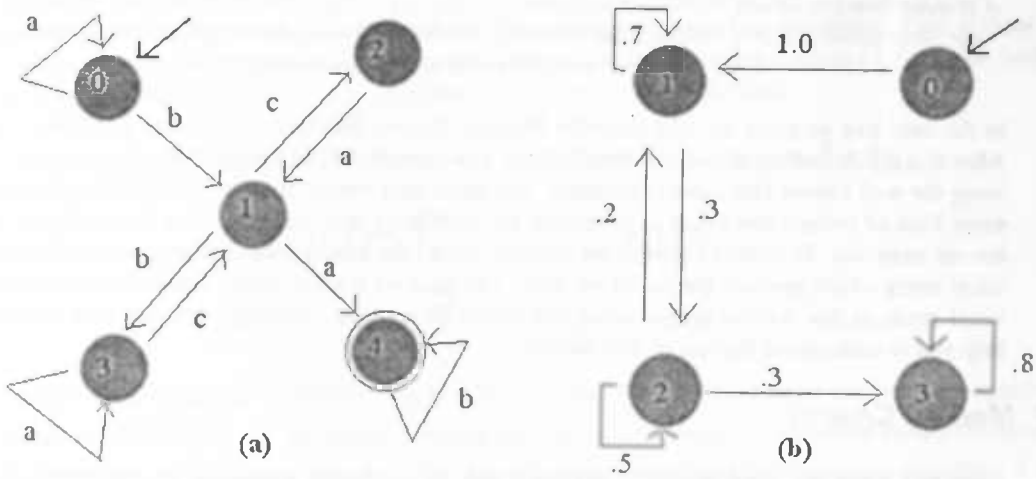


Figure 7-1: Finite State Automata

However for the concept of the FSA to be useful in this context we will have to look at it in a slightly different way. This is also where the Markov Chains come in again. Instead of labelling the transitions with symbols we label the them with transition probabilities, this we call a stochastic automaton. In example is shown in Figure 7-1(b). Here the probability of moving from state 1 to state 2 is 30 %, and, the probability of staying in state 3 is 80 %. These kind of FSA's can be described by a Markov Chain:

$$\pi_0 = (1,0,0,0) \quad \text{, The initial state}$$

$$P = \begin{pmatrix} 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.7 & 0.3 & 0.0 \\ 0.0 & 0.2 & 0.5 & 0.3 \\ 0.0 & 0.0 & 0.0 & 0.8 \end{pmatrix} \text{, The transition probability matrix}$$

The initial distribution is given by (1,0,0,0). We can now calculated the probability of being in a certain state at time t = 3:

$$\pi_{32} = \pi_2 P = \pi_1 PP = \pi_0 PPP = \pi_2 P^3 = (0,1,0,0)P^2 = (0,0.7,0.3,0)P = (0,0.55,0.36,0.09)$$

Thus at t = 3 state 1 is the most likely state. In this model the sum of probabilities of being in a certain state is not always 100 %, due to the fact that at state 3 one "leaves" the model. In a strict Markov Model this is not allowed, though in Hidden Markov Models it is very common as we shall see in the next section. Clearly the probability distribution at a certain time only depends on the previous distribution. This holds for all Markov Models, though there are variations on the theme that allow a dependence on a fixed number of previous distributions. Another assumption is often made, that is, the assumption that the transition probability matrix is invariant or independent of time.

7.4 The theory of HMM's

In a Hidden Markov Model the underlying process is assumed to behave like a finite state automaton with unknown transition probabilities. These unknown probabilities are the hidden part of the model. At each transition a symbol is generated, these are the symbols we observe. This is illustrated by the Urn model we will now discuss.

Assume there are a certain number, say N, of Urns. These Urns contain balls with R different colors, the fraction of balls with a certain colour differ from Urn to Urn. There are two persons participating. The first person starts at a certain Urn and randomly Picks a ball from this Urn. After picking a ball the first person select a next Urn using some random procedure, which is different for each Urn. The second person is standing behind a curtain and can not see the first person. Every time the first person picks a ball he shouts the colour of the ball and moves on. Now the problem to be solved by the second person is to guess the sequence of Urns using the sequence of colours, knowing the number of Urns and the number of colours.

The Hidden Markov Model is defined by (a) a transition probability matrix Γ modelling the random procedure to go from one Urn to another, (b) an observation probability matrix Π corresponding to the probability of a certain ball colour to be selected at each Urn, and (c) an initial state distribution δ corresponding to the probability of all the Urns to be the first. The states are labelled 0,1.. N-1. Clearly the states in the model correspond to the Urns in the example. The observations symbols, corresponding to the colours of the selected balls, are labelled 0,1.. R-1. The observation sequence is denoted $\{S_t : t \in \mathrm{N}\}$. The sequence of states, which is to be estimated is denoted $\{C_t : t \in \mathrm{N}\}$. The definitions indicate that observation and state sequence can be infinitely long, to be able to perform computations in the model the sequence lengths are always limited.
In the remaining part of this chapter we will denote a realisation of the stochastic variable X simply with X, thus $P(X_t = x_t)$ is written as $P(X_t)$.
The transition probability matrix is defined by $(\Gamma)_{ij} = \gamma_{ij} = P(C_t = i, C_{t+1} = j)$,the probability of moving from state i to j. The observation probability matrix is defined by $(\Pi)_{si} = \pi_{si} = P(S_t = s | C_t = i)$, the probability of observing symbol s in state i. The Hidden Markov Model is denoted $\lambda = (\Gamma, \Pi, \delta)$
An important assumption is made:

The realisations of the probabilities of the observation symbols in a sequence are independently distributed, these distributions only depend on the state.

The following properties of the stochastic variables are used to deduce the Forward-Backward and the Baum-Welch algorithm presented further on, for a detailed description see [10]:

1. $\forall t \in [1,T] :: P(S_1, S_2, \ldots S_T | C_t) = P(S_1, S_2, \ldots S_t | C_t) \cdot P(S_{t+1}, S_{t+2}, \ldots S_T | C_t)$
2. $\forall t \in [1, T-1] :: P(S_1, S_2, \ldots S_T | C_t, C_{t+1}) = P(S_1, S_2, \ldots S_t | C_t) \cdot P(S_{t+1}, S_{t+2}, \ldots S_T | C_{t+1})$
3. $\forall t, l : 1 \le t \le l \le T : P(S_l, S_{l+1}, \ldots S_T | C_t, C_{t+1}, \ldots, C_l) = P(S_l, S_{l+1}, \ldots S_T | C_l)$
4. $\forall t \in [1,T] :: P(S_t, S_{t+1}, \ldots S_T | C_t) = P(S_t | C_t) \cdot P(S_{t+1}, S_{t+2}, \ldots S_T | C_t)$

There are two problems to be solved. The first one is that of classification: does a certain observation sequence fit the model, or mathematically: compute $P(S^{(l)} | \lambda)$. The second problem is called the training problem, given a set of observation sequences find the best model or optimise the model. We first address the classification problem. This problem is solved by the so-called Forward-Backward algorithm, this is called the likelihood estimation too.

Two probabilities are defined:

1. The forward probabilities: $\alpha_t(i) = P(S_1, S_2, \ldots, S_t, C_t = i)$
2. The backward probabilities: $\beta_t(i) = P(S_{t+1}, S_{t+2}, \ldots, S_T | C_t = i)$

With these definitions the probability of the observation sequence being in state i at time t can be computed my multiplying the forward and backward probability. The likelihood of an observation sequences can be computed using only the forward probabilities:

$$L_t = \sum_{0}^{N-1} \alpha_t(i) \cdot \beta_t(i) = P(S_1, S_2, \ldots, S_t) \Rightarrow L_T = \sum_{0}^{N-1} \alpha_t(i) = P(S_1, S_2, \ldots, S_T)$$

The Forward-Backward algorithm calculates all the forward and backward probabilities. This is done in an iterative fashion where the forward probabilities are computed with increasing indices (j) and the backward probabilities with decreasing indices (i) , resulting in the following probabilities:

$$\beta_T(i) = 1, \alpha_1(i) = P(C_1 = i) \cdot P(S_1 = s_1 | C_1 = i) = \delta_i \cdot \pi_{s_1 i}$$

$$\alpha_{t+1}(j) = (\sum_{i=0}^{N-1} \alpha_t(i) \cdot \gamma_{ij} \delta_i) \cdot \pi_{s_{t+1} j}$$

$$\beta_t(i) = (\sum_{j=0}^{N-1} \pi_{s_{t+1} j} \cdot \beta_{t+1}(j)) \gamma_{ij}$$

The training problem improves the Hidden Markov Model using a set of observation sequences by maximising the likelihood of $P(S^{(l)} | \lambda)$ for all the observation sequences. The model is therefore called the maximum likelihood estimator, see [8] for estimator theory. The maximum likelihood maximisation algorithm is called the Baum-Welch re-estimation algorithm in honour of its inventors. The notations c,d and e defined below are used accordingly to Levison *et al.* The number of transition from state i to state j, given the observation sequence, is computed:

$$c_{ij} = \sum_{t=0}^{T-1} P(C_t = i, C_{t+1} = j | S_1, S_2, \ldots, S_T) = \gamma_{ij} \sum_{t=0}^{T-2} \alpha_t(i) \cdot \pi_{s_{t+1} j} \beta_{t+1}(j) / L_T$$

The number of occurrences of state j and observation symbol k at time t, given the observation sequence, where time t can also be seen as an index in an observation sequence is computed:

$$d_{jk} = \sum_{t=0}^{T-1} P(S_t = k, C_t = j | S_1, S_2, \ldots, S_T) = \sum_{\{t: s_t = k\}} \alpha_t(j) \cdot \beta_t(j) / L_T$$

The probability that i, given the observation sequence is computed:

$$e_i = P(C_t = i | S_1, S_2, \ldots, S_T) = \alpha_1(i) \cdot \beta_1(i) / L_T$$

These results have been deduced using the properties (1) - (4) described above. Instead of using the commonly used log likelihood a function Q is defined so that the following inequality holds, which gives a useful alternative for the log-likelihood itself.

$$\log(L_T(\bar{\lambda}) / L_T(\lambda)) \geq (Q(\lambda, \bar{\lambda}) - Q(\lambda, \lambda)) / L_T(\lambda)$$

Where $L_T(\lambda)$ denotes the likelihood of an observation sequence evaluated at parameter values for the Hidden Markov Model λ. Clearly replacing an existing model λ with a new model $\bar{\lambda}$ so that $Q(\lambda, \bar{\lambda}) > Q(\lambda, \lambda)$ increases the likelihood, taken into account the concavity of the log function. The function Q is a measure for the probability of sequence to fit the model, expressed by a sort of inner product of the estimated model based on the observation sequence and the existing model. This way it all comes down to

expectation maximisation with maximum likelihood estimators. Therefore the Baum-Welch algorithm belongs to a class of algorithms called **EM**[8] **algorithms**. The function Q is defined:

$$Q(\lambda, \bar{\lambda}) = \sum_{i=0}^{M-1} e_i \log \bar{\delta}_i + \sum_{j=0}^{M-1}\sum_{k=0}^{N-1} d_{ij} \log \bar{\pi}_{kj} + \sum_{i=0}^{M-1}\sum_{j=0}^{M-1} c_{ij} \log \bar{\gamma}_{ij}$$

It is shown by Levison *et al.* the function Q is maximised by setting $\bar{\lambda}$ to $\hat{\lambda} = (\hat{\delta}, \hat{\Pi}, \hat{\Gamma})$, where

$$\hat{\delta} = e_i \,/\, \sum_{i=0}^{M-1} e_i, \hat{\pi}_{kj} = d_{jk} \,/\, \sum_{k=0}^{N-1} d_{kj}, \hat{\gamma}_{ij} = c_{ij} \,/\, \sum_{j=0}^{M-1} c_{ij}$$

The estimation described here is just one step in the re-estimation algorithm. The model is improved in what we call a learning cycle using all the observation sequences in a training set. The learning cycle can be repeated until a desired accuracy is reached, or, more commonly applied, when the difference of the total probability of the training set being fitted by an improved model and the old model is smaller than some pre-defined value.

In Hidden Markov Models the hidden part is mainly the state probability matrix. It is not known when the underlying model, that is to be estimated, moves from one state to the other. There is always some implicit assumption on the probability distribution of the observation probabilities. In the model described above a uniform discrete probability distribution is applied, in the next section alternatives will be presented.

7.5 Using HMM's for speech recognition

The model described above, defined according to Levison *et al.* is only one of many possible model types. The model type suggested by Rabiner, Juang and others differs at several points, of which two are relevant in this context. Both differences concern the Architecture of the model. The first difference is the state transition matrix. In the model proposed by Rabiner *et al.* states are only connected to itself and one successor, thus the state transition matrix becomes a Jordan matrix: this kind of model is called a (Feed) Forward Hidden Markov Model, as the MLP is a Feed Forward Network, a three state Feed Forward HMM is shown in Figure 7-2 .A second difference is that the model proposed by Rabiner *et al.* does not produce or parse symbols within states but at state transitions. Thus the model can be defined by two arrays of transition probabilities: the recurrence probabilities and the next probabilities and two arrays of observation probabilities per state, the observation probabilities at recurrent and (so-called) next transitions.

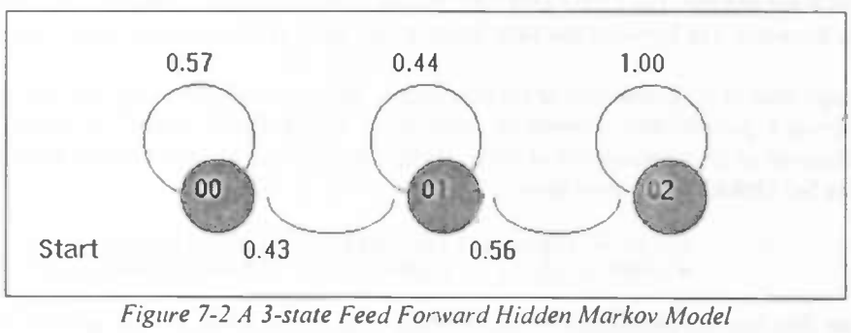


Figure 7-2 A 3-state Feed Forward Hidden Markov Model

The initial state probabilities are no longer necessary, for the first state is now uniquely defined. The testing and training algorithm changes slightly, where the forward and backward probabilities are computed slightly different, we discuss these modifications on an implementational level in the next section.

The uniform discrete probability distribution described, the one we implemented, is not always applied in the context of speech recognition. A discrete alternative is a multi-binominal distribution. All discrete forms require a form of vector quantisation to limit the number of observation symbols. J. Tebelskis argues that

[8] Expectation Maximisation

vector quantisation introduces unnecessary distortion. In his Post Doctorate Thesis he applies a continuous multivariate Gaussian probability distribution for the observation probabilities. and avoids vector quantisation. In his model the mean and standard deviations are stored in what is the observation probability matrix in our model. He takes into account that the observations are correlated, but without further argument he states that the matrices of means and standard deviations are diagonal matrices. Thus he does not compute and use the correlation of the observation probabilities.

One single Hidden Markov Model is not sufficient for classification. Usually one Hidden Markov Model is trained per keyword. The set of Hidden Markov Models to classify keywords from a certain language has yet to be integrated to construct a classifier. There are several techniques available to construct a classifier. One approach is to merge the Hidden Markov Models into one single Hidden Markov Model. To determine what keyword fits the model best, after the probability of the observation sequence has been computed, is to use a decoding algorithm. Usually a Viterbi algorithm is applied, described by Rabiner and Juang. This algorithm determines the probability of a state sequence which can be used to retrieve the most likely state sequence and thus the most likely keyword. Instead of merging the models after they have been trained, the entire model can be constructed and trained using a technique called Bayesian Model Merging, *see* [1]. We implemented a method that has much in common with the MLP classification described in the previous chapter.

The pre-processor we apply does not take account of variable formant or phoneme length. The variability in speaking speed is compensated by the compression of the frequency bands in the spectrogram, *see chapter 5*. Two methods to compensate for the variability in formant lengths are discussed in [5] and [11].

7.6 Implementation

The training and testing of the Kohonen network is described in Appendix H. Vector quantisation turns a spectrogram into an observation sequence of length 16, corresponding to the number of time slices in the normalised spectrogram. The number of input symbols, equal to the number of codebook vectors in the Kohonen Network, is the same for all Hidden Markov Models. The number of states for each keyword was determined experimentally as described in Appendix I.1.

How the Hidden Markov Model Classifier was trained with the InterActive Hidden Markov Model Program (Ia*HMM*) is explained in Appendix I.2. With Ia*HMM* C-code was generated. The C-code of the classificating routine can be found in Appendix I.3. The classifier tests each Hidden Markov Model in the set with a given observation sequence. After the probabilities (scores) have been computed rejection criteria equal to that of the MLP are applied. The label of the best Hidden Markov Model is returned, with each model corresponding to one keyword. The keyword has been found at this point or the word has been rejected.

The logarithm of the probability of the observation sequence being fitted by the HMM is computed only using the forward probabilities. Generated code for a single HMM model is shown in Appendix I.4. The modification of the computation of these probabilities compared to the model described above occurs in the routine **SetAlpha()** in the inner loop:

```
accum = alphas[i][j]*GetP(model,seq[i],j,0);
alphas[i+1][j] = alphas[i][j-1]*GetP(model,seq[i],j-1,1)+accum;
```

On the first line the probability of the recurrent case is computed. On the second line the probability of the next transition is computed and the two probabilities are added. Thus the forward probability is set to the probability of observing symbol seq[i] and moving to state j at a certain point in the sequence, clearly this is a minor change of the model described above. The memory required for the computation of the forward probabilities is shared among the Hidden Markov Models in a classifier, where the reserved memory is set to be large enough for the model containing the most states.

7.7 Discussion

The modification of only using recurrent and next transitions greatly reduces required testing and training time and saves much memory. The memory size does no grow with[9] $O(n^2)$ where n is the number of states, but with O(n), where the use of observation probabilities changes the memory use from O(n*k) to O(2*n*k) = O(n*k), where k denotes the number of observation symbols. Unfortunately the last modification makes an often applied optimisation of using the logarithms of the forward probabilities inapplicable. This optimisation would have no effect. Firstly, the multiplications would turn into addition, this is not an improvement in our case because the TMS320C50 performs a multiplication as fast as an addition. Secondly, the log alpha's contain most information because the probabilities are very small for most models. Therefore they require less storage than the alphas, this is a serious drawback because now the forward probabilities have to computed using float or worse even double arithmetic, which takes 2 to 4 times as much memory and the required processor speed increases significantly. Clearly the requirements on processor speed which are higher with this optimisation not being applicable, could have been relaxed would the TMS320C3x DSP have been used.

[9] The growth is expressed with the so called big O notation, well-known in the fields of Analysis and Algorithmics

8. Experiments and evaluation

In this chapter we present the results of some of the performed experiments as well as an overview of the hardware requirements and development time for both recognition methods and supporting (pre-processor) code. These results are discussed in the next chapter.

8.1 Evaluation criteria corresponding measurements

The most important criterion is obviously the recognition rate. However, as we presented this report as a feasibility study, there are several other aspect to consider. One important question to be answered is ofcourse: What does is all cost. In order to answer this question we have to consider both hardware requirements and development times. Though we already considered these criteria in chapter 1, we repeat them one more time: (1) Recognition rate (performance); (2) Realisability; (3) Hardware requirements: accuracy, Memory and speed; (4) Realisation time. In the remaining part of this chapter we will review all the components presented in chapters 4 to 7 with respect to three of the four criteria. The second, realisability, is an important one, however at this point we conclude that both methods including the required pre-processing can be implemented on the TMS320C50 though some extra memory was required.

8.2 Performance of the Multilayer Perceptron

Several experiments were conducted with varying rejection criteria and different amounts of hidden neurons. The results of the experiments with different hidden neurons are summarised in Table 8-1. Code generation with InterAct results in the same net for nine and ten hidden neurons, thus ten hidden neurons were used. The average recognition rate, without rejection criteria, is 93 percent for this MLP. Experiments showed that MLP's containing over 12 hidden neurons have problems generalising.

#hidden neurons	Recognition rate on train-set	Recognition rate on test-set	train-/test percentage
10	99.1	93.8	70/30
10	98.7	92.6	70/30
9	95,8	90.3	75/25
9	97.0	91.5	87/13
9	95.9	86.9	81/19
8	97.9	84.3	75/25
8	96.3	88.2	75/25

Table 8-1 Results varying number of hidden neurons

The rejection criteria can be determined by searching the for thresholds that result in a desired recognition or error rate. However we conducted several experiments manually just to test the influence of rejection criteria on the error and recognition rate. The results for a 256-9-13 MLP are shown in Table 8-2.

Abs threshold	Diff. Threshold	Recognition rate	Error rate
0.95	0.7	50.0	0.77
0.9	0.9	44.9	0.31
0.9	0.8	77.5	0.92
0.9	0.7	80.0	1.23
0.85	0.7	83.9	2.15
0.8	0.6	87.1	2.46
0.8	0.45	88.0	2.46
0.8	0.2	88.3	2.62
0.2	0.0	94.0	5.85

Table 8-2 The effect so rejection criteria on the MLP

8.3 Performance of the HMM Classifier

Extensive testing and training results for the HMM Classifier can be found in Appendix I. Because the recognition rate on the test set was quite low we conducted three experiments: (a) determining the recognition rate of the best and second best model, (b) determining the recognition rate of the best and second best model form a classifier trained with all available data, (c) the influence of rejection criteria on the classifier used in (b). The results of the first experiment or shown in Table 8-3. Recognition rate of the classifier on the test-set is 67.4.

keyword	best model	Recognition rate of best model	second best model	Recognition rate of second best model
"0"	"0"	84.0	"Bel"	14.0
"1"	"1"	85.7	"7"	7.1
"2"	"3"	42.9	"2"	18.6
"3"	"3"	29.6	"2"	26.8
"4"	"4"	88.6	"2"	4.3
"5"	"5"	71.4	"7"	11.4
"6"	"6"	38.3	"2"	35.2
"7"	"7"	85.7	"M"	4.3
"8"	"8"	100.0	-	-
"9"	"9"	58.6	"7"	35.7
"M"	"9"	31.0	"9"	21.9
"Bel"	"Bel"	100.0	-	-
"Opnieuw"	"Opnieuw"	90.0	"9"	5.7

Table 8-3 Recognition rate HMM's (best and second best) trained on 54 % of data

The second experiment was conducted in the same manner, only now all available data was used to train and test the model. Results are shown in Table 8-4. The recognition rate of this classifier was 63.15. Clearly these test results give no indication whatsoever of the performance of this classifier. Nevertheless the results do give an indication of the separability of the data after vector quantisation. More will be said about this in the next chapter.

keyword	best model	Recognition rate of best model	second best model	Recognition rate of second best model
"0"	"0"	66.3	"Bel"	13.4
"1"	"1"	61.1	"9"	10.5
"2"	"2"	52.9	"1"	7.0
"3"	"3"	51.2	"1"	9.9
"4"	"4"	58.1	"9"	11.1
"5"	"5"	65.1	"Opnieuw"	8.1
"6"	"6"	72.7	"7"	5.2
"7"	"7"	49.4	"9"	9.3
"8"	"8"	80.2	"5"	9.9
"9"	"9"	62.8	"M"	11.6
"M"	"M"	69.8	"9"	10.5
"Bel"	"Bel"	53.5	"0"	22.1
"Opnieuw"	"Opnieuw"	77.9	"0"	9.3

Table 8-4 Recognition rate HMM's, all data used

With the last experiment the influence of rejection criteria is measured. The results are shown in Table 8-5. Rejection criteria must maximise both recognition and rejection rate, if they are used, so that the error is minimal. The difference in the alpha difference threshold has a greater impact on the rejection rate, thus is must be chosen with care. However looking at the recognition rates we may well skip rejection criteria at all.

Thresh Abs.	Thresh Dif.	Recognition rate	Rejection Rate
2000	5	32.1	63.1
2000	3	42.4	51.6
2000	1	54.7	25.3
2000	0	63.2	0.0
75	2	47.9	41.4
75	1.5	54.7	25.2
75	1	51.4	33.7
50	5	31.9	66.5
50	3	42.1	52.0
50	1	54.4	25.8
50	0	62.7	0.7
40	5	27.0	71.5
40	3	35.6	59.2
40	1	46.4	35.5
40	0	53.4	0.13
30	0	22.4	57.4

Table 8-5 The effects of rejection criteria on HMM's

8.4 Hardware requirements

The hardware requirements are highly dependent on the used accuracy. In the current implementation float arithmetic was required at several points, some improvement will be necessary at this point to optimise for the TMS320C50. However we will look at hardware requirements for the current working version of the system. First of all, the required speed which is the most important criteria, for if the DSP is not fast enough nothing can be done but look for another one. As we have mentioned before the only critical point in the execution is the real-time processing of samples, this issue has been covered in chapter 5. Experiments showed that after the processing of a block of 128 samples there is time left for about 1000 instruction cycles, thus 99 % of the processor time is used.

The required memory for the pre-processor, the MLP and the HMM classifier are shown in Table 8-6. All supporting assembly routines (interrupt service routines, control, display routines etc.) are included in the pre-processor. In the total require memory of the MLP and HMM classifier the required memory for the pre-processor is included.

Component	Text	Data	Constants	Total
Pre-processor	3274	1200	2313	6787
MLP	2264	46	2628	**11725**
HMM Classifier	3674	44	4416	**14921**

Table 8-6 Memory requirements (required words)

The time required to pre-process and recognise one keyword were measured using the DSP. The pre-processing one word takes about 0.5 [s]. The computation of the MLP outputs takes about 1.5 [s]. The computation of the likelihoods in the HMM Classifier take about 2.5 [s].

8.5 Development time

Three people where directly involved in this project. One being a hardware engineer, the other two computer scientist. No administration of the time spend on several components of the system was made. I should be said that the total development of the system was underestimated which complicated affairs on the hardware development. The project required a lot of study particularly to the TMS320C50 and Hidden Markov Models. Only the time spent on implementation is taken into account, for required studying time is highly dependent on the background of the engineers involved. An estimation of the development time for several components is listed in Table 8-7.

Component	man-weeks	percentage
Hardware	26	65
Pre-processor	8	20
Kohonen (LVQ)	1	2.5
Simulation	3	7.5
MLP	1	2.5
HMM's	1	2.5
Total	**40**	**100**

Table 8-7 Development time

8.6 Summary

With respect to the criteria, the Multilayer Perceptron performs best without a doubt. The recognition rate of the MLP is about 20 % higher than that of the HMM classifier. Still, with recognition rate of less than 95 % on the keywords the probability of recognising a number of 10 digits without errors is less than 60 % as current. As for the speed of the system the MLP beats the HMM by a second, but still requires 1 second for each keyword, which makes a total of over 10 seconds for a command sequence including a phone number. The entire system requires more memory than available at first, with the MLP again coming out best with only about 12 K (words) compared to about 15 K for the HMM Classifier. The development time appears to be consumed mainly by hardware development and pre-processor implementation. The applied recognition method does not appear to require a large amount of development time, provided one is familiar with the technique. Aside from these experimental results there is more to be said about the several components of the system, as will be in the next chapter.

9. Discussion

9.1 Which is best ?

With regard to the results presented in the previous chapter, this question is easily answered: The MLP beats the HMM by far. However, considering the recognition rate and speed, the system cannot be considered a practical application; imagine having to repeat a command several times and waiting for several seconds before the system reacts at all. *How to irritate people* seems to be a better name than *practical application.*

The MLP had two major advantages over the HMM which explain why it comes out best. The HMM classifier was offered less information in the first place. We already mentioned that the LVQ introduces distortion. Clearly there is a huge difference between the spectrogram offered to the MLP containing 256 bytes and an observation sequence with only 24 different symbols. Moreover the HMM implemented is limited by several assumptions, i.e. the assumption of the underlying probability distributions of observation symbols and the assumption that the human voice can be modelled by a finite state automaton.

But with the MLP performing better than the HMM classifier, how can the popularity of the HMM approach be explained ? First of all it should be said that HMM's generally performs better, actually recent publications show they are as current the best method for speech recognition. Limitations of our implementation of the HMM classifier were caused by strict limitation on the development time. The success of HMM's is generally subscribed to the use of *a priori* knowledge: in some sense the model contains information about both the numbers of phonemes in a word and the number of possible phonemes. Methods for using *a priori* knowledge in a neural networks is a subject of scientific research which has as yet no resulted in performance achieved with HMM's . In other fields neuro-fuzzy[10] techniques, which benefit form *a priori* knowledge through the use of fuzzy logic, appears to be quite succesful. Moreover HMM's are easier to understand with respect to their internal workings. A human organ is modelled with a stochastic automaton, each state corresponding to a particular phoneme, thus each probability in the model corresponds to a known event. Hence within HMM's the weak parts are easier to find and improve, i.e. by using a Viterbi algorithm one can easily find the states which give rise to a specific output. Which part of the MLP is responsible for a specific output and how the behaviour of the MLP can be explained in terms of the modelled object are questions which are still hard to answer. To put it in a different way: the weights in the MLP have no specific meaning. Especially mathematicians will prefer a well defined and understandable model over a "black-box" neural network.

9.2 Problems

Developing hardware takes the bigger part of the available time. As opposed to software bugs, hardware bugs are harder to spot and it takes more time to correct them. One has to be very careful with setting up experiments to test the hardware and interpreting the results, for hardware response is influenced by measurements (Heisselberg inequality). Subsequently one has to be very careful not to underestimate the hardware development time. The TMS320C50 has a huge instruction set. Though small and simple assembly programs are not hard to write one has to take into account that effective use of the DSP requires an understanding of its design and the available instruction set. Again this takes time.

The performance of the system is mainly determined by the hardware and pre-processing technique. In our case we failed to fit the software in memory and the response time of the system was too low. Performance with regard to issues like word length (accuracy), memory requirements and speed one finds in scientific publications are often hard to achieve due to target specific implementational problems. Savings on memory, accuracy or speed can only be achieved at the cost of one another. Pre-processing and both recognition methods require floating-point-operations as current which are expensive with regard to speed and memory. Moreover the systems recognition rate is too low to make it of practical use.

[10] Hybrids of neural networks and fuzzy logic

9.3 Recommendations

Without a doubt the system can be improved on several points. First of all, as we have noted before, the TMS320C50 is probably not the best choice for the application we built. The TMS320C3x offers a good alternative apart from the huge amount of DSP's available. But then again, nowadays practically everyone has a PC with modem at home. We may thus consider using a PC as target platform for the application. This will save us all the trouble of a hardware implementation including the strict requirements it imposes on the system.

Both MLP's and HMM's require a huge amount of constant numbers: the weights in the MLP and probabilities in the HMM. Moreover processor time was saved by using lookup tables for several function, i.e. the sigmoid in the perceptrons and the sine and cosine in the HMM. These are all stored in RAM. Typically these can be stored in programmable ROM: for the TMS320C3x and TMS320C5x detailed circuit descriptions are available for EPROM extensions of the started kit boards.

Apart from which recognition method is applied performance is highly dependent on the pre-processing techniques applied. The applied pre-processing appears to require a large amount of memory. Though several publications (among which [18]) show that FFT-bases techniques perform slightly better. They require by far more memory. Alternatively a LPC or Formant Coders, *See* [17], may be used. A first order derivative or difference vector is often used, however with the kind of Hidden Markov Models we applied (using observations at transitions), this might be an overkill Moreover several pre-processing techniques are available as hardware components relieving the DSP of this task.

Clearly the HMM classifier does not only perform bad, it requires a relatively large amount of memory and is to slow to be of practical use. With regard to the performance several adjustments can be made to increase performance. First of all the discrete observation probability distributions can be replaced with continues multivariate probability density functions, *See* [11, 14]. This way the vector quantisation can be omitted, and more information can be used for classification. Secondly, the recognition rate can be improved by modelling or compensating for (time-warping) different phoneme lengths, *See* [5, 11]. Memory requirements can be reduced by using one single Hidden Markov Model instead of one for each keyword [1, 20]. This would require the use of a decoding algorithm, i.e. Viterbi. A more daring approach would be to use a general HMM instead of the (Feed) Forward HMM, allowing common phonemes to be shared. Though this would reduce memory requirements we do not expect a better performance for other HMM architectures.

The MLP is actually a very simple neural architecture. We have already presented some alternatives in chapter 6: recurrent networks, SOFM and TDNN's. The main advantage of these types of ANN's over the MLP is the use of temporal information either through recurrence or through tapped delay lines. These types of ANN's are more closely related to HMM's.

An interesting alternative for the recognition methods presented is the combination of a neural networks and Hidden Markov Models. These hybrid models are currently the subject of a lot of research in the field of speech recognition, and the appear to be very successful. In [13] a HMM/NN hybrid is described where two neural networks are used per state: one for the estimation of the likelihood of the observation sequences and another one for the re-estimation (learning) of the observation and transition probabilities. A recognition rate of 75 % is achieved on the TIMIT database. Clearly such a model would require even a larger amount of memory than the models described in this report.

We did not benefit in any way from context information on a grammatical level. For example the grammar presented in chapter 2 ensures that after the keyword "M" only one digit is to be recognised. However in our system the advantage of using context information is limited. The use of grammatical information is applied more often for continues speech recognition. Moreover semantic information can be used to improve performance.

9.4 Commercial products

Several speech recognition systems have become available recently. Software for speaker dependent speech recognition supporting over 17,000 words are available for less than F.200,-. Specific hardware for telephone applications are widely available too, these products cost around $600. Dutch telephone companies are going to offer voice dialling, supporting 20 to 40 phone numbers *(NRC Handsblad, august 12 1997)*. This service will cost only a few dimes per call. Generally over 98 % recognition rate is achieved with commercial products. Clearly our system can not compete with existing commercial products (for the moment).

Thus both speech recognition techniques and signal processing hardware are available to develop a useful speech recognition system. However it is practically impossible to determine what techniques are applied in these commercial products. Because they require some kind of on-line training we think the recognition method in these products is not as fancy as those presented in scientific publications which are widely available on the internet. The main problem with scientific publications about ASR appears to be that practically no attention is paid to hardware requirements although most of the development time is spend on hardware and the hardware interface.

9.5 Final word

After reading this report some may feel somewhat disappointed with the final results. However it was never our intention to build a tip-top speech recognition system: we tried to present the methods from and engineers point of view. Hence the goals of developing a working system and gaining experience were more important to us than performance. We thought it important to show how a ASR system can be developed, as this is of interest to engineers. The references to publications on pre-processing and Hidden Markov Models and the pitfalls discussed throughout this report will be very helpful to anyone wishing to develop a better ASR system using HMM's and/or DSP's. Furthermore we attempted to provide sufficient information to find ways of improving the current implementation. We believe, by following the recommendations above, the current system may well be turned into a success.

Hope this has been of interest to you. If you have been, I am most awfully pleased.

10. References

10.1 Literature

|1| *Hidden Markov Model Induction by Bayesian Model Merging*
A. Stolcke & S. Omohundro

|2| *Introduction to algorithms*
Cormen, Leierson & Rivest

|3| *Mathematische Modellen*
Dehling & Trentelman
University of Groningen

|4| *Digital Signal Processing, a practical approach*
E.C. Ifeachor & B.W. Jervis

|5| *Non-stationary Hidden Markov Models for Speech Recognition*
D.X. Sun & L. Deng

|6| *Hidden Markov Models, Estimation and Control*
R.J. Elliot *et. al.*

|7| *A tutorial on Hidden Markov Models and Selected Applications in Speech Recognition*
L.R. Rabiner

|8| *Inleiding Statistiek*
Dehling
University of Groningen

|9| *Inleiding Waarschijnlijkheidrekening*
Dehling & Kalma
University of Groningen

|10| *Hidden Markov and Other Models for Discrete-valued Time Series*
MacDonald & W. Zucchini

|11| *Modeling variability between and within speech segments for automatic speech recognition*
W.J. Holmes

|12| *Neural Networks*
S. Haykin (1994)

|13| *1- Hidden Neural Networks: a framework for HMM/NN Hybrids*
2- Joint Estimations of Parameters in Hidden Neural Networks
S.K. Riis & A. Krogh

|14| *Speech Recognition using Neural Networks*
J. Tebelskis (1995)

|15| *TMS320C2x/C2x/C2xx/C5c Optimizing C Compiler*
Texas Instruments (1995)

[16] *TMS320C50 User's Guide*
Texas Instruments (1993)

[17] *Speech analysis*
T. Robinson (1997)
Cambridge University Engineering Department

[18] *A comparison of pre-processors for the Cambridge recurrent error propagation network speech recognition system*
T. Robinson *et. al.*
Cambridge University Engineering Department

[19] *Dynamic Error Propagation Networks*
A.J. Robinson
Cambridge University Engineering Department

[20] *Identification of Hidden Markov Models with Grouped state values*
I.B. Collings & J.B. Moore
Cooperative Research Centre for Sensor Signal and Information Processing SPRI Building Technology Park Adelaide &
Department of Systems Engineering, RSISE, Australian National University

10.2 Other resources

The program written for the construction of Hidden Markov Model Classifiers was based on source code for Hidden Markov Models written by R. Myers. This code is available on the Internet. This source code implements models corresponding to those of Rabiner *et. al.* as described in chapter 7.

The phonetic fonts used in this report were downloaded from internet. The are provided by The Summer Institute of Linguistics (SIL), this is an organisation of linguists dedicated to the study and promotion of the thousands of minority languages around the world. International Publishing Services serves SIL by developing products to assist in the publication of linguistic texts. The package name is SILIPA93 and it can be installed very easily on a windows platform.

Appendix A: List of Figures and Tables

Appendix B: Phonemes

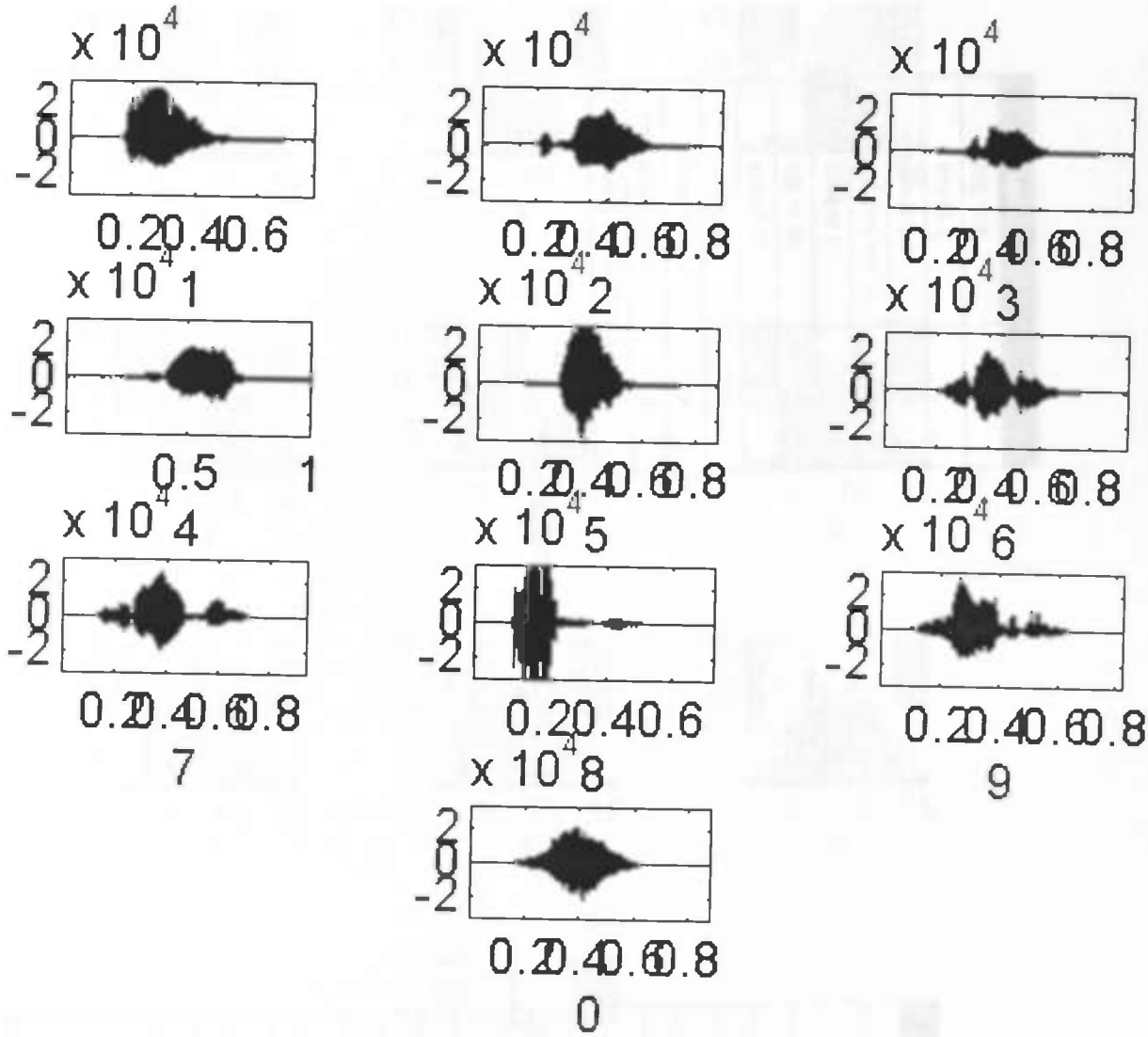
Phoneme	Sounds like
I	Vowels
a:	a in fast
æ	a in fat
ʌ	u in but
ə:	ur in burst
e	e in let
εə	a in care
i	i in will
i:	ee in free
iə	ere in here
ou	o in stone
ɔ	o in not
ɔ:	aw in law
u	oo in foot
u:	oo in food
uə	oor in poor
ə	a in ago
ai	i in wine
au	ow in how
ei	a in fate
ɔi	oy in boy
ã	a in the French blanc
õ	o in the French bon
y	ij in the Dutch blij
ɛ̃	i in the French vin

II	Consonants
g	g in get
j	y in yes
ŋ	ng in sing
ʒ	j in the Dutch journaal
dʒ	j in joke
ʃ	sh in she
ð	th in this
θ	th in thin
w	w in well
R	r in the Dutch bier
x	ch in the Dutch lach

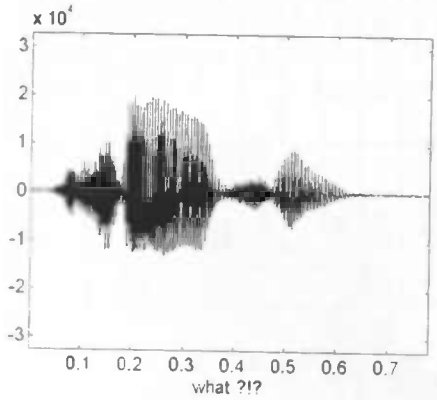
Appendix C:

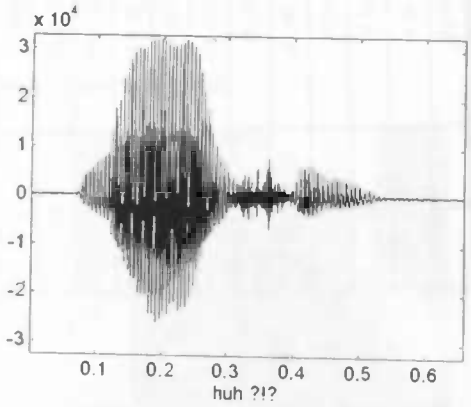
Appendix C: Puzzles for determining identifying features

Appendix C.1 Time plots

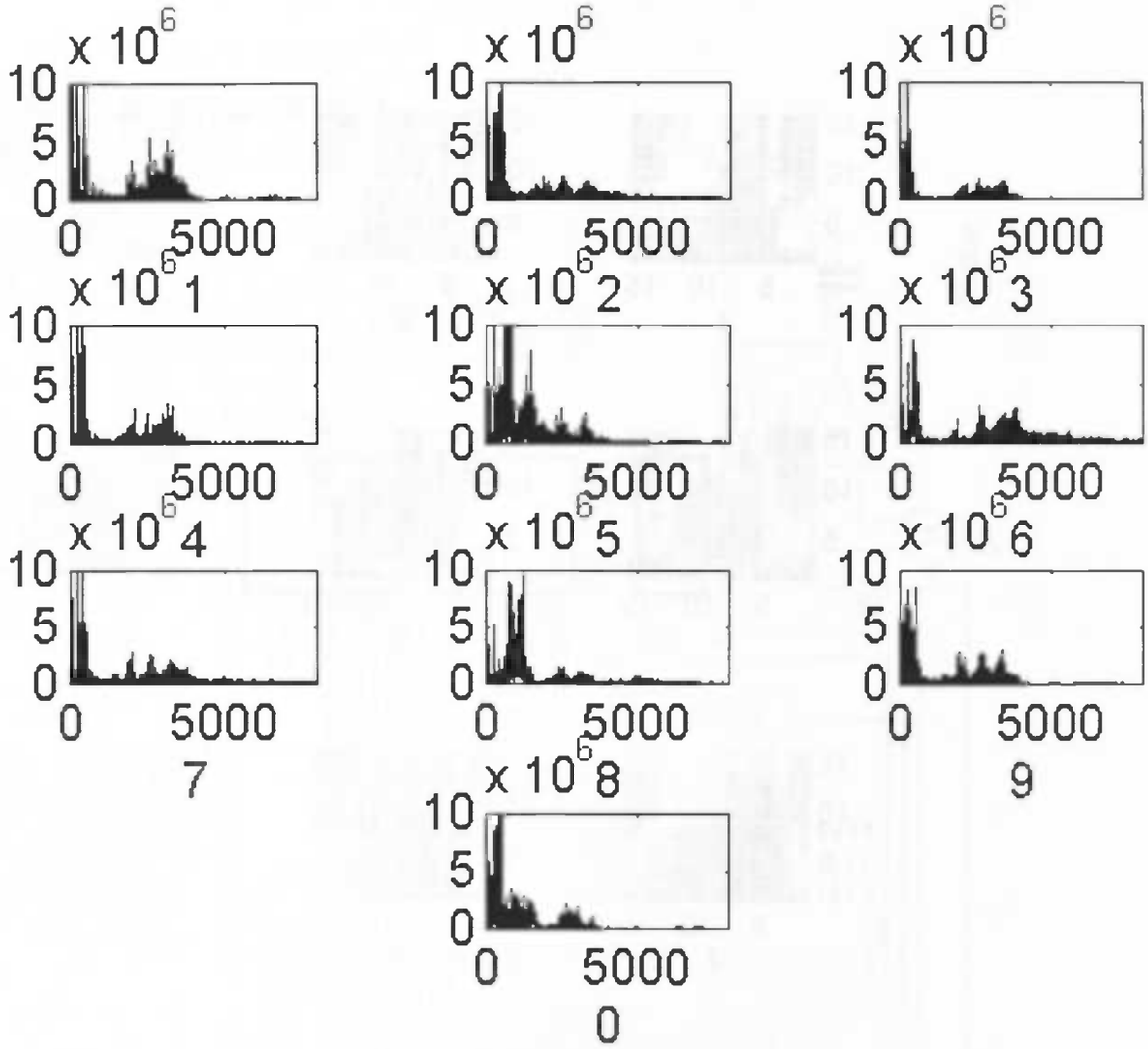


Classify the next two speech signals

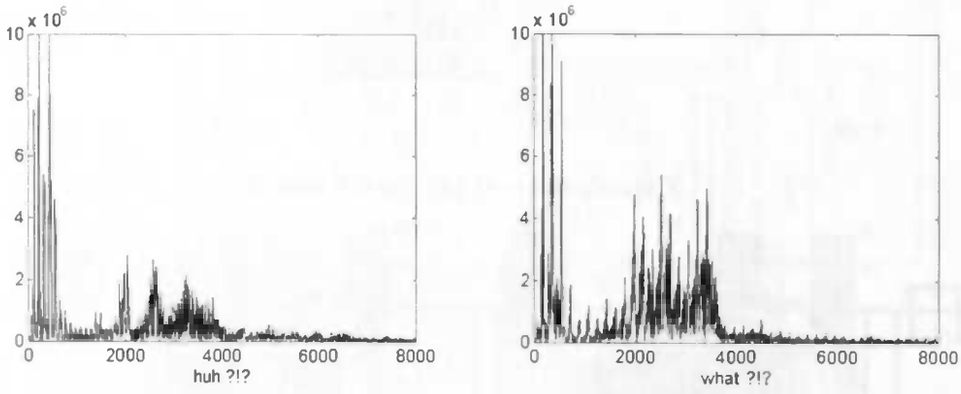




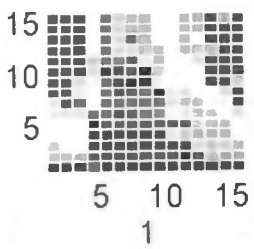
Appendix C.2 Frequency plots

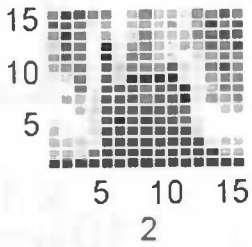


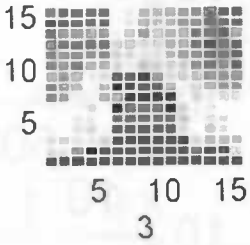
Classify the next two speech signals

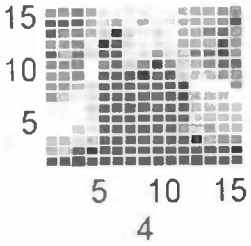


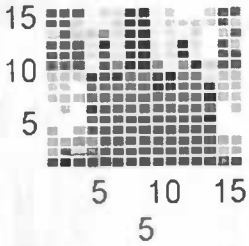
Appendix C.3 Spectrograms

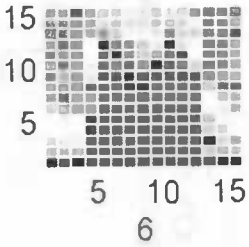


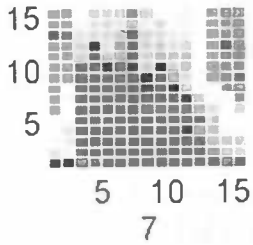


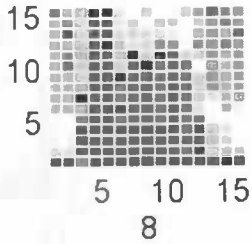


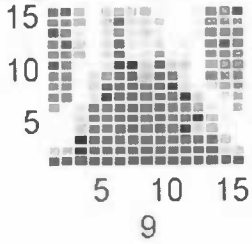


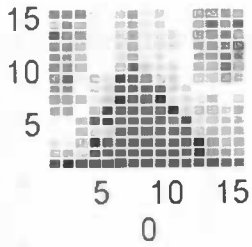




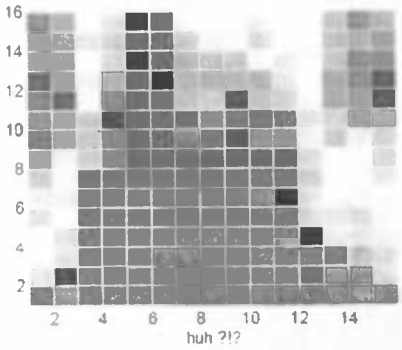


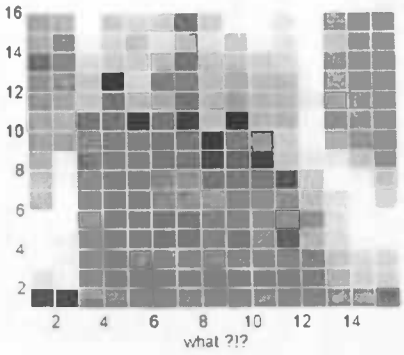




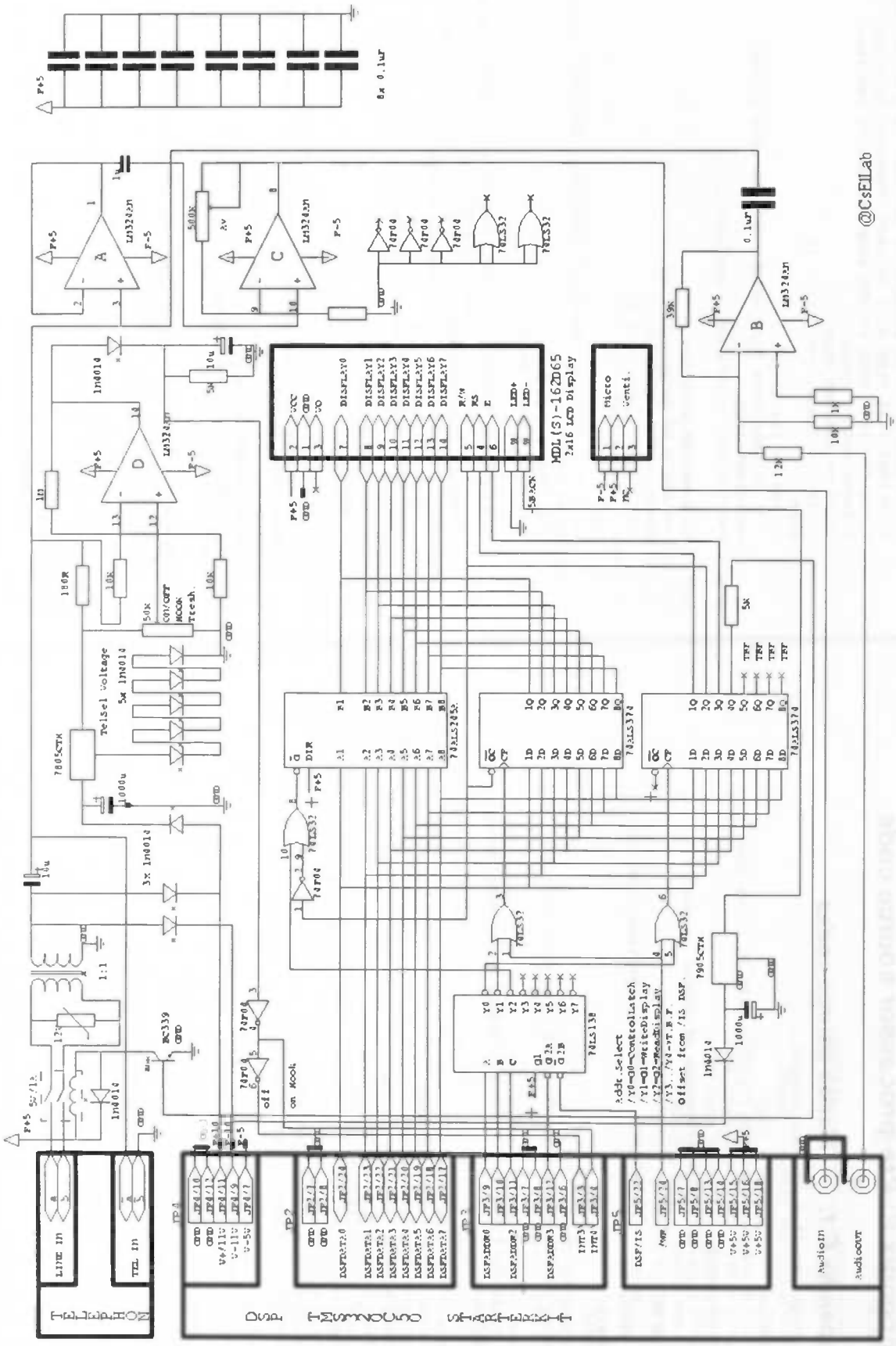


Classify the next two speech signals





Appendix D: Hardware circuitry



Appendix E: Pre-processor source code

Appendix E.1: Matlab source codes

```
function result = spectro (signal);
%
% SYNTAX:
%       S = spectro (signal)
%
% DESCRIPTION:
%       Computes the spectrogram of signal 'signal'.
%
% SEE ALSO:
%       plotspec, spectro2, spectro3, spectro4, spectro5

globals;

H = hamming(FRAME_SIZE);
hamming_window = round (H*255)/255;

signal_length = length (signal);
nof_frames = fix ((signal_length-(FRAME_SIZE-FRAME_RATE)) /
FRAME_RATE)

window = (1:FRAME_SIZE);
for time_frame = 1:nof_frames
   fsig = signal(window);
   fsig = fsig.*hamming_window;
   fsig = fft(fsig, 2*FFT_SIZE);
   fsig = fsig(2:FFT_SIZE + 1);  % forget DC-level and mirror
   fsig = abs(fsig);
   fsig = log10 (fsig);
   result(:,time_frame) = fsig;
   window = window + FRAME_RATE;
end

function result = compress (vector, desired_size)

   factor = length(vector)/desired_size;

   for i = 0:desired_size-1
      % real range of histogram for current x value
      rfirst = factor * i;
      rlast = factor * (i+1);

      % int range of histogram for current x value
      first = fix (rfirst);

      % Solution for evaluating last that corresponds to the
      % C-implementation. The value of last will be the floored
      % value of rlast. If rlast is already fixed, say for example
      % 3.0 than it is no problem to set last to 3 since the weight
      % in that last step will be zero. However, this cannot be done
      % for the last step in the loop because it can result in an
      % index out of range.
      if (i == desired_size - 1)
         last = length (vector) - 1;
      else
         last = fix (rlast);
         % in this case curr_weight will be zero in the
         % last step.
      end

      %Old matlab solution
      %consists of dangerous float zero test !
      %last =  fix (rlast);
      %if (rlast - last == 0.0)
      %   last = last - 1;
      %end

      total_weight = 0.0;
      rbar_height = 0.0;
      for curr = first:last

         if (first == last)
            curr_weight = rlast - rfirst;
         elseif (curr==first)
            curr_weight = 1.0 - rfirst + first;
         elseif (curr==last)
            curr_weight = rlast - last;
         else
            curr_weight = 1;
         end

         total_weight = total_weight + curr_weight;
         rbar_height = rbar_height + curr_weight * vector(curr+1);
      end

      rbar_height = rbar_height / total_weight;
      result(i+1) = rbar_height;
   end

   % Result must have the same type (row or column) as vector.
   if (size (vector, 1) ~= 1)
       result = result';
   end;

function result = spectro2 (signal);
%
% SYNTAX:
%       S = spectro2 (signal)
%
% DESCRIPTION:
```

```
%       Computes the spectrogram of signal 'signal'. The difference
with
%       the function spectro is that the number of frequency bands is
reduced
%       to NOF_FREQ_BANDS.
%
% SEE ALSO:
%       plotspec, spectro, spectro3, spectro4, spectro5
globals;
s = spectro (signal);
nof_frames = size (s, 2);
for time_frame = 1:nof_frames
   fsig = s(:,time_frame);
   fsig = compress (fsig, NOF_FREQ_BANDS);
   result(:,time_frame) = fsig;
end

function result = spectro3 (signal);
%
% SYNTAX:
%       S = spectro3 (signal)
%
% DESCRIPTION:
%       Computes the spectrogram of signal 'signal'. The difference
with
%       the function spectro2 is that cut-off values CUT_SPEC_LO and
%       CUT_SPEC_HI are used to limit the dynamic range and the
values
%       in this range are scaled to a BYTE value (0..255).
%
% SEE ALSO:
%       plotspec, spectro, spectro2, spectro4, spectro5

globals;

s = spectro2 (signal);

nof_frames = size (s, 2);
for time_frame = 1:nof_frames
   fsig = s(:,time_frame);
   fsig = max (fsig, CUT_SPEC_LO);
   fsig = min (fsig, CUT_SPEC_HI);
   fsig = (fsig - CUT_SPEC_LO)/(CUT_SPEC_HI - CUT_SPEC_LO);
   fsig = round (fsig * 255);
   result(:,time_frame) = fsig;
end

function result = spectro4 (signal);
%
% SYNTAX:
%       S = spectro4 (signal)
%
% DESCRIPTION:
%       Computes the spectrogram of signal 'signal'. The difference
with
%       the function spectro3 is that the number of time bands is now
%       rescaled to NOF_TIME_BANDS.
%
% SEE ALSO:
%       plotspec, spectro, spectro2, spectro3, spectro5

globals;

s = spectro3 (signal);

nof_freq_bands = size (s, 1);
for freq_band = 1:nof_freq_bands
    fsig = s(freq_band,:);
    fsig = compress (fsig, NOF_TIME_BANDS);
    fsig = round (fsig);
    result (freq_band, :) = fsig;
end

function S = spectro5 (signal);
%
% SYNTAX:
%       S = spectro5 (signal)
%
% DESCRIPTION:
%       Computes the spectrogram of signal 'signal'. The difference
with
%       the function spectro4 is that values are scaled from min-max
to
%       0-255.
%
% SEE ALSO:
%       plotspec, spectro, spectro1, spectro2, spectro3, spectro4

globals;

S = spectro4 (signal);
min_val = min (S(:));
max_val = max (S(:));

S = ((S - min_val) / (max_val - min_val))*255;
S = round (S);
```

Appendix E.2: C Source code

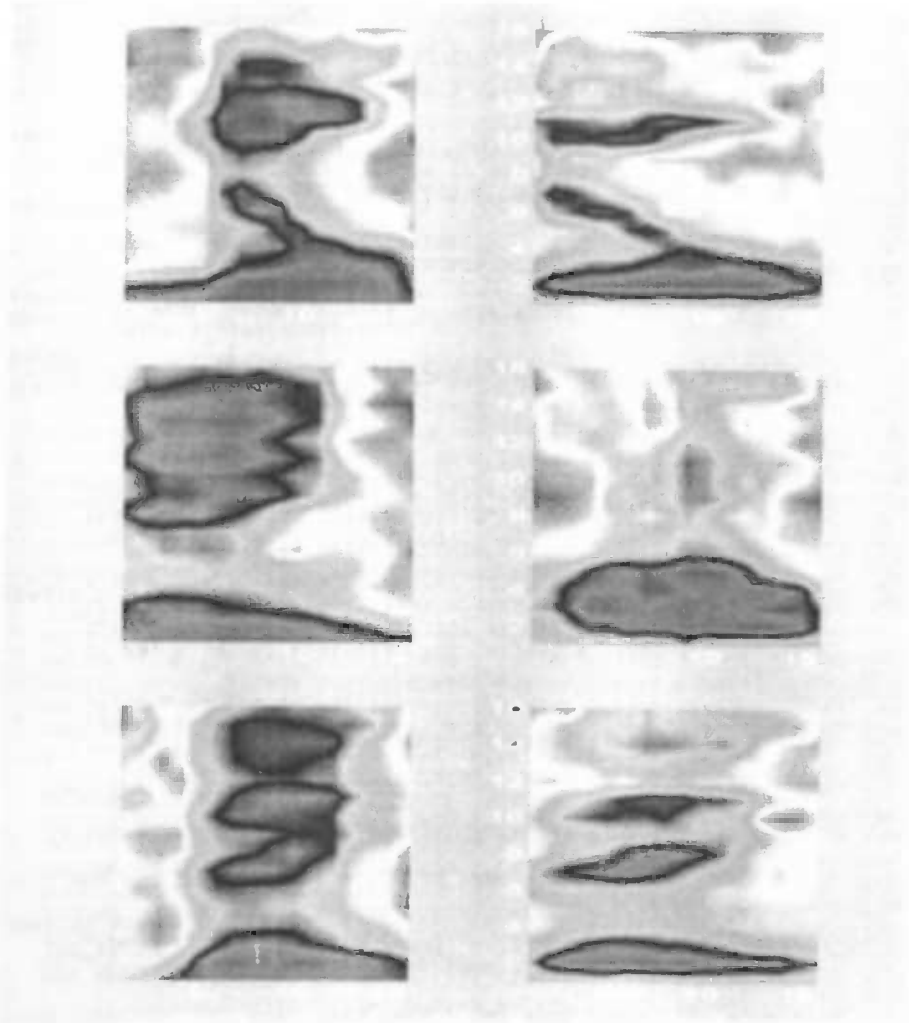
```
void FFT( DSP_WORD pSample )
{
        static float real[ 256 ], imag[ 256 ] ;
        int i,j,k, lel, le,l,ip ;
        float tmp , RSUM , ur, ui, wr, wi ,tr, ti, H;
        static int first_time = 1;
        static int rev[256];

        if (first_time)
        {       int temp_i;
                first_time = 0;
                rev[255] = 255;
                j = 0 ;
                for( i = 0 ; i < 255 ; i++) {
                        rev[i] = j;
                        k = 128 ;
                        while( k <= j ) {
                                j -= k ;
                                k >>= 1 ;
                        }
                        j += k ;
                }
        }

        for( i = 0 ; i < 256 ; i++ ) {
                j = rev[i];
                H = (float) ((j>127)?Hamming[255 -
j]:Hamming[j])/255.0 ;
                real[ i ] = (float) GET_SAMPLE( j + pSample ) * H ;
                imag[ i ] = 0.0 ;
        }

        lel = 1 ;
        for( l = 0 ; l < 8 ; l++ )
        {       le = lel << 1 ;
                ur = (float) 1.0 ;
                ui = (float) 0.0 ;
                wr = cos( PI/lel ) ;
                wi = -sin( PI/lel ) ;
                for( j = 0 ; j < lel ; j++ )
                {       i = j ;
                        while(i < 256 )
                        {       ip = i + lel ;
                                tr = real[ ip ] * ur - imag[ip]*ui;
                                ti = imag[ip] * ur + real[ip]*ui ;
                                real[ip] = real[i] - tr ;
                                imag[ip] = imag[i] - ti ;
                                real[i] += tr ;
                                imag[i] += ti ;
                                i += le ;
                        }
                        tmp = ur *wr - ui * wi ;
                        ui = ui * wr + ur * wi ;
                        ur = tmp ;
                }
                lel <<= 1 ;
        }
        RSUM = (DSP_FLOAT) 0.0 ;
        for( i = 1 ; i < 129 ; i++ )
        {       RSUM += log10( sqrt( real[i] * real[i] + imag[i] *
imag[i] ) ) ;
                if( !(i % (FFT_SIZE/FREQ_BANDS)) ) {
                        RSUM /= (FFT_SIZE/FREQ_BANDS) ;
                        if( RSUM < CUT_SPEC_LO )
                                RSUM = CUT_SPEC_LO ;
                        if( RSUM > CUT_SPEC_HI )
                                RSUM = CUT_SPEC_HI ;
                        RSUM -= CUT_SPEC_LO ;
                        RSUM /= (CUT_SPEC_HI - CUT_SPEC_LO) ;
                        RSUM *= 255 ;
                        time_slice[ (i / 8) - 1 ] = floor(RSUM+0.5) ;
                        RSUM = (DSP_FLOAT) 0.0 ;
                }
        }
        return ;
}
```

```
void Compress( DSP_WORD freq_band , DSP_WORD start, DSP_WORD size )
{
   DSP_WORD i, curr, first, last, tmp, i_spc ;
   DSP_FLOAT factor, rfirst, rlast, curr_weight, total_weight,
rbar_height;

   i_spc = ( FREQ_BANDS - 1 - freq_band ) * TIME_BANDS ;

   factor = ((DSP_FLOAT) (size)/((DSP_FLOAT) TIME_BANDS));

   for( i = 0 ;  i < TIME_BANDS ; i++ )
   {
      /* real range of histogram for current x value */
      rfirst = factor * i;
      rlast = factor * (i+1);

      /* int range of histogram for current x value */
      first = (DSP_WORD) floor ((DSP_FLOAT) rfirst );
      if (i == TIME_BANDS-1) last = size-1;
      else  last = (DSP_WORD) floor ((DSP_FLOAT) rlast );

      total_weight    = (DSP_FLOAT) 0.0;
      rbar_height     = (DSP_FLOAT) 0.0;
      for (curr = first; curr <= last; curr++)
      {
         if (first == last)
            curr_weight = rlast - rfirst;
         else if (curr==first)
            curr_weight = (DSP_FLOAT) 1.0 - (rfirst - (DSP_FLOAT)
first);
         else if (curr==last)
            curr_weight = rlast - (DSP_FLOAT) last;
         else
            curr_weight = (DSP_FLOAT) 1.0;
         total_weight += curr_weight;

               if( freq_band % 2 ) /* odd */
                     rbar_height += curr_weight *
       GET_MSB(spectro_buf[(start+curr) %
       SPECTRO_BUF][freq_band/2]);
               else /* even */
                     rbar_height += curr_weight *
       GET_LSB(spectro_buf[(start+curr) %
       SPECTRO_BUF][freq_band/2]);
      }
      rbar_height = rbar_height / total_weight;
      tmp = (DSP_WORD) floor(rbar_height + 0.5);
      if( tmp > spc_max ) spc_max = tmp ;
      if( tmp < spc_min ) spc_min = tmp;
      PUT_SPECTRO( i_spc , tmp ) ; i_spc++ ;
      return ;
}
```

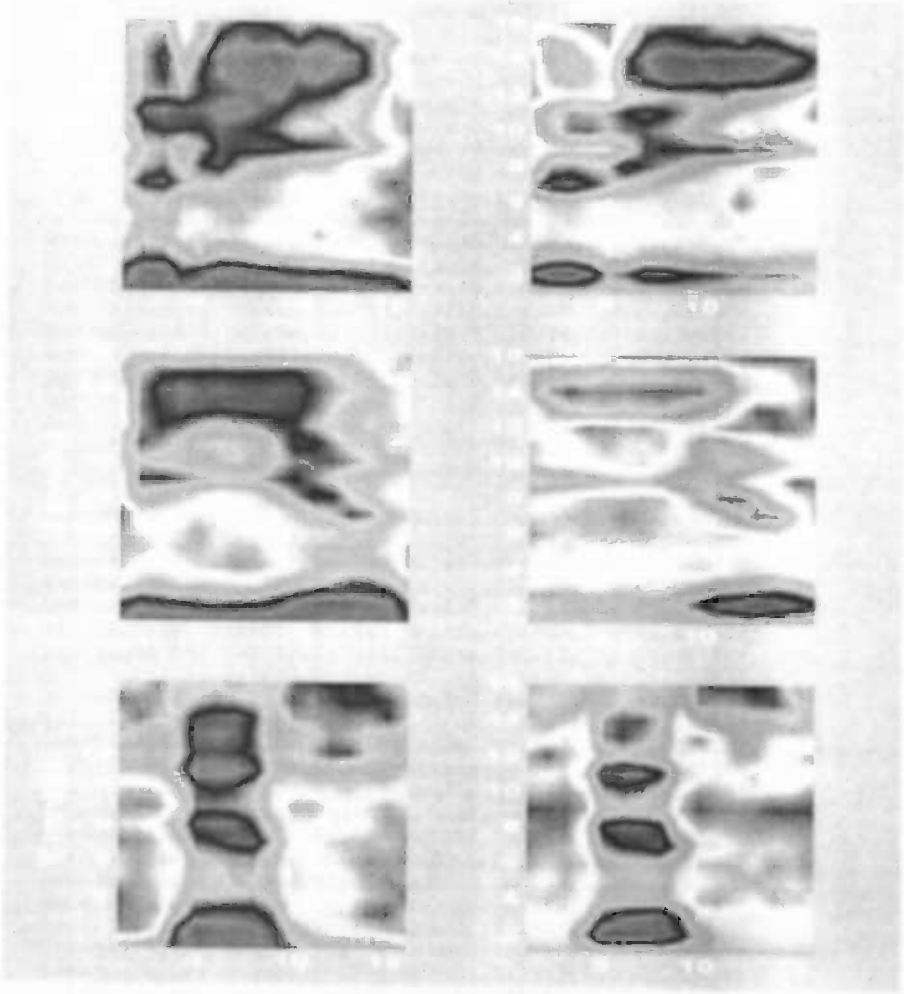
```
void NormSpectro( DSP_WORD start, DSP_WORD size)
{
        DSP_WORD i, tmp ;
        DSP_FLOAT aux ;
        spc_min = 255 ; spc_max = 0 ;

        for( i = 0 ; i < FREQ_BANDS  ; i++ )
                Compress( i , start , size ) ;

        aux = (DSP_FLOAT) 255 / ( spc_max - spc_min ) ;
        for( i = 0 ; i < FREQ_BANDS * TIME_BANDS ; i++ ) {
                        tmp = GET_SPECTRO( i ) ;
                        tmp= (DSP_WORD) floor( (tmp - spc_min) * aux +
.5);
                        PUT_SPECTRO( i, tmp ) ;
        }
        return ;
}
```

Appendix F: Spectrograms Matlab and DSP

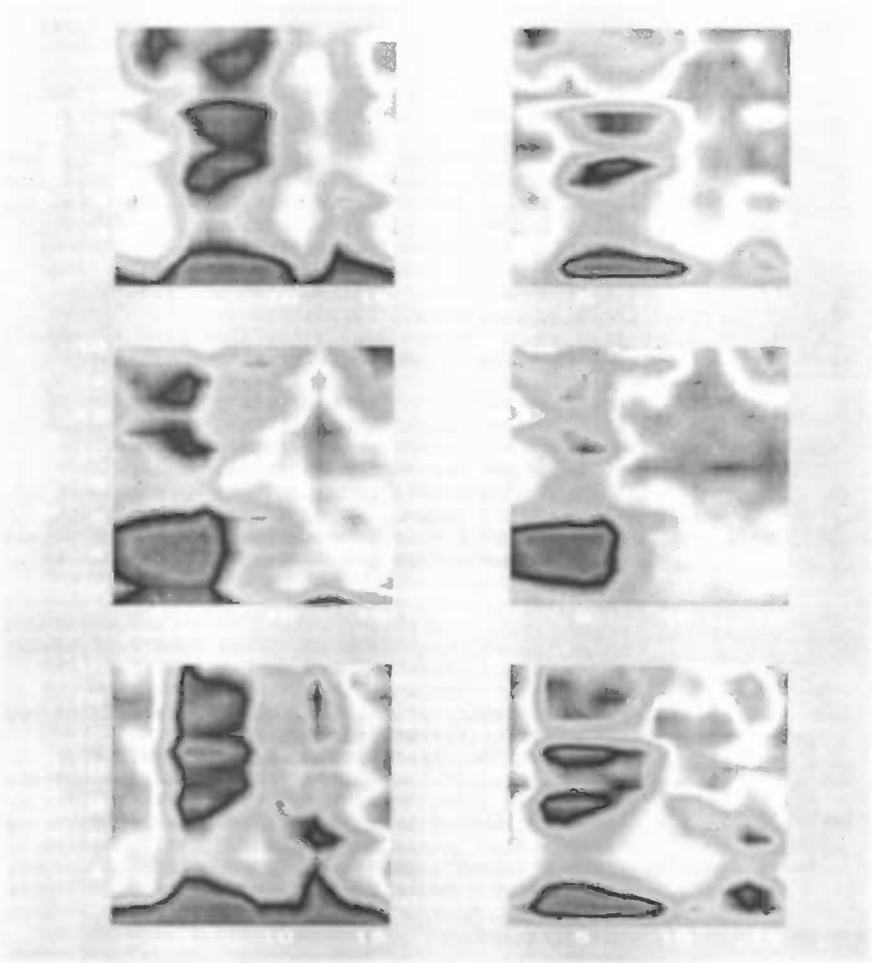
Matlab pre-processed **DSP pre-processed**



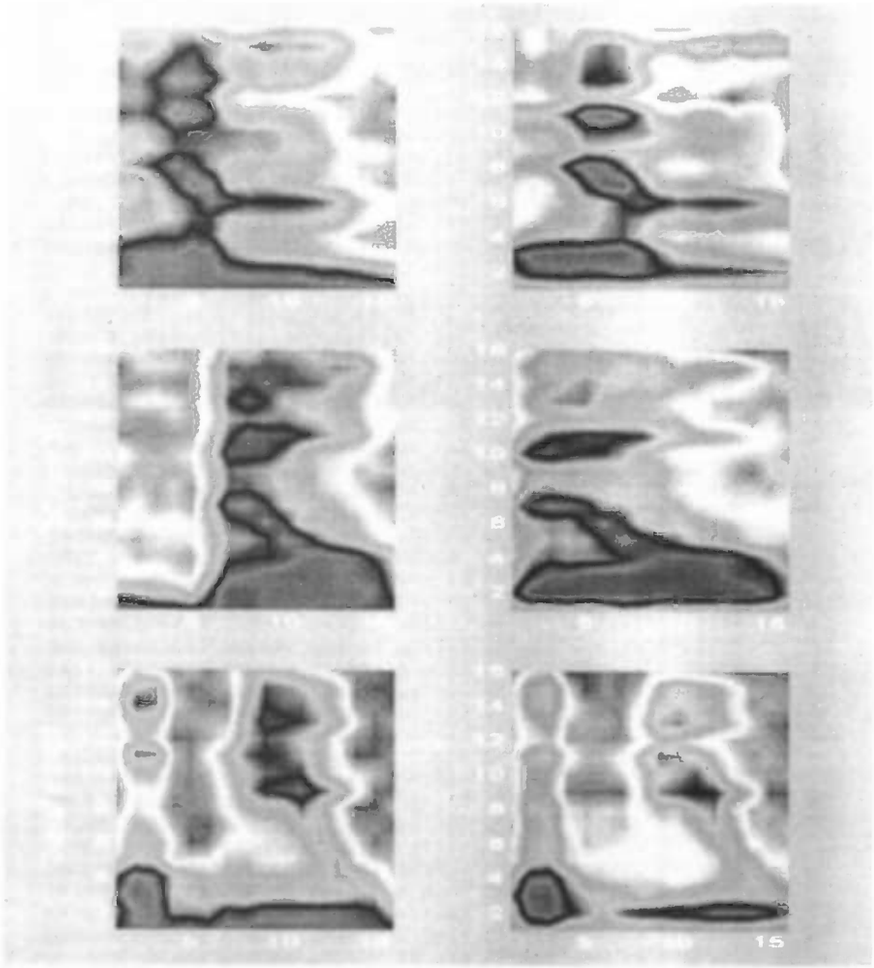
Matlab pre-processed **DSP pre-processed**



Matlab pre-processed **DSP pre-processed**



Matlab pre-processed **DSP pre-processed**



Appendix G: Multilayer Perceptron C code

```
#define NR_INPUT_NEURONS  256
#define NR_HIDDEN_NEURONS 10
#define NR_OUTPUT_NEURONS 13

#define LoB(a)    ((DSP_sword)((a) & 0x00ff))
#define HoB(a)    ((DSP_sword)((a) >> 8 & 0x00ff))
#define LoBS(a)   ((DSP_sword)((a) & 0x0080 ? (a) | 0xff00 : (a) &
0x00ff))
#define HoBS(a)   ((DSP_sword)((a) & 0x8000 ? (a) >> 8 | 0xff00 : (a)
>> 8))
#define Lookup(a) \
   (((a)%2) ? HoB(SIGMOID_TABLE[((a)>>1)]) :\
              LoB(SIGMOID_TABLE[((a)>>1)]))
#define Scale(a,b) ((b) < 0 ? (a) / -(b) : (a) * (b))


typedef unsigned short DSP_word;
typedef short          DSP_sword;
typedef long           DSP_long;

/* Lookup table for neuron transfer function */
static DSP_word SIGMOID_TABLE[64] = {
   0x7f81,0x8486,0x898b,0x8e90,0x9395,0x989a,0x9c9f,0xa1a3,
   0xa6a8,0xaaac,0xaeb1,0xb3b5,0xb7b9,0xbbbd,0xbfc0,0xc2c4,
   0xc6c7,0xc9cb,0xccce,0xcfd1,0xd2d4,0xd5d6,0xd8d9,0xdadb,
   0xddde,0xdfe0,0xe1e2,0xe3e4,0xe5e6,0xe7e7,0xe8e9,0xeaea,
   0xebec,0xeded,0xeeee,0xeff0,0xf0f1,0xf1f2,0xf2f2,0xf3f3,
   0xf4f4,0xf5f5,0xf5f6,0xf6f6,0xf7f7,0xf7f7,0xf8f8,0xf8f8,
   0xf9f9,0xf9f9,0xf9fa,0xfafa,0xfafa,0xfbfb,0xfbfb,0xfbfb,
   0xfbfb,0xfcfc,0xfcfc,0xfcfc,0xfcfc,0xfcfc,0xfdfd,0xfdfd};
static DSP_word w1[1280] = {
   0x0017,0x03f4,0xec00,0xf60f,0x1009,0x101a,0x1721,0x545e,
   0x040b,0xf800,0xeeeb,0x0804,0xf5fd,0xfe0c,0x1214,0x3251,
   0x121e,0x0f16,0xecf6,0x07f6,0xe4d9,0xd1ef,0xf5fb,0x1d34,
   0x0521,0x1a0c,0xd9d6,0xdfd1,0xd9e5,0xf2f6,0xfcf9,0x1427,
   0xf700,0xf1f1,0xdbec,0x01ff,0xff00,0xf507,0x0300,0x1e2c,
   0xdfd6,0xe6fa,0x0d24,0x2d18,0x0914,0x1a13,0x0c06,0x1022,
   0xf7d9,0xe8f9,0x01fd,0x170e,0xf5fa,0xf9f7,0xebdf,0x010d,
   0x05f5,0xdddc,0xcfcc,0xc8d4,0xded6,0xdfd9,0xc8d2,0xe2e9,
   0x0b1b,0x1f18,0x0ffe,0xfff1,0xfbfd,0x0b0c,0xfdf7,0x0a0a,
   0x232c,0x2411,0x1447,0x4139,0x301f,0x241f,0x2a27,0x1f1f,
   0x1d10,0xfedc,0xd8e7,0xfe10,0x1bf6,0xebe6,0xe0f5,0xfff9,
   0x120d,0x110d,0xf8f4,0xf712,0x1716,0x1d07,0xe4f7,0x0d0e,
   0x0af2,0xf8fc,0xff00,0x0313,0x0914,0x1d1f,0x1b23,0x2023,
   0xfbef,0x0a2e,0x3d4e,0x6684,0x8167,0x5731,0x0e23,0x2823,
   0xf8c6,0xe0df,0xedf7,0xdcc1,0xa79c,0x9599,0xa9cd,0xe0fb,
   0x19ec,0x01f7,0x051a,0x09ed,0xe6e0,0xd6cb,0xcee4,0xe8f9,
   0x5e38,0x02f7,0xfde4,0xe9e1,0xf2f4,0x0703,0x1212,0x0404,
   0x6b44,0x140d,0x07f1,0xf3e3,0xdee9,0xf4f1,0xfbfb,0x00f3,
   0x4b45,0x441f,0x07e7,0xe5f6,0xf2f8,0xf0e3,0xfc03,0x09fb,
   0x2215,0x280a,0xf8f2,0x0c0f,0x00f9,0xe6ed,0x0006,0xfc00,
   0x1a10,0x1af3,0xfae1,0xeef9,0xfaf5,0xe6e0,0xedee,0xfbe7,
   0x0bf9,0xf6e9,0xe3e0,0xf2ea,0xe30b,0x190e,0x1d10,0x0904,
   0x0af4,0x02fd,0x0a07,0x1915,0x1233,0x2722,0x1911,0x00fd,
   0x04ef,0x060b,0x0bfe,0x070c,0x0c18,0x171c,0x1605,0x05ff,
   0x5836,0x3218,0x1913,0x1a15,0x0307,0x0209,0x2214,0xf4f3,
   0x6036,0x1013,0x3b34,0x2c2a,0x07f9,0xefea,0x0c05,0xfbf5,
   0xdbc9,0xd0d5,0xebf7,0xfb09,0x0ef6,0xfff9,0xff0a,0x0006,
   0xd8cf,0xdde5,0xe7f6,0x00fb,0xf9f1,0xfc04,0xfbf9,0xf907,
   0xfad6,0xd1e1,0xe2df,0xe5dc,0xe5d9,0xe9e4,0xfcf6,0xe3f9,
   0x19f9,0xd3d8,0xf508,0xfef0,0xf1d9,0xd5d5,0xe4e0,0xcfe4,
   0xe5f5,0x2035,0x3325,0x2525,0x2e24,0x2119,0x1af7,0xd2ee,
   0x7eb2,0xe1f4,0x0107,0x070d,0x1e0b,0x080f,0x15fc,0xd0cf,
   0xa4d6,0x0e12,0xe8db,0xedf3,0xe7fd,0xdcdf,0xf40e,0x1f0e,
   0xaaed,0x211f,0x03fa,0xf4fa,0x08fd,0x1003,0x1728,0x1c1f,
   0xabcf,0xf301,0xfb20,0x0ff9,0x080e,0x0cf6,0xf805,0x0013,
   0xc0c6,0xd3ef,0xdae9,0xdddd,0xeef2,0xe7d6,0xd9f2,0xeff1,
   0xee08,0x1619,0x00fc,0xe6e6,0xdfd3,0xd8d7,0xecf3,0x0a01,
   0x1627,0x1e10,0x01f9,0x0421,0x1c0c,0x0d08,0x1627,0x1f1d,
   0x1500,0x00f3,0xdce6,0x0e27,0x0bf6,0xebfb,0x0f0f,0x200b,
   0xe5f8,0xd8e8,0xe40b,0x0706,0xf1f8,0xe9b9,0xecfe,0x0505,
   0xd2eb,0x0428,0x0b0e,0x3027,0x2917,0x08fa,0x1124,0x4242,
   0xb7c6,0xf1d0,0xd407,0x3651,0x5e6c,0x5d25,0x3036,0x494c,
   0x0007,0x00cf,0xe327,0x4244,0x3c59,0x3a16,0x252a,0x2929,
   0x1f23,0x070c,0x000a,0xeced,0xd902,0x1420,0x2f41,0x1e06,
   0x0b4c,0x5150,0x4a4c,0x1c04,0xd1c1,0xc1ac,0xb5c5,0xe4cd,
   0x2e64,0x627f,0x6f75,0x3906,0xcedf,0xe7eb,0xcabf,0xf0d6,
   0x30f3,0xb484,0x9cb6,0xb0ba,0xa8b0,0xb2c9,0xe807,0x19d6,
   0x4019,0xe0f3,0xf9f1,0xebf2,0xf50c,0x1709,0x182c,0x390b,
   0xbcec,0x0208,0x0b1e,0x27fe,0x0a13,0x0500,0xefde,0xcfda,
   0xbeee,0x0dfb,0xfc1e,0x351a,0x2528,0x07f3,0xcad2,0xd3f1,
   0xaee6,0xf7f2,0x1025,0x1f20,0x2807,0xf3d4,0xd3c5,0xd1eb,
   0xadba,0xdad0,0x0920,0x332e,0x341c,0x0af2,0xe5e0,0xefee,
   0xc3ed,0x0103,0x1339,0x5042,0x4227,0x2903,0xe2d0,0xd8ea,
   0x001b,0x302e,0x101f,0x191c,0x1ffc,0xe9db,0xe5ed,0xf2f8,
   0x5861,0x4742,0x2d19,0x03fa,0xfcf2,0x07f8,0x2628,0x1f0f,
   0x344c,0x434e,0x3836,0x3615,0xfce5,0xeefb,0x2a2d,0x2717,
   0x1835,0x2112,0x06fd,0xf8eb,0xe6e8,0xe7fa,0x1a15,0x0816,
   0x17f9,0xc3a7,0x9492,0x7176,0x9ec6,0xdaff,0x232b,0x3435,
   0xf9f4,0xd6d1,0xc2bc,0xa994,0xc3e3,0xf401,0x2018,0x1320,
   0xd0f2,0xe2d2,0xf1d4,0xdddf,0xe8ed,0xdfe1,0xf0e5,0xf504,
   0x8fad,0xd4db,0xe5db,0xe7e7,0x0922,0x2b26,0x1c05,0x001c,
   0x9cc5,0xcecd,0xbdbc,0xd1d7,0x0114,0x301f,0x2cfe,0xdd00,
   0xef10,0x153d,0x2b23,0x1108,0xff08,0x1e2a,0x2516,0xfd1b,
   0x2240,0x3332,0x2101,0xfffd,0xfdef,0xfe0c,0x03fb,0xecf0,
   0xfe01,0x0f1c,0x09ee,0x00f2,0x0103,0x0434,0x2531,0x15f8,
   0x0d10,0x2619,0xf5ee,0x07ed,0x05fd,0xf318,0x0a0f,0xfb12,
   0x04f8,0x1623,0x1400,0x1a10,0x150c,0x1b14,0x051a,0xfbfc,
   0xdee7,0xf723,0x2e25,0x2220,0x1b17,0x0817,0x1211,0x051f,
   0xeafa,0x122a,0x2203,0x120c,0xfc0a,0x1615,0x0beb,0xe5e5,
   0x1930,0x2a02,0xd1d1,0xcfd9,0xf8fe,0xf0f2,0xd1c0,0xf0e5,
   0x2e42,0x2c02,0xe3cb,0xe1ea,0xf2d9,0xd2cf,0xb6a6,0xcdec,
```

```
0x172e,0x270d,0xfcd6,0xe3e0,0xe3dc,0xd5d8,0xc3b1,0xcaf2,
0x0503,0xf7d3,0xdbe5,0xfee6,0xcdcd,0xd2c2,0xc6b9,0xccf5,
0x05d9,0xc8b1,0xd3b6,0xd4dc,0xcfb8,0xd1f9,0x050b,0x1c2d,
0x0ae9,0xe1ef,0x0025,0x3747,0x3a2e,0x1532,0x401f,0x2a41,
0x2812,0x1611,0x4b6d,0x7279,0x8464,0x3428,0x2f15,0x1538,
0x000c,0x1c0f,0x2142,0x4632,0x4c3b,0x2935,0x2b1e,0x2632,
0xd5b4,0xce9a,0xa7a3,0xa596,0xb0d0,0xc4f1,0x0d2b,0x202c,
0xf1f6,0x2df8,0xdcde,0x00fc,0x1712,0xf300,0xf4eb,0xf114,
0x0021,0x4945,0x3024,0x08e5,0xf4ed,0xd3c9,0xb9a1,0xdaea,
0x4512,0xedf7,0xfa2c,0x2630,0x2413,0x04d1,0xd7cc,0xb2a9,
0x762a,0xecee,0xf107,0xdefc,0xf9d7,0xc4b1,0xafc8,0xb6aa,
0x6c48,0x0be8,0xfcf2,0xda07,0xfbf8,0x0ef2,0xf113,0xeccd,
0x4830,0x02fd,0x0608,0xff25,0x1d12,0x0417,0x0d24,0xfede,
0x2c02,0xf9ec,0xd9d3,0xd90f,0x06d7,0xdfde,0xdce8,0xead0,
0x1f0c,0xe2eb,0xdfec,0xfa1e,0xfde4,0xeeeb,0xe8ee,0x00ff,
0xe4da,0xe4ff,0x1f26,0xfee6,0x00f2,0x06ef,0xea06,0x0f03,
0xe5cf,0x050d,0xeff9,0xd0e6,0xe2f7,0x2a1a,0x040a,0x0615,
0xf8dc,0xcdb4,0xabc6,0xaede,0xeee4,0x01f9,0xf512,0xfb00,
0x0bf7,0xf3e2,0x0a20,0x1aeb,0x9c93,0xb0b8,0xadc7,0xc1ae,
0x0bff,0x2533,0x4553,0x4f35,0xfce6,0x0905,0xf8ea,0xf3c7,
0x20fe,0x1925,0x3c5d,0x4a4c,0x3e20,0x1e16,0xfff4,0xf0d5,
0x4933,0x1004,0x1f5a,0x6b55,0x4d36,0x374d,0x261d,0x1dff,
0x8068,0x18f9,0x1e63,0x4d58,0x3e1f,0x3d39,0x0c36,0x4b1a,
0x22c9,0xaab8,0xb2de,0xc4db,0xec08,0x3048,0x1833,0x6441,
0xafa9,0xccbb,0xb0d3,0xd5d5,0xdade,0xfa03,0xf802,0x112e,
0xdb09,0x301c,0x15ed,0xebed,0x0001,0x0b07,0x001b,0x0b23,
0xb8f5,0x16f9,0xf0d4,0xebfb,0x0f13,0x1115,0x031b,0x0315,
0xccde,0xe9ed,0xf6e8,0xfaf0,0x0f16,0x1111,0x0a1e,0x1711,
0xfcfb,0xfa01,0x1500,0x030b,0x151b,0x1e0f,0x0d18,0x0e11,
0x0616,0x181a,0x1806,0x0504,0x172b,0x1d13,0x1514,0x0014,
0xfcff,0x100a,0xfef2,0xe4e4,0xf904,0x00f3,0xebf0,0xeffb,
0xe9e1,0xc8d5,0xe1de,0xf2fd,0x131b,0x1d0f,0xf1fc,0xf507,
0x1112,0xe9e2,0x020a,0x121e,0x2016,0x0108,0x0606,0x0b0a,
0x0b0e,0xf2eb,0xf2ec,0xe6d2,0xd0da,0xdef2,0xf1ef,0xf9f6,
0xdce8,0xddf5,0xf5dc,0xd1d8,0xe4e9,0xdadf,0xd3d9,0xf50c,
0x0907,0xfa13,0x1d0a,0x1d0a,0xfefa,0xf1ec,0xd6d9,0xfe0d,
0x000f,0x060b,0x0ff2,0x0f0d,0x0dfb,0xf1f4,0xf1eb,0x071e,
0x1825,0x1e03,0xf5de,0xf3f4,0x0803,0xf9f5,0xfa11,0x0e15,
0x060b,0xf0bb,0xa876,0x98a8,0xc9e5,0xe4f9,0xfa06,0x141a,
0xef0b,0x02f5,0xfef0,0x0c03,0x0d10,0x0304,0x0005,0xf403,
0x2322,0x1218,0x2b17,0x2e2e,0x2e30,0x2a1a,0x1e13,0x0bff,
0x3005,0xf1fe,0xf300,0x0416,0x0afd,0x0407,0x070a,0x0702,
0x0ce8,0xeafd,0xe9f7,0x080a,0xffeb,0xfc04,0x1b10,0x0b07,
0x00dd,0xf6fd,0xf2f0,0xf905,0xedea,0x0203,0x1405,0x0708,
0x11f5,0x0714,0x0102,0xfcff,0xf3f1,0xff07,0x150d,0x0904,
0x06f6,0x0412,0x0506,0xff02,0x02ff,0x0c10,0x291f,0x0d09,
0x00fc,0x030e,0x040a,0x0d06,0xfff8,0xf7fe,0x1208,0x02fe,
0xf3f2,0x0406,0x0004,0x0a06,0xffe5,0xebf0,0xf9f8,0x01fc,
0xf3ea,0xf7fe,0x0106,0x110f,0xf9ef,0xf8f3,0xf5f3,0xf503,
0xded1,0xd7f3,0xecfe,0x0f1c,0x1705,0x1000,0xfffc,0xff07,
0x010b,0x2523,0x0dfe,0x0e18,0x271e,0x231a,0x1404,0xfef6,
0x1818,0x1e21,0x10fe,0xff04,0xfffd,0x0d0a,0x04f3,0xfafa,
0x1001,0x0207,0xfcf8,0xfaef,0xe7e5,0xf0f9,0xf2ea,0xe9f1,
0xcbcf,0xe5f1,0xf2e9,0xf1f8,0xf5f6,0xfb04,0x0af3,0xf3f4,
0x8184,0xbada,0xe2e6,0xecf1,0xedf3,0x0919,0x1c0b,0x0600,
0xf501,0x1c27,0x2421,0x2319,0x0bf2,0xfa02,0x0a04,0x080a,
0x362c,0x2119,0x130b,0x0708,0xfdeb,0xeaea,0xf4f9,0x050c,
0x0503,0xf8ed,0x0006,0xfaf7,0x0af9,0xf3e2,0xe9ea,0xe5db,
0xf90f,0xe9eb,0x1213,0x1d17,0x07fa,0xc3c4,0xc9c9,0xe3ed,
0x3111,0xf6ed,0x0a11,0x2525,0x0a02,0xedeb,0xfbe6,0xdae8,
0xff12,0xf1c7,0xf206,0x4746,0x3223,0x0af3,0xfce3,0xf1fc,
0xf221,0x12e3,0x0c20,0x3e49,0x1721,0x18fc,0xe6c9,0xcbc5,
0xe2fb,0xe4bf,0xf8de,0xf60c,0xed11,0xeaea,0xd7b3,0xbcda,
0xfaf4,0xc4b4,0xe6de,0x040b,0xfa05,0xfc11,0xfcd6,0xd4e7,
0x0c28,0x14f6,0x140b,0x2e3f,0x2d26,0x1b1b,0x08c2,0xd2f9,
0x0020,0xfeeb,0xf8c5,0xfc3a,0x3237,0x2f2e,0x3138,0x2d08,
0xfff9,0xcaad,0xbc9d,0xb607,0x1721,0xf03d,0x6f7d,0x7f79,
0xdad1,0xa0b0,0xad89,0x99af,0xc2d7,0xb9e6,0x0010,0x5059,
0x050c,0xffe8,0xdaa6,0xcece,0xe2f5,0xe1f4,0x0221,0x3126,
0x8b79,0x35f2,0xe7c4,0xc9cd,0xfb23,0x2718,0x292b,0x4e30,
0x4233,0xface,0xdc96,0xb1b2,0xe409,0x1911,0x2538,0x5928,
0xf9e0,0xddc8,0xe1a6,0xa5c9,0xed2e,0x3a2c,0x2921,0x160a,
0xe7fd,0xf7f9,0x3b5d,0x3d1c,0x272b,0x1805,0xccd9,0xcbdd,
0x11c4,0xa6ba,0xebf6,0xfce7,0xe7f9,0xeee7,0xfad7,0xdeda,
0xf5e0,0xc2dc,0xfa11,0x1d20,0x2121,0x120d,0x08ff,0xf8db,
0xfe0f,0xe7f2,0x10fd,0x02f5,0x0a13,0xfb0a,0x1306,0xfefa,
0xebfc,0xeefe,0xeee2,0xedef,0xfafe,0xf409,0x0a1b,0x07ed,
0xe3e1,0xdd00,0xfcfa,0x192a,0x2740,0x2d26,0x2521,0xfa00,
0xdadf,0xe8fe,0xfffc,0x1311,0x0f11,0x1b14,0x07f2,0xe3ed,
0xe6f1,0x0501,0xf3f1,0xff02,0x0825,0x1b23,0x1107,0xfaf4,
0x1c0a,0x140f,0xfdfd,0x060a,0x0009,0x0622,0x302f,0x1907,
0x210f,0xeff3,0xeae5,0xf812,0x000a,0x0a19,0x2c3d,0x2404,
0x0e19,0x210b,0xf2fa,0x0308,0xeef7,0xceb2,0xb8ee,0xf8ee,
0x463d,0x484b,0x351d,0x13f5,0xbfbb,0x9f7a,0x81af,0xdaf4,
0x371c,0x122a,0xf6e9,0xebec,0xcbd0,0xa89e,0x95b3,0xe2dc,
0x1e05,0x0107,0x00f9,0x1424,0x110c,0x00f7,0xdd07,0x1614,
0x1e00,0xfa00,0xeef9,0x1a39,0x272f,0x1a14,0xe908,0x2201,
0x1810,0x0700,0xcec4,0xe609,0x142f,0x3234,0x2b54,0x2f08,
0x150b,0xfcfb,0xe3e6,0xe4e1,0xd7dd,0xe903,0x010d,0x05f3};
static DSP_word w2[65] = {
    0xa7aa,0xd618,0x4737,0x1235,0xc5fc,0xa329,0xec7c,0xed09,
    0xf47a,0x9d18,0xdd20,0xe2e2,0xd30f,0xc72b,0xdc4b,0xb3f0,
    0xe1eb,0xb8cc,0x4c3e,0x984a,0xc30c,0x0029,0x42a3,0x0117,
    0x6a04,0x14ce,0x4ad0,0xa631,0xc601,0xf236,0x9a1e,0xfed9,
    0xef79,0x001d,0x0bdb,0xbc4b,0xf306,0xd91f,0xfa5e,0xc898,
    0x12a6,0x60b9,0x5604,0x10df,0xdbe0,0xaa74,0x3731,0xcab9,
    0xd629,0xa3b7,0xd17b,0x3a81,0xed15,0x1384,0xbd9c,0x62a2,
    0x74f6,0xfd43,0xe2f0,0xf303,0xc5b0,0xc8d4,0xcd91,0x1fb7,
    0x69e4};
static DSP_sword b1[10] = {
    13891,-8445,-1507,-5373,2861,-14686,8055,13182,
    20107,-1743};
static DSP_sword b2[13] = {
    29735,-25588,32689,30721,30286,30901,-26753,29306,
    31772,-7513,-21288,-19793,30122};
static DSP_sword s1[10] = {
```

```
      9,    9,    6,    6,    5,    5,    9,   12,
      5,    7};
static DSP_sword s2[13] = {
      7,    5,   10,    6,    6,    7,    4,    8,
      6,    5,    5,    6,    4};

extern DSP_word in[NR_INPUT_NEURONS];
extern DSP_word out[NR_OUTPUT_NEURONS];
static DSP_word hid[NR_HIDDEN_NEURONS];

extern float c_sum[];
extern float c_bias[];
extern float c_out[];

DSP_word Sigmoid(DSP_long sum)
{
   if(sum > 127) return(255);
   if(sum < -127) return(0);
   if(sum < 0) return(255 - Lookup(-sum));
   else return(Lookup(sum));
}

void InterActNet(void)
{

   DSP_word *i, *w, *n;
   DSP_long sum;
   DSP_sword *b, *s;
   float fsum;

   int j = 0;

   i = in;
   w = w1;
   b = b1;
   s = s1;
   n = hid;
   while(n < hid + NR_HIDDEN_NEURONS) {
      sum = *b;
      c_bias[j] = (float)*s * (float)sum / (255.0 * 127.0) ;
      fsum = c_bias[j];

      i = in;
      while(i < in + NR_INPUT_NEURONS/2) {
         sum += LoB(*i)*LoBS(*w) + HoB(*i)*HoBS(*w);
         fsum +=
            ((float)LoB(*i) / 255.0) *
            ((float)*s * (float)LoBS(*w) / 127.0) +
            ((float)HoB(*i) / 255.0) *
            ((float)*s * (float)HoBS(*w) / 127.0);

         i++;
         w++;
      }
      c_sum[j] = (float) *s * (float)sum / (255.0 * 127.0);

      Scale(sum, *s) / 51;
      *n = Sigmoid(sum);

      c_out[j] = *n;
      j++;

      n++, b++, s++;
   }
   n = out;
   w = w2;
   b = b2;
   s = s2;
   while(n < out + NR_OUTPUT_NEURONS) {
      sum = *b;

      c_bias[j] = (float)*s * (float)sum / (255.0 * 127.0) ;

      i = hid;
      while(i < (DSP_word *)(hid + NR_HIDDEN_NEURONS)) {
         sum += *i++ * HoBS(*w);
         sum += *i++ * LoBS(*w);
         w++;
      }
      c_sum[j] = (float) *s * (float)sum / (255.0 * 127.0);

      Scale(sum, *s) / 51;
      *n = Sigmoid(sum);

      c_out[j] = *n;
      j++;

      n++, b++, s++;
   }
}
```

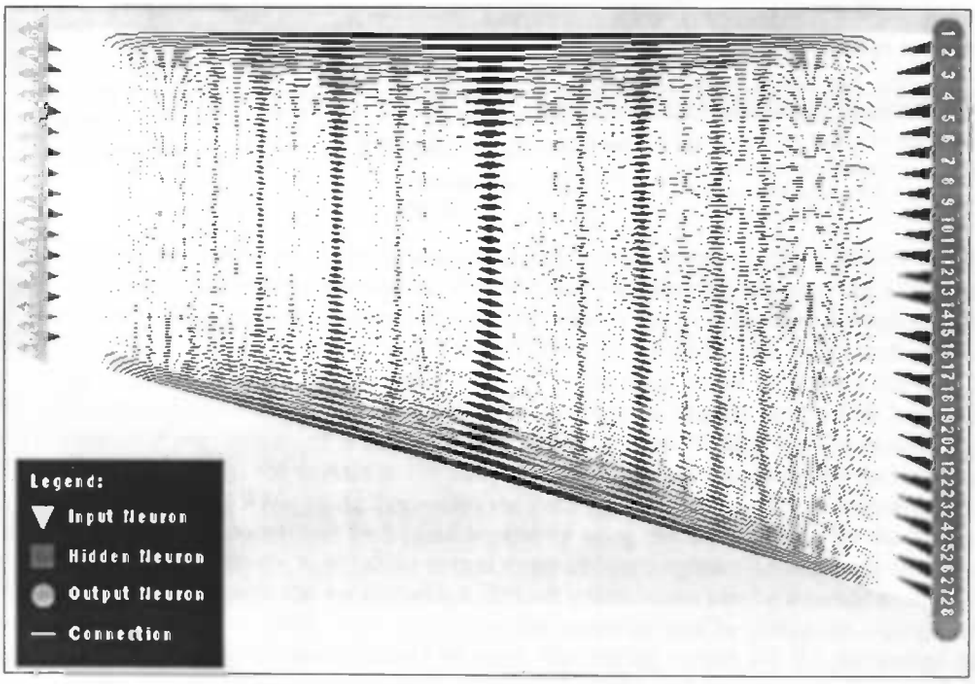
Appendix H: Training and Testing of the Kohonen Network

Theory

The Kohonen network or Self Organising Feature Map (SOFM) can be used to find a set of vectors, called a codebook, that splits the domain into a number of classes. Each class is represented by one or more codebook vectors. The training of a SOFM is unsupervised. There are two commonly used techniques for training a SOFM. The first method uses a technique called lateral inhibition, where input vectors belong to a certain class, are modified, depending on the learning rate, in the direction of the input vector, the adjacent classes are modified so that the distance between the vectors representing theses classes and the input vector is increased. This method is inspired biologically by the human optical system which is bases on lateral inhibition too. The other training method is the opposite of the first. Instead of increasing the distance between the input vector and adjacent classes, the distance is decreased too. The range of influence of the input vector on adjacent classes is determined by a learning parameter called neighbourhood, when the neighbourhood is set to the number of codebook vectors all the vectors will be equal. The classification of an input vector, is nothing more than finding the closest vector in the codebook using the Euclidean distance measure. For a more detailed description of the SOFM see [NNSH]. Each class is given a label, so that the n-dimensional input is mapped to a 1-dimensional output (symbol). This is called vector quantisation.

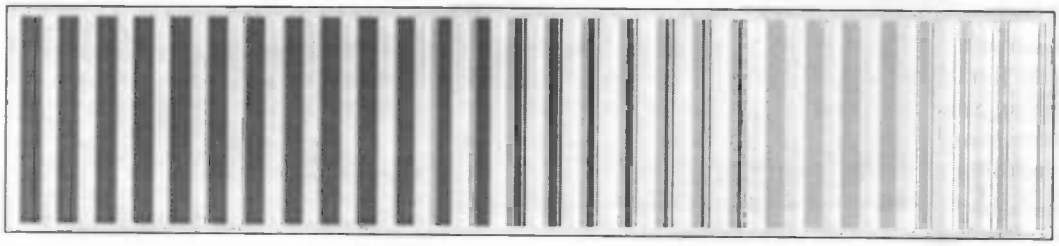
Network implementation

The Kohonen Network was implemented in Inter*Act*. There is a complete package available for learning vector quantisation (LVQ) called LVQ_PAK for free at *cochlea.hut.fi*. We trained and tested several networks with different codebook sizes. The number of input neurones in a Kohonen Network corresponds to the dimension of the input vectors, the number of output neurones to the number of codebook vectors. We did not use a hidden layer though this is a possibility. Without the hidden layer the weight on the connections between input and output layer are simply the entries of the codebook vectors, so that all connection for the input layer to one output vector form exactly one codebook vector. The output neurones are fully connected, where all the weights on the connections are fixed, the distances between adjacent output neurones are equal. The quality of a codebook was measured by the average distance between the input vectors and the codebook vectors they were mapped to. The Kohonen network with 28 vectors is shown in the figure below:



Training

The procedure followed for training the Kohonen Network corresponds to that described in the OCR[1] example included in the Inter*Act* on-line help. First all the codebook vectors were trained with the neighbourhood set to the codebook size, so that all the codebook vectors were approximately equal to the mean of all input vectors. Thereafter the neighbourhood was decreased slowly, thus the codebook vectors started to differentiate. After the neighbourhood had been decreased to a certain value training continued with the learning rate slowly decreasing, thus all codebook vector were forced to specialise a bit further without taking the risk of loosing to much information already stored in the codebook. With 16-dimensional input vectors the codebook vectors are not as easily visualised as the OCR codebook vectors. We represented the codebook vectors by a colour plot, with each line in the plot of a vector corresponding to one value in the codebook vector, a codebook containing 24 vectors is shown in the figure below:



Results

We trained networks with 24,28,32,48 and 64 codebook vectors. Using the average Euclidean distance error measure the following results were obtained:

#vectors	#error
24	0.367
28	0.366
30	0.347
32	0.355
48	0.343
64	0.331

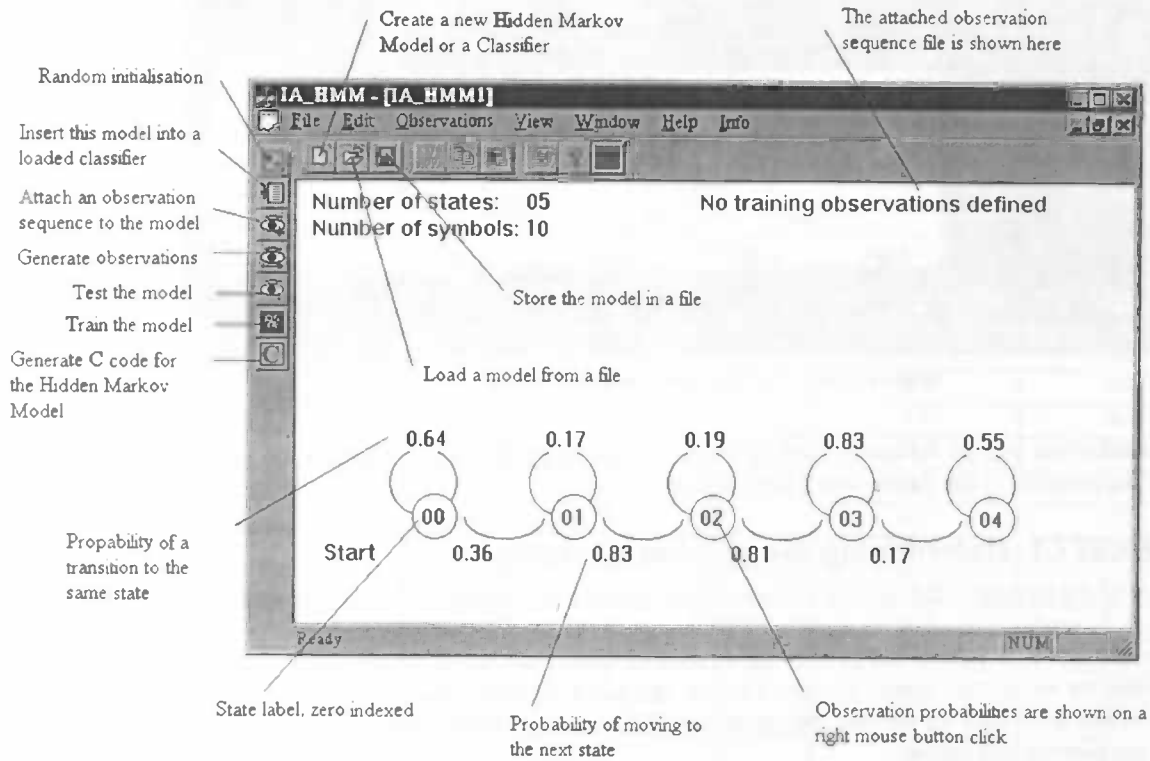
There are several fancy ways to improve the performance of a Kohonen network, though higher dimensional problems, like ours, are harder to solve using Kohonen because it is not easy getting a grip on the training algorithm. Furthermore one should realise that the quality of the input vectors and the relative frequency of certain vectors are of great influence on the performance. In Chapter 8 some of these problems are discussed.

[1] Optical Character Recognition

Appendix I: Using IaHMM

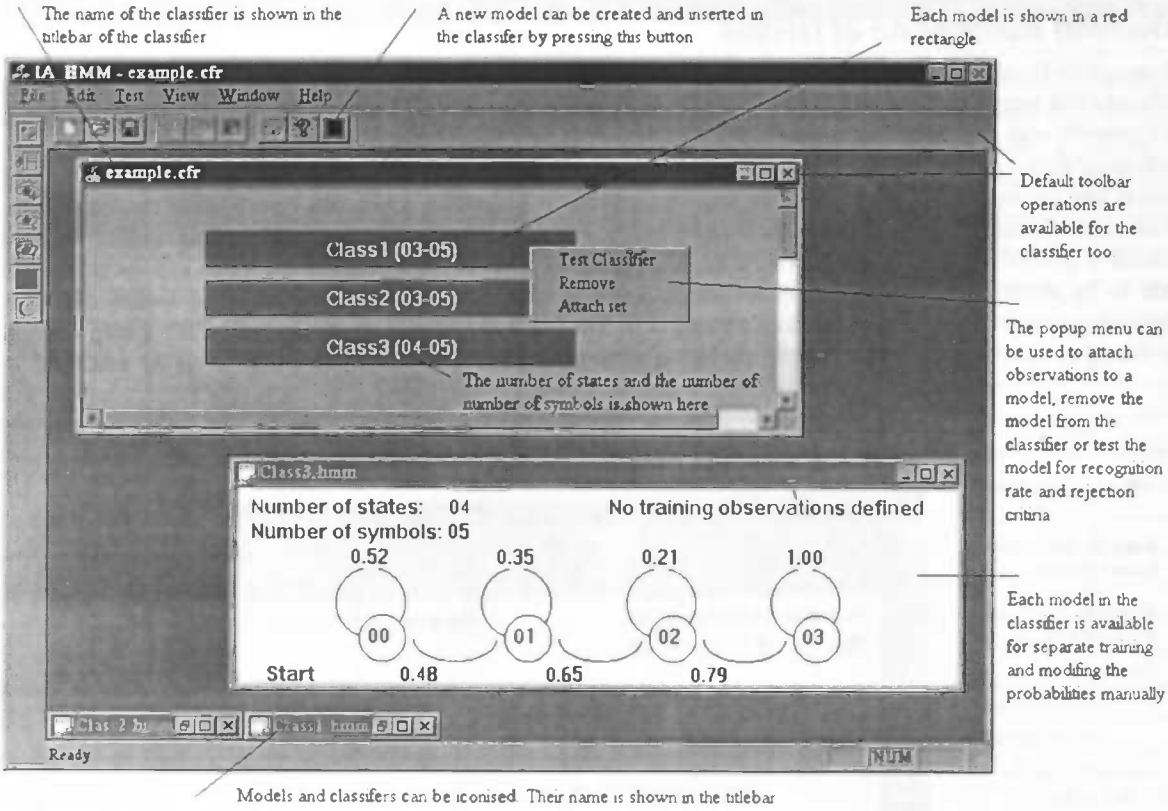
General description of IaHMM

Inter*active* Hidden Markov Modelling is a program that can be used to train Feed Forward Hidden Markov Models and integrate them into a classifier. The program currently only supports models of the type described in Chapter 7, with the modification included. Hidden Markov Models can be created only within the program by selecting *File - New - Ia_HMM* . At the moment only random initialised models can be created, the number of states and the number of input symbols must be specified. All probabilities can be modified manually from the *Edit* menu. A model can be inserted in a classifier by selecting *File - Add to Classifier*. Source code for a single model is generated by selection *File - Code generation*. In order to train or test a model an observation sequence has to be attached to the model by selecting *Test - Attach Observation Sequence*. The testing and training operations can be found in the menus *Edit* and *Test*. The icons in the toolbar on the left of the screen correspond to the operations in the menu's. The figure below shows the model and some operations using the toolbars.



Classifiers can be created within the program and using the context menu in the Windows Explorer if the program has been installed properly (at installation classifier documents are registered). The Classifier does not contain the actual models but only pointer to the models. Thus when a model is changed through the classifier the actual model is modified too. In a classifier file only the filenames of the models are stored with an absolute path name, this requires that the directories where the models are stores must be the same after copying a classifier from one PC to another, this method will be improved in a next version. Models can be added to a classifier by creating a new model and use random initialisation by selecting *Edit - New Hidden Markov Model*. The classifier can be used in two ways. Firstly, the models in the classifier can be trained separately on the same training set, this option if *Edit - Multi-train*. When using this option the parameters have to be set in the menu *Edit-Options-Multi-train*. Secondly, all the models can be trained separately using the Train option for the Hidden Markov Model. By selecting a model with the right mouse button a pop-up menu appears. In this menu several operations applying only to the selected model can be performed. Default observations can be attached to all models using *Test - Attach default operations*. With *Test - Multi-test* the classifier can be tested on a single observation sequence file, otherwise the pop-up menu should be used. The testing results are the percentage of correctly classified observation sequences, the percentage incorrectly classified and the percentage rejected observation sequences. Some information for the determination of rejection criteria is computed too in the tests. Rejection criteria can be set in *Edit - Options - Rejection criteria*. For C-code generation select *File - Generate Code*. The Hidden Markov Models are automatically included in the generated code. The generated code is presented in the

last two sections of this appendix. The figure below shows the classifier and some operations on the included models.



This introduction will do for understanding the rest of this appendix after Chapter 7 has been read. We hope to release the program in the future with a full manual.

Appendix I.1: Determining the number of states

For each of the keywords the number of states were determined separately. An estimate of the require number of states can be made on basis of the number of phonemes in the keyword and the variation in phoneme length. The number of states was determined by creating a classifier with a number of models of different size and training them using the Multi-train option described above. We had 170 observation sequences available for each keyword, 100 were used for training the remaining 70 for testing. Models were evaluated using the test set. The results are summarised below:

Keyword	#states	Recognition rate on train set	$-\sum \log(\alpha_{train})$	Recognition rate on test set	$-\sum \log(\alpha_{test})$
"0"					
	3	0	3012	1.4	2229
	4	2	2700	0.0	2065
	5	4	2501	4.3	1911
	6	8	2381	8.6	1836
	7	9	2383	8.6	1879
	8	16	2296	21.4	1789
	10	61	2254	55.7	1809
"1"					
	5	6.0	2548	5.7	1823
	6	11.0	2505	5.7	1845
	7	9.0	2519	7.1	1786
	8	34.0	2346	21.4	1756
	10	40.0	2366	60.0	1739

Keyword	#states	Recognition rate on train set	$-\sum \log(\alpha_{train})$	Recognition rate on test set	$-\sum \log(\alpha_{test})$
"2"					
	4	20.8	3587	18.6	3612
	5	4.0	3683	2.9	3637
	6	34.7	3513	31.4	3768
	7	14.9	3544	12.9	3707
.	8	25.7	3672	34.3	3728
"3"					
	4	1.0	3127	15.5	3412
	5	4.0	2942	32.4	3459
	6	6.0	2868	9.9	3514
	7	20.0	2789	15.5	3511
	8	31.0	2758	19.7	3651
	10	38.0	2762	7.0	4042
"4"					
	4	2.0	3231	8.6	2491
	5	4.0	3224	10.0	2466
	6	6.0	3086	17.1	2403
	7	34.0	2936	28.6	2441
	8	33.0	2919	25.71	2406
	10	21.0	2933	10.0	2491
"5"					
	3	5.0	3650	5.7	3588
	4	13.9	3505	17.1	3584
	5	9.9	3649	20.0	3587
	6	24.8	3627	5.7	3838
	7	16.8	3558	28.6	3528
	8	23.8	3510	14.3	3612
	10	5.9	3769	8.6	3729
"6"					
	3	0.0	3077	21.1	3118
	4	2.0	3054	11.3	3175
	5	7.0	2820	9.9	3191
	6	15.0	2778	15.5	3325
	7	13.0	2753	1.4	3391
	8	24.0	2706	28.2	3586
	10	39.0	2701	12.7	3569
"7"					
	5	3.0	3316	10.0	2492
	6	3.0	3221	4.3	2429
	7	13.0	3143	7.1	2387
	8	6.0	3121	4.3	2373
	9	18.0	3099	15.7	2384
	10	14.0	3082	22.9	2381
	12	43.0	3076	35.7	2420
"8"					
	3	0.0	2599	0.0	1774
	4	3.0	2411	8.6	1617
	5	10.0	2274	4.3	1384
	6	22.0	2253	24.3	1557
	7	32.0	2202	35.7	1527
	8	11.0	2267	11.4	1572
	10	22.0	2221	15.7	1588

Keyword	#states	Recognition rate on train set	$-\sum \log(\alpha_{train})$	Recognition rate on test set	$-\sum \log(\alpha_{test})$
"9"					
	4	23.8	3415	98.6	2712
	5	16.8	3369	1.4	2999
	6	14.9	3485	0.0	3436
	7	5.9	3534	0.0	3824
	8	23.8	3366	0.0	3723
	9	14.9	3404	0.0	3724
"M"					
	2	0.0	3192	43.7	3698
	3	1.0	2776	12.7	3838
	4	6.0	2627	8.0	3919
	5	17.0	2445	2.8	3788
	6	76.0	2347	32.4	3924
"Bel"					
	3	0.0	2852	0.0	1914
	4	1.0	2523	0.0	1758
	5	2.0	2352	4.3	1624
	6	5.0	2248	17.1	1545
	7	27.0	2161	25.7	1503
	9	65.0	2118	52.9	1533
"Opnieuw"					
	5	21.8	3533	2.9	3238
	6	7.9	3505	7.1	3054
	7	6.9	3545	24.3	2933
	8	25.7	3391	37.1	2859
	9	17.8	3461	24.3	3068
	11	19.8	3517	4.3	3310

Appendix I.2: Finding the best models

After determining the required number of states classifiers containing 5 or more models were created for each keyword. All the models in these classifiers where trained on the same data set (the train set). The best of these models was used in the classifier. The best models are summarised in the table below.

Keyword	Best of	#states	Recognition Rate on test set	$-\sum \log(\alpha_{test})$
"0"	5	6	30 %	1848
"1"	5	6	49 %	1761
"2"	5	6	74 %	3120
"3"	5	5	28 %	3561
"4"	5	7	43 %	2436
"5"	5	5	32 %	3198
"6"	5	3	38 %	3223
"7"	6	9	26 %	2384
"8"	6	7	27 %	1532
"9"	5	5	77 %	2712
"M"	5	3	35 %	3887
"Bel"	5	7	29 %	1503
"Opnieuw"	6	8	39 %	2841

Appendix I.3: Source code of the classifier

Interface file (Final.h)

```
/*********************************************************/
/*   Classifier code generator                           */
/*   (C)1997    M. van Veelen                            */
/*                                                       */
/*   Interface for classifier FINALCFR                   */
/*********************************************************/
#ifndef FINALCFR
#define FINALCFR
#ifndef WORD
#define WORD unsigned int
#endif

#define FINALCFR_LENGTH 16
#define FINALCFR_MODELS 13

/*     The Hidden Markov Models:                         */
#define HMM_0  0x0000
#define HMM_1  0x0001
#define HMM_2  0x0002
#define HMM_3  0x0003
#define HMM_4  0x0004
#define HMM_5  0x0005
#define HMM_6  0x0006
#define HMM_7  0x0007
#define HMM_8  0x0008
#define HMM_9  0x0009
#define HMM_M  0x000a
#define HMM_BEL        0x000b
#define HMM_OPNIEUW    0x000c

extern int FinalCfr(WORD *sequence);

#endif
```

Implementation (Final.c)

```
/*Implementation of Final                                */

#include "Final.h"
#include <math.h>
#define TRELLIS 17
#define LENGTH  16
#define MODELS  13
#define MIN_SCORE -1.000000e+009
#define MIN_DSCORE 0.000000e+000
#define double float
double scaling[17];
double alphas[17][9];
int NS[13]={6,6,5,6,7,5,6,9,7,4,6,7,8};
/* Probability tables                                    */
double Pr0[6]={7.910745e-001,4.810741e-001,5.459318e-001,4.405809e-
001,3.070574e-001,1.000000e+000};

double Pn0[6]={2.089255e-001,5.189259e-001,4.540682e-001,5.594191e-
001,6.929426e-001,1.000000e-006};

double Pro0[24][6]={
        {5.349002e-001,4.834842e-002,1.000000e-006,1.000000e-
006,1.000000e-006,4.142259e-001},
        {1.695867e-001,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,2.712447e-001},
        {1.465471e-001,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.533753e-001},
        {2.623536e-002,3.206087e-001,1.000000e-006,1.000000e-
006,1.000000e-006,4.413969e-002},
        {7.234275e-002,1.000000e-006,1.000000e-006,1.000000e-
006,9.999976e-001,1.331104e-002},
        {1.000000e-006,2.529871e-002,1.000000e-006,1.000000e-
006,1.000000e-006,2.112212e-002},
        {6.804711e-003,1.539700e-001,1.000000e-006,1.000000e-
006,1.000000e-006,2.241070e-002},
        {8.358553e-003,1.511070e-001,4.289522e-006,1.000000e-
006,1.000000e-006,1.288820e-002},
        {1.000000e-006,4.060617e-002,1.000000e-006,1.000000e-
006,1.000000e-006,4.526184e-003},
        {1.000000e-006,2.148100e-001,1.000000e-006,1.250279e-
006,1.000000e-006,1.813019e-002},
        {1.640048e-003,1.211542e-002,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006},
        {1.000000e-006,3.311342e-002,1.000000e-006,1.000000e-
006,1.000000e-006,4.548965e-003},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006},
        {1.000000e-006,2.035377e-005,2.028562e-002,1.000000e-
006,1.000000e-006,1.633225e-003},
        {1.000000e-006,1.000000e-006,3.612236e-002,1.000000e-
006,1.000000e-006,1.587190e-003},
        {3.165389e-003,1.000000e-006,1.000000e-006,2.110627e-
001,1.000000e-006,3.024176e-003},
        {2.884745e-002,1.000000e-006,1.000000e-006,7.889320e-
001,1.000000e-006,1.000000e-006},
        {1.000000e-006,1.000000e-006,7.458454e-002,1.000000e-
006,1.000000e-006,1.000000e-006},
        {1.000000e-006,1.000000e-006,2.377859e-001,1.000000e-
006,1.000000e-006,3.124289e-003},
        {1.570953e-003,1.000000e-006,3.828809e-001,1.000000e-
006,1.000000e-006,7.592999e-003},
```

```
        {1.000000e-006,1.000000e-006,1.160553e-001,1.000000e-
006,1.000000e-006,1.000000e-006},
        {1.000000e-006,1.000000e-006,1.274438e-001,1.000000e-
006,1.000000e-006,3.114912e-003},
        {1.000000e-006,1.000000e-006,4.835638e-003,1.000000e-
006,1.000000e-006,1.000000e-006}
};

double Pno0[24][6]={
        {6.801474e-002,1.478917e-002,1.000000e-006,5.509881e-
003,1.272385e-002,7.278481e-002},
        {6.119781e-002,6.978453e-002,1.427700e-001,1.911687e-
001,5.802599e-001,3.164557e-003},
        {3.705545e-001,9.520775e-002,1.719636e-002,1.127654e-
002,1.681636e-002,6.329114e-003},
        {1.563794e-001,6.585096e-002,2.291752e-002,1.057663e-
002,9.016006e-003,2.320675e-002},
        {1.066744e-002,4.686830e-002,1.518294e-001,4.340478e-
001,1.289514e-001,5.485232e-002},
        {2.305734e-002,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006},
        {3.462537e-002,1.144778e-001,2.300011e-002,8.826737e-
003,1.187872e-002,8.438819e-002},
        {6.167837e-002,1.717068e-001,1.444531e-001,6.776853e-
002,3.647706e-002,2.109705e-002},
        {3.053068e-003,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,9.493671e-003},
        {2.391702e-002,2.102836e-002,1.000000e-006,1.000000e-
006,1.000000e-006,7.700422e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,5.813496e-003,7.067511e-002},
        {2.149083e-002,2.911777e-002,1.000000e-006,1.000000e-
006,1.000000e-006,1.265823e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.023207e-001},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.793249e-002},
        {1.200000e-002,1.554285e-001,2.289000e-002,1.156222e-
002,1.741669e-002,7.278481e-002},
        {2.356402e-002,4.122494e-002,5.275759e-002,3.640769e-
002,1.742816e-002,6.329114e-003},
        {2.325395e-002,2.326939e-002,4.015598e-002,8.281728e-
003,1.168475e-002,3.691983e-002},
        {4.258301e-002,5.195298e-002,8.183178e-002,1.273170e-
001,8.202806e-002,8.438819e-003},
        {5.813778e-003,1.000000e-006,4.404657e-002,1.744182e-
002,1.162789e-002,4.219409e-002},
        {1.163257e-002,1.235374e-002,9.077048e-002,2.341748e-
002,3.482292e-002,7.594937e-002},
        {4.067958e-002,7.279457e-002,1.409801e-001,3.479236e-
002,1.171393e-002,5.274262e-003},
        {1.000000e-006,1.000000e-006,1.000000e-006,5.813953e-
003,1.000000e-006,7.700422e-002},
        {5.836885e-003,1.414377e-002,2.440090e-002,5.790917e-
003,1.134069e-002,9.177215e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,2.742616e-002}
};

double Pr1[6]={7.749643e-001,4.934820e-001,4.156969e-001,3.723111e-
001,5.042632e-001,1.000000e+000};

double Pn1[6]={2.250357e-001,5.065180e-001,5.843031e-001,6.276889e-
001,4.957368e-001,1.000000e-006};

double Pro1[24][6]={
        {4.483606e-001,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,3.089716e-001},
        {1.181973e-001,1.000000e-006,1.000000e-006,1.000000e-
006,1.123793e-002,2.319940e-001},
        {1.695564e-001,1.000000e-006,1.000000e-006,1.000000e-
006,1.062464e-006,3.144818e-001},
        {1.185753e-001,1.000000e-006,1.000000e-006,1.000000e-
006,7.643660e-001,5.311213e-002},
        {9.493965e-003,1.000000e-006,1.000000e-006,1.000000e-
006,5.796200e-002,1.131085e-002},
        {5.074079e-003,1.000000e-006,1.000000e-006,1.000000e-
006,2.255897e-002,4.162711e-003},
        {3.659620e-002,1.000000e-006,1.000000e-006,1.000000e-
006,1.330973e-001,7.557873e-003},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.095161e-
002,1.000000e-006,1.366819e-003},
        {1.688267e-003,1.000000e-006,1.000000e-006,1.000000e-
006,5.715318e-003,1.000000e-006},
        {1.688257e-003,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006},
        {1.688267e-003,1.000000e-006,1.000000e-006,1.000000e-
006,5.060095e-003,1.000000e-006},
        {2.577357e-002,1.000000e-006,1.000000e-006,1.379958e-
001,1.000000e-006,1.408312e-003},
        {5.232258e-003,1.000000e-006,1.000000e-006,3.375300e-
001,1.000000e-006,1.000000e-006},
        {1.000000e-006,1.000000e-006,2.806555e-001,7.511799e-
002,1.000000e-006,7.094588e-003},
        {1.479087e-002,1.000000e-006,1.000000e-006,4.384003e-
001,1.000000e-006,1.564092e-003},
        {1.690357e-002,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.367128e-003},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006},
```

```
        {2.638029e-002,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,9.576822e-003},
        {1.000000e-006,8.386708e-001,9.527062e-003,1.000000e-
006,1.000000e-006,2.911567e-002},
        {1.000000e-006,1.613242e-001,1.000000e-006,1.000000e-
006,1.000000e-006,4.254504e-003},
        {1.000000e-006,1.000000e-006,7.098134e-001,1.000000e-
006,1.000000e-006,1.266096e-002}
};

double Pno1[24][6]={
        {1.403666e-001,1.501638e-002,1.000000e-006,8.738072e-
003,2.206788e-002,3.589375e-002},
        {1.393213e-001,1.480370e-001,1.400297e-001,9.377132e-
002,1.969897e-002,6.819813e-002},
        {1.299506e-001,1.892322e-001,1.817881e-001,2.004590e-
001,4.214132e-001,6.748026e-002},
        {6.813364e-002,4.797981e-002,9.153018e-002,1.842854e-
001,2.598999e-001,6.030151e-002},
        {3.224894e-002,2.238719e-002,2.031039e-002,1.719168e-
002,2.383479e-002,5.527638e-002},
        {1.000000e-006,5.797602e-003,1.000000e-006,1.000000e-
006,1.000000e-006,5.240488e-002},
        {1.869951e-002,1.165592e-002,2.996940e-002,1.061561e-
001,5.153405e-002,6.101938e-002},
        {8.561189e-003,2.049476e-002,1.094962e-002,1.000000e-
006,1.000000e-006,4.666188e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,5.168701e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,6.460876e-003},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,6.479384e-003,2.440775e-002},
        {4.803626e-003,1.726434e-002,5.636645e-003,1.436006e-
001,1.255019e-002,2.153625e-003},
        {2.779499e-002,6.510984e-003,6.515208e-002,2.487801e-
002,3.534722e-002,1.507538e-002},
        {7.846990e-002,6.534960e-002,1.695007e-001,5.739183e-
002,2.925441e-002,3.015075e-002},
        {4.166386e-002,1.459369e-002,3.634526e-002,3.530479e-
002,4.282926e-002,6.676238e-002},
        {9.596403e-003,7.751540e-003,5.828548e-003,1.162415e-
002,1.000000e-006,6.676238e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,2.727925e-002},
        {5.833815e-003,1.000000e-006,5.793625e-003,5.798497e-
003,5.829442e-003,5.168701e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,3.948313e-002},
        {5.813447e-003,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,5.886576e-002},
        {6.900181e-002,6.196060e-002,5.141854e-002,5.843243e-
003,5.709274e-003,1.363963e-002},
        {7.800687e-002,1.036385e-001,1.573618e-002,1.161163e-
002,5.894115e-003,2.512563e-002},
        {8.334075e-002,4.850790e-002,1.892922e-002,1.679373e-
002,5.967663e-003,2.656138e-002},
        {5.839274e-002,2.138219e-001,1.510806e-001,7.655086e-
002,5.169003e-002,4.666188e-002}
};

double Pr2[5]={7.796637e-001,6.947879e-001,4.696806e-001,4.862087e-
001,1.000000e+000};

double Pn2[5]={2.203363e-001,3.052121e-001,5.303194e-001,5.137913e-
001,1.000000e-006};

double Pro2[24][5]={
        {4.048232e-001,1.000000e-006,1.000000e-006,4.145656e-
001,2.982837e-001},
        {1.729206e-001,2.798716e-003,1.000000e-006,1.000000e-
006,2.508889e-001},
        {1.015767e-001,1.000000e-006,1.000000e-006,7.223982e-
002,2.577403e-001},
        {3.122722e-002,1.000000e-006,1.000000e-006,1.822213e-
001,6.468364e-002},
        {2.550362e-003,1.000000e-006,1.000000e-006,7.608433e-
003,5.082031e-002},
        {1.000000e-006,2.305020e-003,1.000000e-006,2.844435e-
002,2.085522e-002},
        {5.158293e-003,2.603158e-002,1.000000e-006,1.000000e-
006,2.379114e-002},
        {3.081476e-002,1.516860e-002,1.000000e-006,1.094379e-
001,1.000000e-006},
        {5.346510e-002,1.000000e-006,1.000000e-006,5.218070e-
002,5.715518e-003},
        {5.626324e-002,1.000000e-006,1.000000e-006,1.189871e-
001,4.744983e-003},
        {3.288613e-002,1.000000e-006,7.771521e-002,1.000000e-
006,1.386833e-003},
        {3.122620e-002,1.000000e-006,2.586079e-001,1.000000e-
006,1.000000e-006},
        {2.466570e-002,1.000000e-006,1.285082e-001,1.000000e-
006,1.000000e-006},
        {2.677887e-002,3.114907e-003,2.361064e-001,1.000000e-
006,1.000000e-006},
        {1.191203e-002,4.144285e-002,2.278114e-001,1.000000e-
006,1.000000e-006},
        {1.000000e-006,1.280564e-001,1.000000e-006,1.000000e-
006,1.404094e-002},
        {1.000000e-006,2.553795e-003,1.000000e-006,6.143777e-
003,1.000000e-006},
        {1.660360e-003,2.528651e-003,2.625241e-002,1.000000e-
006,1.000000e-006},
        {1.000000e-006,1.276984e-002,1.000000e-006,1.000000e-
006,1.335589e-003},
```

```
        {1.000000e-006,8.650213e-002,1.000000e-006,1.000000e-
006,1.335545e-003},
        {1.795274e-003,2.895619e-001,1.000000e-006,8.167708e-
003,4.376019e-003},
        {3.591164e-003,2.197017e-001,1.000000e-006,1.000000e-
006,1.000000e-006},
        {1.000000e-006,1.674620e-001,1.000000e-006,1.000000e-
006,1.000000e-006},
        {6.682826e-003,1.000000e-006,4.499504e-002,1.000000e-
006,1.000000e-006}
};

double Pno2[24][5]={
        {2.007066e-003,4.970391e-002,5.889331e-002,7.429227e-
002,2.547170e-002},
        {3.922156e-002,3.682112e-002,6.927495e-002,8.614402e-
002,2.641509e-002},
        {1.913503e-001,1.738270e-001,2.205921e-001,2.714445e-
001,5.471698e-002},
        {1.105508e-001,3.170642e-002,6.879504e-002,8.489951e-
002,5.754717e-002},
        {4.623570e-002,5.291540e-002,2.039954e-002,9.277289e-
003,5.566038e-002},
        {2.746890e-002,3.125040e-002,2.158045e-002,1.767920e-
002,5.943396e-002},
        {9.443293e-002,3.791427e-002,3.139195e-002,3.890418e-
002,8.207547e-002},
        {2.184789e-001,6.501249e-003,1.000000e-006,2.207136e-
002,7.547170e-003},
        {2.807779e-002,1.000000e-006,2.175858e-002,5.880866e-
002,1.886792e-003},
        {1.000000e-006,5.912305e-003,1.872697e-002,1.510947e-
002,2.924528e-002},
        {1.000000e-006,1.000000e-006,9.352701e-002,3.500526e-
002,8.113208e-002},
        {1.000000e-006,1.000000e-006,1.059090e-001,2.548842e-
002,9.056604e-002},
        {1.000000e-006,8.116536e-003,1.753209e-002,3.488434e-
002,3.018868e-002},
        {2.843904e-002,1.128437e-001,3.031023e-002,4.070503e-
002,3.679245e-002},
        {8.270200e-003,1.058991e-001,3.515885e-002,1.241701e-
002,4.716981e-003},
        {3.407011e-002,1.000000e-006,1.000000e-006,2.027306e-
002,1.792453e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,6.509434e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,6.132075e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,8.207547e-002},
        {2.982725e-002,1.000000e-006,1.000000e-006,1.000000e-
006,4.150943e-002},
        {5.368016e-002,1.496276e-001,6.276993e-002,3.581534e-
002,4.433962e-002},
        {2.882844e-002,3.956911e-002,4.069758e-002,3.504231e-
002,1.792453e-002},
        {3.520854e-002,7.041761e-002,5.351415e-002,5.266880e-
002,1.320755e-002},
        {2.384889e-002,8.697264e-002,2.916782e-002,2.906975e-
002,1.320755e-002}
};

double Pr3[6]={7.337320e-001,5.876458e-001,4.302526e-001,3.726283e-
001,4.526626e-001,1.000000e+000};

double Pn3[6]={2.662680e-001,4.123542e-001,5.697474e-001,6.273717e-
001,5.473374e-001,1.000000e-006};

double Pro3[24][6]={
        {4.861115e-001,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,3.424690e-001},
        {1.675040e-001,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,2.403343e-001},
        {1.340975e-001,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,2.333681e-001},
        {1.805681e-002,3.527151e-002,1.000000e-006,1.000000e-
006,7.580979e-001,3.803052e-002},
        {7.823599e-003,1.591486e-002,1.000000e-006,1.000000e-
006,4.126472e-002,1.557364e-002},
        {3.807665e-003,1.189883e-002,1.000000e-006,1.000000e-
006,1.000000e-006,1.502590e-002},
        {2.531787e-003,1.750307e-001,1.000000e-006,1.000000e-
006,1.000000e-006,2.880604e-002},
        {8.523727e-003,9.398791e-002,1.000000e-006,1.000000e-
006,1.000000e-006,1.503500e-002},
        {3.434676e-003,3.912322e-003,1.000000e-006,1.000000e-
006,1.000000e-006,7.567632e-003},
        {2.109857e-003,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006},
        {9.133961e-002,1.000000e-006,1.000000e-006,1.000000e-
006,1.673528e-001,1.000000e-006},
        {4.634850e-002,4.468126e-003,1.000000e-006,2.571316e-
001,3.328211e-002,1.000000e-006},
        {2.613147e-002,1.000000e-006,1.000000e-006,7.312829e-
001,1.000000e-006,1.000000e-006},
        {1.000000e-006,1.000000e-006,2.349005e-001,1.000000e-
006,1.000000e-006,2.474725e-003},
        {1.000000e-006,1.888078e-001,2.793933e-002,1.158087e-
002,1.000000e-006,1.376939e-002},
        {1.000000e-006,8.115970e-002,1.000000e-006,1.000000e-
006,1.000000e-006,7.513367e-003},
        {1.000000e-006,1.631841e-002,1.000000e-006,1.000000e-
006,1.000000e-006,3.756476e-003},
        {1.000000e-006,8.159324e-003,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006},
```

```
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006},
        {1.000000e-006,2.019247e-002,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006},
        {1.000000e-006,2.711743e-001,1.299248e-002,1.000000e-
006,1.000000e-006,2.003485e-002},
        {1.000000e-006,5.502058e-002,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006},
        {1.000000e-006,1.868047e-002,1.022522e-001,1.000000e-
006,1.000000e-006,9.824376e-003},
        {2.178448e-003,1.000000e-006,6.219108e-001,1.000000e-
006,1.000000e-006,6.416478e-003}
};

double Pno3[24][6]={
        {1.726209e-001,6.478681e-002,7.861897e-002,4.181155e-
002,6.132889e-002,4.786836e-002},
        {4.786687e-002,1.319391e-001,1.307210e-001,1.483266e-
001,1.264528e-001,3.440539e-002},
        {1.261410e-001,1.295946e-001,1.475278e-001,1.331508e-
001,3.651530e-001,8.975318e-003},
        {1.050441e-001,3.541484e-002,4.544988e-002,9.193341e-
002,6.277191e-002,4.412865e-002},
        {4.175439e-002,8.136787e-002,5.641448e-002,5.134906e-
002,7.971580e-003,5.235602e-003},
        {6.909474e-003,6.338416e-003,1.000000e-006,1.000000e-
006,1.000000e-006,1.570681e-002},
        {6.070628e-002,2.186137e-002,9.803853e-003,1.000000e-
006,5.836290e-003,5.983545e-002},
        {1.624934e-001,1.142433e-002,4.582128e-006,1.000000e-
006,1.000000e-006,4.712042e-002},
        {1.440389e-002,1.000000e-006,1.000000e-006,6.773868e-
003,1.585253e-002,4.487659e-003},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,7.404637e-002},
        {9.809121e-003,1.000000e-006,1.000000e-006,4.147910e-
002,1.690738e-001,5.534779e-002},
        {1.000000e-006,1.606731e-002,1.702648e-002,2.525863e-
001,5.696074e-002,2.991773e-002},
        {2.237996e-002,3.545784e-002,1.813993e-001,1.317038e-
001,4.712235e-002,7.329843e-002},
        {2.883060e-002,7.470538e-002,9.097543e-002,2.358541e-
002,2.325524e-002,4.188482e-002},
        {6.180723e-002,1.308199e-001,9.634759e-002,1.283474e-
002,1.000000e-006,6.581900e-002},
        {1.785519e-002,1.764441e-002,1.000000e-006,5.813086e-
003,1.000000e-006,3.739716e-003},
        {5.813961e-003,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,5.160808e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,2.692595e-002},
        {1.000000e-006,5.813331e-003,1.000000e-006,1.000000e-
006,1.000000e-006,6.507105e-002},
        {2.934621e-004,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,5.759162e-002},
        {4.458724e-002,6.490931e-002,5.867792e-003,1.000000e-
006,1.000000e-006,5.310396e-002},
        {1.000000e-006,4.949634e-002,1.000000e-006,1.000000e-
006,1.000000e-006,4.487659e-003},
        {2.370202e-002,2.360546e-002,2.358478e-002,1.221095e-
002,1.000000e-006,7.404637e-002},
        {4.698006e-002,9.875302e-002,1.162576e-001,4.644109e-
002,5.821990e-002,5.534779e-002}
};

double Pr4[7]={5.944779e-001,6.148817e-001,4.577638e-001,5.047761e-
001,2.410220e-001,2.158811e-001,1.000000e+000};

double Pn4[7]={4.055221e-001,3.851183e-001,5.422362e-001,4.952239e-
001,7.589780e-001,7.841189e-001,1.000000e-006};

double Pro4[24][7]={
        {4.921853e-002,5.560484e-001,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,2.045672e-001},
        {4.796925e-001,1.534539e-002,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,3.369936e-001},
        {1.298812e-001,9.508021e-002,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,2.100146e-001},
        {1.307153e-002,1.077754e-001,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.173312e-001},
        {1.189355e-002,5.316444e-002,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,4.008692e-002},
        {1.000000e-006,6.874054e-003,1.000000e-006,1.000000e-
006,7.470243e-002,1.000000e-006,1.353166e-002},
        {2.230175e-002,5.081824e-002,1.000000e-006,1.000000e-
006,1.000000e-006,1.277895e-002,1.679744e-002},
        {1.000000e-006,4.063838e-002,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.031760e-002},
        {4.775036e-003,4.305273e-002,4.762603e-002,1.000000e-
006,1.000000e-006,1.000000e-006,1.463474e-002},
        {1.000000e-006,1.000000e-006,4.012383e-002,1.000000e-
006,1.000000e-006,1.000000e-006,1.222139e-003},
        {1.000000e-006,3.465184e-003,2.228627e-001,1.000000e-
006,1.000000e-006,1.000000e-006,1.000000e-006},
        {1.000000e-006,4.550348e-003,5.599337e-001,1.270432e-
001,1.000000e-006,1.000000e-006,1.383937e-002},
        {4.467169e-002,1.000000e-006,1.170314e-002,8.021299e-
001,1.000000e-006,1.000000e-006,9.688107e-003},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,3.971110e-001,1.000000e-006,1.000000e-006},
        {5.643087e-002,5.137672e-003,7.096459e-002,1.000000e-
006,1.000000e-006,2.405818e-001,4.130298e-003},
        {5.155745e-002,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,4.485818e-001,1.348311e-003},
        {7.903196e-003,7.273238e-003,1.000000e-006,1.000000e-
006,1.000000e-006,3.529903e-002,1.531146e-003},
```

```
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.297451e-003},
        {7.931958e-003,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.000000e-006},
        {1.000000e-006,7.075588e-003,1.000000e-006,1.000000e-
006,1.000000e-006,4.223447e-002,1.000000e-006},
        {7.139503e-002,1.000000e-006,4.678135e-002,1.000000e-
006,1.000000e-006,2.205191e-001,1.348805e-003},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,2.563074e-001,1.000000e-006,1.000000e-006},
        {1.183531e-002,3.700007e-003,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.000000e-006},
        {3.743818e-002,1.000000e-006,1.000000e-006,7.082287e-
002,2.718730e-001,1.000000e-006,1.319133e-003}
};

double Pno4[24][7]={
        {2.801125e-001,2.898990e-001,1.619374e-001,3.096485e-
002,1.000000e-006,1.000000e-006,3.371869e-002},
        {1.309557e-001,8.962620e-002,1.035564e-001,1.275324e-
001,1.150495e-001,1.489707e-001,1.348748e-002},
        {9.914736e-002,9.640002e-002,6.439961e-002,1.070537e-
001,1.290961e-001,1.333999e-001,7.707129e-003},
        {8.612608e-002,6.290783e-002,4.068358e-002,1.179391e-
002,4.292709e-002,1.432067e-001,5.780347e-003},
        {4.266513e-002,5.302134e-002,6.499418e-002,8.082575e-
002,3.649329e-002,9.816985e-003,2.986513e-002},
        {2.977585e-002,1.000000e-006,1.000000e-006,2.853523e-
002,2.061930e-002,1.759450e-002,1.926782e-002},
        {1.004273e-001,1.392348e-002,3.285943e-002,5.055536e-
002,1.278738e-001,2.084858e-001,4.431599e-002},
        {2.813337e-002,4.314703e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.190565e-002,5.587669e-002},
        {1.121021e-002,6.257474e-002,1.437489e-002,1.943930e-
002,7.387158e-003,1.000000e-006,9.152216e-002},
        {1.000000e-006,1.000000e-006,1.347836e-003,1.000000e-
006,1.000000e-006,1.000000e-006,5.394990e-002},
        {6.628252e-003,1.260212e-002,5.295035e-002,5.899681e-
002,7.432825e-002,7.174700e-002,1.000000e-006},
        {5.278625e-002,7.490418e-002,9.637054e-002,1.047811e-
001,4.826858e-002,2.699113e-002,1.926782e-002},
        {9.854563e-002,1.252702e-001,1.169281e-001,1.172900e-
001,7.671106e-002,6.957061e-002,1.926782e-003},
        {5.845222e-003,1.870668e-002,4.956912e-002,4.268825e-
002,7.082076e-003,1.744239e-002,5.202312e-002},
        {2.171966e-002,6.462710e-002,9.167342e-002,9.437194e-
002,1.469923e-001,6.596347e-002,2.504817e-002},
        {1.000000e-006,2.325579e-002,1.804360e-006,1.000000e-
006,7.650348e-003,1.396612e-002,9.152216e-002},
        {5.937112e-005,1.000000e-006,1.000000e-006,1.000000e-
006,5.506606e-003,1.860956e-002,2.408478e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,8.959538e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,6.358382e-002},
        {1.000000e-006,6.144942e-003,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,9.152216e-002},
        {1.000000e-006,1.000000e-006,1.202672e-003,6.346717e-
003,1.056805e-001,1.887897e-002,5.973025e-002},
        {1.000000e-006,1.000000e-006,5.814820e-003,1.000000e-
006,1.000000e-006,1.000000e-006,3.371869e-002},
        {1.000000e-006,1.000000e-006,5.811647e-003,1.000000e-
006,2.882458e-002,6.058328e-003,9.152216e-002},
        {5.860544e-003,6.131259e-003,9.552301e-002,1.188240e-
001,1.950846e-002,1.739115e-002,9.633911e-004}
};

double Pr5[5]={5.432901e-001,7.651580e-001,7.205051e-001,5.915413e-
001,1.000000e+000};

double Pn5[5]={4.567099e-001,2.348420e-001,2.794949e-001,4.084587e-
001,1.000000e-006};

double Pro5[24][5]={
        {1.000000e-006,4.725877e-001,1.000000e-006,1.000000e-
006,2.514039e-001},
        {1.000000e-006,1.607070e-001,1.000000e-006,6.441133e-
001,1.011320e-001},
        {1.000000e-006,1.363038e-001,1.000000e-006,1.174014e-
001,1.245501e-001},
        {3.302423e-002,6.172337e-002,1.000000e-006,1.000000e-
006,1.093918e-001},
        {1.012552e-002,4.766879e-002,1.000000e-006,1.000000e-
006,4.261356e-002},
        {4.887405e-003,6.807878e-003,1.000000e-006,1.000000e-
006,4.956407e-002},
        {1.551402e-001,3.015655e-002,1.000000e-006,1.000000e-
006,1.200301e-001},
        {4.791684e-002,4.255133e-003,1.000000e-006,1.000000e-
006,9.873514e-003},
        {1.000000e-006,7.447401e-002,4.638075e-003,1.000000e-
006,1.092541e-001},
        {1.848448e-002,5.315177e-003,1.000000e-006,2.081435e-
001,1.227842e-002},
        {1.510499e-002,1.000000e-006,1.000000e-006,1.000000e-
006,6.812187e-003},
        {9.725206e-002,1.000000e-006,1.000000e-006,1.000000e-
006,2.654824e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.648704e-003},
        {6.353600e-002,1.000000e-006,2.037534e-002,1.000000e-
006,1.000000e-006},
        {1.285847e-001,1.000000e-006,1.000000e-006,3.033613e-
002,1.000000e-006},
        {3.836821e-002,1.000000e-006,8.443613e-002,1.000000e-
006,2.554482e-003},
```

```
        {6.370803e-003,1.000000e-006,1.277098e-001,1.000000e-
006,1.524172e-002},
        {1.466195e-002,1.000000e-006,1.154420e-002,1.000000e-
006,6.470244e-003},
        {2.904375e-002,1.000000e-006,1.778678e-001,1.000000e-
006,3.773277e-003},
        {9.436620e-002,1.000000e-006,4.707761e-001,1.000000e-
006,3.560681e-003},
        {1.258280e-001,1.000000e-006,5.354462e-002,1.000000e-
006,1.000000e-006},
        {4.398667e-002,1.000000e-006,1.353165e-002,1.000000e-
006,1.000000e-006},
        {7.331577e-002,1.000000e-006,3.557432e-002,1.000000e-
006,1.000000e-006},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,3.297357e-003}
};

double Pno5[24][5]={
        {2.214066e-001,1.139584e-001,1.000000e-006,1.000000e-
006,8.368201e-003},
        {1.314150e-001,2.111279e-001,2.445130e-001,2.161814e-
001,7.949791e-002},
        {1.751682e-001,1.000000e-006,1.000000e-006,6.803473e-
002,5.020921e-002},
        {1.759349e-001,1.771002e-002,1.000000e-006,1.000000e-
006,5.774059e-002},
        {3.027953e-002,1.000000e-006,1.837887e-002,1.000000e-
006,4.937238e-002},
        {1.000000e-006,7.860647e-002,2.525467e-003,6.800046e-
003,1.004184e-002},
        {3.087638e-002,1.497736e-001,1.000000e-006,6.097493e-
002,1.506276e-002},
        {1.000000e-006,3.921332e-002,2.948675e-002,1.166251e-
002,5.020921e-003},
        {5.206459e-002,6.881363e-002,9.985823e-002,6.382986e-
002,7.531381e-002},
        {4.156987e-002,4.275567e-002,9.868013e-002,1.015576e-
001,1.087866e-002},
        {1.614997e-002,1.000000e-006,1.000000e-006,1.000000e-
006,4.518828e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,6.443515e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,3.598326e-002},
        {1.000000e-006,1.000000e-006,3.464154e-002,4.073740e-
002,6.192469e-002},
        {1.000000e-006,1.000000e-006,2.339014e-002,1.576199e-
001,1.000000e-006},
        {1.490395e-002,8.534121e-003,8.586408e-002,4.279989e-
002,8.368201e-004},
        {1.606267e-002,1.042920e-001,1.201519e-002,6.154825e-
003,1.004184e-002},
        {1.000000e-006,1.000000e-006,5.583991e-003,5.794762e-
003,6.443515e-002},
        {6.714906e-002,1.041494e-001,6.095481e-002,6.369466e-
002,7.866109e-002},
        {2.701745e-002,5.582557e-002,1.349168e-001,8.917199e-
002,6.778243e-002},
        {1.000000e-006,5.238477e-003,7.811109e-002,2.428427e-
002,3.598326e-002},
        {1.000000e-006,1.000000e-006,1.744201e-002,1.744138e-
002,6.108787e-002},
        {1.000000e-006,1.000000e-006,5.363583e-002,2.325598e-
002,4.769874e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,6.443515e-002}
};

double Pr6[6]={8.027592e-001,3.889041e-001,5.263512e-001,3.227152e-
001,3.636688e-001,1.000000e+000};

double Pn6[6]={1.972408e-001,6.110959e-001,4.736488e-001,6.772848e-
001,6.363312e-001,1.000000e-006};

double Pro6[24][6]={
        {3.962796e-001,1.000000e-006,1.000000e-006,1.000000e-
006,8.407544e-002,3.825097e-001},
        {1.351203e-002,2.909224e-001,1.000000e-006,1.000000e-
006,1.000000e-006,4.185693e-002},
        {2.107176e-002,3.627771e-001,1.000000e-006,1.000000e-
006,1.626931e-002,5.897854e-002},
        {6.365146e-003,4.780878e-002,1.000000e-006,1.380280e-
001,1.000000e-006,8.708539e-003},
        {1.907423e-003,3.510267e-002,1.000000e-006,1.000000e-
006,1.000000e-006,9.778387e-003},
        {1.453951e-003,1.000000e-006,1.000000e-006,2.245720e-
001,1.000000e-006,5.948144e-003},
        {1.550191e-003,1.000000e-006,1.000000e-006,3.283955e-
001,1.000000e-006,9.987028e-003},
        {1.659470e-002,2.203220e-003,1.000000e-006,7.858497e-
002,1.000000e-006,1.856862e-002},
        {6.633752e-002,1.000000e-006,1.000000e-006,1.000000e-
006,8.253017e-001,3.961314e-002},
        {4.645854e-001,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,3.814696e-001},
        {7.229808e-003,1.638633e-001,1.000000e-006,1.000000e-
006,3.365797e-002,2.151436e-003},
        {1.000000e-006,7.329439e-003,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,4.069051e-002,1.000000e-006},
        {1.000000e-006,2.944485e-002,1.000000e-006,1.524073e-
001,1.000000e-006,1.000000e-006},
        {1.000000e-006,4.981362e-002,1.000000e-006,1.474470e-
002,1.000000e-006,5.927144e-003},
```

```
        {1.000000e-006,1.000000e-006,1.641297e-001,2.629076e-
002,1.000000e-006,7.357205e-003},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006},
        {1.000000e-006,1.000000e-006,1.000000e-006,3.697207e-
002,1.000000e-006,1.000000e-006},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006},
        {1.000000e-006,1.000000e-006,2.006644e-002,1.000000e-
006,1.000000e-006,1.000000e-006},
        {3.112299e-003,1.000000e-006,3.148471e-001,1.000000e-
006,1.000000e-006,6.140764e-003},
        {1.000000e-006,1.000000e-006,2.466527e-001,1.000000e-
006,1.000000e-006,1.000000e-006},
        {1.000000e-006,1.000000e-006,2.542993e-001,1.000000e-
006,1.000000e-006,2.100473e-002},
        {1.000000e-006,1.073076e-002,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006}
};

double Pno6[24][6]={
        {1.407851e-001,1.245064e-001,8.141721e-002,3.753751e-
002,1.641209e-001,5.691643e-002},
        {1.240285e-001,7.680814e-002,7.556052e-002,9.240603e-
002,7.265688e-002,4.827089e-002},
        {9.203432e-002,2.008200e-001,2.007786e-001,1.778558e-
001,9.013379e-002,3.674352e-002},
        {3.057639e-002,1.358053e-003,1.049385e-002,7.257143e-
002,5.829104e-002,5.547550e-002},
        {1.730693e-002,9.177434e-003,1.398839e-002,1.108877e-
002,1.000000e-006,2.881844e-002},
        {6.476825e-003,1.022755e-002,3.182347e-002,3.657611e-
006,1.000000e-006,4.538905e-002},
        {2.923924e-002,1.178923e-002,6.114321e-002,4.451946e-
002,9.698414e-003,4.322767e-002},
        {7.146850e-002,1.308991e-001,6.638913e-002,1.526888e-
001,4.656060e-002,4.322767e-002},
        {9.144038e-002,1.000000e-006,6.384271e-003,8.648188e-
002,6.142215e-002,2.953890e-002},
        {2.053509e-001,1.000000e-006,7.332257e-003,7.737721e-
002,3.582365e-001,3.386167e-002},
        {1.067915e-006,1.000000e-006,1.000000e-006,3.224217e-
002,2.684775e-002,1.152738e-002},
        {1.859080e-002,1.000000e-006,1.000000e-006,2.325544e-
002,1.000000e-006,2.377522e-002},
        {1.586341e-002,7.392100e-003,1.000000e-006,1.000000e-
006,1.000000e-006,3.170029e-002},
        {1.000000e-006,9.470311e-002,3.078696e-002,3.037904e-
002,8.584645e-003,6.700288e-002},
        {1.000000e-006,6.696279e-002,7.491744e-002,1.500892e-
002,1.243502e-002,4.682997e-002},
        {2.093473e-002,6.561334e-002,7.764945e-002,1.000000e-
006,1.000000e-006,2.161383e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,5.813715e-003,3.314121e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.145264e-
002,1.000000e-006,6.844380e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,4.034582e-002},
        {5.813994e-003,1.000000e-006,3.290270e-002,9.675627e-
003,1.070273e-002,6.628242e-002},
        {6.868452e-002,1.140927e-001,1.184322e-001,4.202422e-
002,1.301667e-002,6.340058e-002},
        {1.853488e-002,2.806285e-002,5.271233e-002,4.771267e-
002,3.236737e-002,2.377522e-002},
        {4.286836e-002,4.707717e-002,5.137006e-002,3.571860e-
002,2.911008e-002,2.089337e-002},
        {1.000000e-006,1.050803e-002,5.917510e-003,1.000000e-
006,1.000000e-006,5.979827e-002}
};

double Pr7[9]={7.360403e-001,4.388149e-001,2.660467e-001,7.579181e-
002,1.583038e-001,2.314799e-001,3.956755e-001,3.094282e-
003,1.000000e+000};

double Pn7[9]={2.639597e-001,5.611851e-001,7.339533e-001,9.242082e-
001,8.416962e-001,7.685201e-001,6.043245e-001,9.969057e-
001,1.000000e-006};

double Pro7[24][9]={
        {3.956419e-001,2.406065e-002,1.000000e-006,1.000000e-
006,1.000000e-006,3.235128e-002,1.000000e-006,1.000000e-
006,4.906909e-001},
        {1.141979e-001,8.955472e-003,1.000000e-006,1.000000e-
006,1.000000e-006,2.221917e-001,1.000000e-006,1.000000e-
006,1.543878e-001},
        {1.558914e-001,2.410091e-002,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.008047e-001},
        {1.479794e-001,1.405943e-002,1.000000e-006,1.000000e-
006,1.000000e-006,3.917594e-001,9.741953e-002,1.000000e-
006,3.623449e-002},
        {1.274487e-002,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,6.321050e-002,1.000000e-
006,7.727893e-003},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,7.123436e-002,9.999934e-
001,1.352370e-002},
        {8.249842e-002,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,5.098132e-001,1.000000e-
006,5.803906e-003},
        {1.404098e-002,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.658001e-001,1.000000e-006,1.000000e-
006,2.633201e-002},
```

```
        {3.077127e-002,3.118067e-002,1.000000e-006,1.000000e-
006,1.000000e-006,1.001531e-001,7.212773e-002,1.000000e-
006,1.895554e-002},
        {1.696536e-003,8.234537e-001,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.080055e-001},
        {2.529516e-003,7.418676e-002,1.000000e-006,1.000000e-
006,1.000000e-006,8.773920e-002,1.000000e-006,1.000000e-
006,1.000000e-006},
        {1.597732e-002,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006},
        {4.170009e-003,1.000000e-006,1.000000e-006,1.000000e-
006,2.613368e-001,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006},
        {2.093250e-003,1.000000e-006,7.158366e-002,1.000000e-
006,1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,2.144233e-003},
        {8.923960e-003,1.000000e-006,1.000000e-006,1.000000e-
006,3.024618e-001,1.000000e-006,1.000000e-006,1.000000e-
006,2.268146e-003},
        {1.084298e-002,1.000000e-006,1.000000e-006,1.000000e-
006,3.496964e-001,1.000000e-006,9.526607e-002,1.000000e-
006,1.000000e-006},
        {1.000000e-006,1.000000e-006,1.000000e-006,3.533652e-
002,1.000000e-006,1.000000e-006,9.092509e-002,1.000000e-
006,1.000000e-006},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006},
        {1.000000e-006,1.000000e-006,1.000000e-006,2.287021e-
006,3.245750e-002,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006},
        {1.000000e-006,1.000000e-006,1.000000e-006,8.507316e-
001,1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,2.048621e-003},
        {1.000000e-006,1.000000e-006,2.411401e-001,1.000000e-
006,1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.866585e-002},
        {1.000000e-006,1.000000e-006,5.066557e-001,1.000000e-
006,1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,8.308890e-003},
        {1.000000e-006,1.000000e-006,1.806152e-001,1.139218e-
001,1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,4.097342e-003},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.117543e-
006,5.404034e-002,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006}
};

double Pno7[24][9]={
        {4.746895e-001,1.882871e-001,2.310907e-001,1.610966e-
001,8.617612e-002,5.671077e-002,5.725391e-002,1.331234e-
001,7.692308e-002},
        {4.913967e-002,7.476149e-002,8.886551e-002,1.178758e-
001,1.481276e-001,9.421701e-002,7.880078e-002,9.909346e-
002,8.882784e-002},
        {4.368878e-002,3.206160e-002,4.009320e-002,9.582505e-
002,1.150469e-001,1.770513e-001,1.765325e-001,2.719364e-
001,4.029304e-002},
        {5.750042e-003,1.000000e-006,3.481312e-002,7.869910e-
002,8.185667e-002,8.812956e-002,2.781900e-001,1.719912e-
001,1.831502e-003},
        {2.455584e-002,3.269509e-002,1.000000e-006,1.083945e-
002,3.024953e-002,3.575181e-002,2.990466e-002,2.784380e-
002,8.424908e-002},
        {6.080490e-003,1.167234e-002,1.226735e-002,1.103521e-
002,1.397525e-002,1.381796e-002,1.448775e-006,1.997249e-
006,9.065934e-002},
        {1.000000e-006,1.145531e-002,3.353706e-002,4.530945e-
002,9.603598e-002,9.929735e-002,9.758223e-002,5.972808e-
002,4.945055e-002},
        {2.249020e-002,1.696349e-001,8.818774e-002,5.965911e-
002,2.182144e-002,9.763278e-002,6.003512e-002,2.020402e-
002,7.326007e-002},
        {9.739440e-002,2.323354e-002,1.000000e-006,1.000000e-
006,1.475328e-002,3.022099e-002,1.000000e-006,1.000000e-
006,7.967033e-002},
        {8.338610e-002,1.754368e-001,5.700568e-002,1.829743e-
002,1.000000e-006,1.000000e-006,5.928524e-003,7.128209e-
003,8.241758e-003},
        {4.540174e-002,1.000000e-006,1.000000e-006,1.000000e-
006,8.455736e-003,1.000000e-006,1.000000e-006,1.000000e-
006,6.959707e-002},
        {2.708551e-002,2.140858e-002,1.162829e-002,1.000000e-
006,2.975772e-002,2.835374e-002,3.833013e-003,2.524051e-
002,6.684982e-002},
        {5.815569e-003,2.284498e-002,6.222819e-003,1.000000e-
006,2.641902e-002,1.000000e-006,5.820425e-003,2.325978e-
002,1.000000e-006},
        {5.791051e-003,4.676516e-002,6.447121e-002,7.316854e-
002,2.713464e-002,3.496622e-002,3.495796e-002,4.696747e-
002,1.098901e-002},
        {9.495436e-003,3.494465e-002,7.195444e-002,2.123074e-
002,5.841251e-002,8.210746e-002,2.883802e-002,2.922743e-
002,1.098901e-002},
        {1.746193e-002,2.318247e-002,5.238437e-002,1.458476e-
002,4.097923e-002,2.181824e-002,2.530772e-002,2.380721e-
002,3.571429e-002},
        {1.000000e-006,5.848493e-003,1.000000e-006,2.223814e-
002,8.941828e-003,2.088166e-002,2.242071e-002,1.421522e-
002,1.648352e-002},
```

```
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.162783e-002,1.000000e-006,1.000000e-006,1.000000e-
006,4.578755e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,5.523197e-
003,1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,2.289377e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,4.578755e-003},
        {3.524532e-002,1.083250e-001,1.000366e-001,1.189609e-
001,9.771219e-002,6.062971e-002,3.644446e-002,2.319235e-
002,5.860806e-002},
        {1.744183e-002,1.000000e-006,3.120455e-002,1.178830e-
002,3.488514e-002,1.753669e-002,1.000000e-006,5.489268e-
003,5.494505e-002},
        {5.830704e-003,1.000000e-006,2.587245e-002,4.439947e-
002,2.925795e-002,1.168691e-002,1.162767e-002,1.000000e-
006,2.747253e-003},
        {2.325578e-002,1.744185e-002,5.036295e-002,8.946692e-
002,1.837343e-002,2.918935e-002,4.652076e-002,1.755003e-
002,6.410256e-003}
};

double Pr8[7]={7.001458e-001,3.623343e-001,4.601371e-001,4.385036e-
001,6.255994e-001,2.207214e-001,1.000000e+000};

double Pn8[7]={2.998542e-001,6.376657e-001,5.398629e-001,5.614964e-
001,3.744006e-001,7.792786e-001,1.000000e-006};

double Pro8[24][7]={
        {5.070313e-001,1.000000e-006,1.000000e-006,1.000000e-
006,6.998624e-002,1.000000e-006,3.833917e-001},
        {9.900900e-002,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,3.200976e-001},
        {2.980252e-002,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.050252e-002},
        {1.245832e-002,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,6.348178e-002,1.000000e-006},
        {4.997999e-003,1.000000e-006,1.000000e-006,1.000000e-
006,1.442843e-002,3.995641e-002,1.000000e-006},
        {1.348831e-001,1.000000e-006,1.000000e-006,1.000000e-
006,4.415159e-001,1.000000e-006,2.280063e-002},
        {6.893152e-003,1.000000e-006,1.000000e-006,1.000000e-
006,1.412771e-001,1.000000e-006,1.492740e-002},
        {6.121344e-002,1.000000e-006,1.000000e-006,1.057475e-
001,1.000000e-006,8.760350e-001,1.000000e-006},
        {1.407917e-001,2.513486e-001,1.000000e-006,1.000000e-
006,3.327896e-001,1.000000e-006,5.102046e-002},
        {2.918401e-003,6.530328e-001,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,8.344387e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.000000e-006},
```

```
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.000000e-006},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.000000e-006},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.000000e-006},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.000000e-006},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,2.052200e-002,1.000000e-006},
        {1.000000e-006,5.429736e-002,1.000000e-006,2.827345e-
001,1.000000e-006,1.000000e-006,1.667494e-002},
        {1.000000e-006,1.786454e-002,1.339890e-002,1.000000e-
006,1.000000e-006,1.000000e-006,2.797503e-006},
        {1.000000e-006,2.345468e-002,1.296381e-001,6.115149e-
001,1.000000e-006,1.000000e-006,4.815643e-002},
        {1.000000e-006,1.000000e-006,8.569605e-001,1.000000e-
006,1.000000e-006,1.000000e-006,4.898105e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.000000e-006},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.000000e-006},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.000000e-006},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.000000e-006}
};

double Pno8[24][7]={
        {1.256573e-001,8.110726e-002,3.351256e-006,1.523543e-
003,1.503249e-002,1.303729e-001,2.578269e-002},
        {8.277597e-002,9.571784e-002,1.386087e-001,1.394363e-
001,1.591368e-001,1.635209e-001,6.445672e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.699176e-
002,2.536052e-002,5.468879e-002,2.578269e-002},
        {1.000000e-006,5.984404e-003,5.798661e-003,8.336475e-
003,1.012583e-002,1.570581e-002,8.563536e-002},
        {1.000000e-006,1.000000e-006,6.102311e-003,1.334287e-
002,8.162516e-003,1.831818e-002,4.235727e-002},
        {1.739148e-001,7.355967e-002,9.693382e-002,1.344213e-
001,1.998679e-001,1.013857e-001,3.314917e-002},
        {3.981618e-002,2.811456e-002,5.181578e-002,4.102972e-
002,3.253848e-002,7.952230e-003,4.604052e-002},
        {1.730634e-002,3.916118e-002,9.643865e-002,2.078039e-
001,1.494578e-001,1.092121e-001,1.657459e-002},
        {1.358953e-001,1.281623e-001,1.174607e-001,1.384201e-
001,2.204751e-001,2.156930e-001,4.696133e-002},
        {6.859204e-002,7.472307e-002,2.898933e-002,1.000000e-
006,1.000000e-006,1.000000e-006,4.511971e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,4.972376e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,4.604052e-003},
```

```
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.104972e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,8.195212e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,2.025783e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.197053e-002},
        {1.563064e-001,9.595394e-002,5.060117e-002,1.634342e-
001,4.465538e-002,3.933747e-002,2.670350e-002},
        {1.238655e-002,1.000000e-006,1.836552e-002,1.000000e-
006,1.743162e-002,1.000000e-006,1.381215e-002},
        {1.176644e-001,2.109004e-001,2.464943e-001,4.686536e-
002,4.142951e-002,7.207378e-002,4.051565e-002},
        {6.968351e-002,1.666147e-001,1.365733e-001,8.839341e-
002,7.632560e-002,7.173733e-002,8.471455e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,7.182320e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,9.208103e-004},
        {1.000000e-006,1.000000e-006,5.813592e-003,1.000000e-
006,1.000000e-006,1.000000e-006,7.366483e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,7.642726e-002}
};

double Pr9[4]={8.075779e-001,6.972961e-001,6.038821e-
001,1.000000e+000};

double Pn9[4]={1.924221e-001,3.027039e-001,3.961179e-001,1.000000e-
006};

double Pro9[24][4]={
        {4.083239e-001,1.000000e-006,1.215156e-002,3.833760e-001},
        {1.130788e-001,1.000000e-006,1.000000e-006,1.714660e-001},
        {1.948749e-001,1.000000e-006,1.000000e-006,2.451642e-001},
        {1.781287e-001,1.000000e-006,2.742611e-002,1.114426e-001},
        {1.737039e-002,1.000000e-006,2.193903e-002,2.751616e-002},
        {6.944650e-003,1.000000e-006,5.728445e-002,5.950356e-003},
        {3.926583e-002,1.000000e-006,2.749298e-001,8.908648e-003},
        {2.050237e-002,1.000000e-006,2.368682e-001,2.088639e-003},
        {9.426595e-004,1.000000e-006,4.166189e-002,1.000000e-006},
        {5.961414e-003,1.000000e-006,3.960687e-003,8.403026e-006},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-006},
        {1.000000e-006,6.029639e-002,1.000000e-006,1.221335e-003},
        {1.000000e-006,4.541479e-002,1.000000e-006,1.000000e-006},
        {1.480098e-003,1.576465e-001,1.000000e-006,1.000000e-006},
        {2.908438e-003,1.148961e-001,1.771860e-002,8.767319e-003},
        {7.358651e-003,5.581372e-004,7.092756e-002,5.583616e-003},
        {1.393882e-003,1.000000e-006,8.570123e-002,1.000000e-006},
        {1.000000e-006,1.000000e-006,1.906801e-002,1.000000e-006},
        {1.000000e-006,1.000000e-006,1.516452e-002,1.000000e-006},
        {1.000000e-006,1.767039e-002,4.194520e-002,1.000000e-006},
        {1.464857e-003,1.137708e-001,7.325110e-002,3.675769e-003},
        {1.000000e-006,2.110181e-001,1.000000e-006,5.843133e-003},
        {1.000000e-006,1.022753e-001,1.000000e-006,8.677598e-003},
        {1.000000e-006,1.764494e-001,1.000000e-006,1.030963e-002}
};

double Pno9[24][4]={
        {1.000000e-006,1.000000e-006,1.000000e-006,2.136752e-003},
        {9.701976e-002,7.633303e-002,5.715516e-002,3.525641e-002},
        {1.178355e-001,1.088759e-001,1.368790e-001,2.564103e-002},
        {1.965027e-001,6.210876e-002,3.394168e-001,9.615385e-002},
        {6.675752e-002,9.766347e-003,1.000000e-006,4.059829e-002},
        {1.000000e-006,1.000000e-006,2.252490e-002,6.303419e-002},
        {1.591546e-001,5.693493e-002,2.138027e-001,1.923077e-002},
        {3.950849e-002,3.592603e-001,3.903570e-002,2.457265e-002},
        {2.188166e-002,3.669554e-003,1.000000e-006,2.991453e-002},
        {1.541046e-002,1.000000e-006,1.000000e-006,2.991453e-002},
        {1.000000e-006,1.744046e-002,1.000000e-006,6.517094e-002},
        {1.885713e-002,3.370909e-002,1.757569e-002,8.119658e-002},
        {1.162767e-002,5.847929e-003,1.744301e-002,3.098291e-002},
        {3.862516e-002,7.775518e-002,3.170121e-002,3.205128e-002},
        {3.982011e-002,8.869692e-002,1.816295e-002,9.081197e-002},
        {4.306760e-002,1.000000e-006,9.784920e-003,3.952991e-002},
        {6.423877e-003,1.138671e-002,8.477066e-003,9.508547e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,2.884615e-002},
        {1.000000e-006,1.000000e-006,5.949713e-003,1.602564e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.495726e-002},
        {3.522502e-002,3.566334e-002,1.929979e-002,2.777778e-002},
        {3.716324e-002,1.000000e-006,1.000000e-006,5.341880e-002},
        {1.556535e-002,3.045500e-002,2.404817e-002,9.615385e-003},
        {3.955318e-002,2.209387e-002,3.873993e-002,4.807692e-002}
};

double PrM[6]={7.676718e-001,6.760145e-001,4.508695e-001,3.248580e-
001,5.947756e-003,1.000000e+000};

double PnM[6]={2.323282e-001,3.239855e-001,5.491305e-001,6.751420e-
001,9.940522e-001,1.000000e-006};

double ProM[24][6]={
        {5.305281e-001,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,4.276824e-001},
        {1.855817e-001,1.000000e-006,1.000000e-006,1.000000e-
006,9.999952e-001,1.506126e-001},
        {1.540469e-001,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,2.610288e-001},
        {6.845045e-002,1.000000e-006,1.757230e-002,8.498152e-
001,1.000000e-006,1.117290e-002},
        {1.376667e-002,1.000000e-006,1.000000e-006,1.139321e-
001,1.000000e-006,6.513563e-003},
        {2.358608e-003,1.000000e-006,1.958133e-002,1.000000e-
006,1.000000e-006,8.123887e-003},
```

```
        {1.513394e-002,1.000000e-006,7.955849e-001,1.000000e-
006,1.000000e-006,1.503675e-002},
        {1.447720e-002,1.000000e-006,1.327613e-001,1.000000e-
006,1.000000e-006,5.939762e-003},
        {1.935476e-003,8.104346e-003,1.000000e-006,1.000000e-
006,1.000000e-006,1.929249e-003},
        {7.881235e-003,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,9.500630e-003},
        {1.000000e-006,1.000000e-006,1.000000e-006,3.624803e-
002,1.000000e-006,1.000000e-006},
        {1.124291e-006,1.000000e-006,1.111589e-002,1.000000e-
006,1.000000e-006,1.000000e-006},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006},
        {1.000000e-006,1.719449e-002,1.000000e-006,1.000000e-
006,1.000000e-006,1.330187e-003},
        {4.078313e-003,6.159499e-002,2.744907e-004,1.000000e-
006,1.000000e-006,6.921523e-003},
        {1.000000e-006,4.375103e-002,1.000000e-006,1.000000e-
006,1.000000e-006,4.367163e-003},
        {1.759535e-003,1.000000e-006,2.280388e-002,1.000000e-
006,1.000000e-006,1.000000e-006},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006},
        {1.000000e-006,1.105095e-002,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006},
        {1.000000e-006,9.460685e-002,1.000000e-006,1.000000e-
006,1.000000e-006,8.668026e-003},
        {1.000000e-006,1.341061e-001,3.029602e-004,1.000000e-
006,1.000000e-006,1.765969e-002},
        {1.000000e-006,4.095252e-001,1.000000e-006,1.000000e-
006,1.000000e-006,4.859655e-002},
        {1.000000e-006,2.200639e-001,1.000000e-006,1.000000e-
006,1.000000e-006,1.491617e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006}
};

double PnoM[24][6]={
        {2.383632e-001,8.366438e-002,1.681907e-002,1.000000e-
006,1.056230e-002,4.191617e-002},
        {3.293991e-002,1.000000e-006,8.611092e-003,1.073742e-
001,4.921622e-002,9.580838e-002},
        {1.026015e-001,1.197885e-001,1.158196e-001,1.428590e-
001,4.395682e-001,3.792415e-002},
        {2.477501e-002,3.962942e-002,2.217624e-001,3.863060e-
001,2.229848e-001,5.389222e-002},
        {1.000000e-006,3.026511e-002,4.990025e-002,6.185949e-
003,6.253390e-003,2.594810e-002},
        {3.818871e-002,8.345371e-002,1.000000e-006,1.000000e-
006,5.590814e-003,3.892216e-002},
        {8.611409e-002,1.305099e-001,2.935145e-001,7.956591e-
002,8.425303e-002,1.000000e-006},
        {1.262450e-002,7.571491e-002,1.069820e-001,8.966015e-
002,6.751311e-002,8.682635e-002},
        {1.736569e-002,1.000000e-006,1.000000e-006,1.000000e-
006,9.170447e-003,2.894212e-002},
        {6.100316e-002,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.397206e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,6.087824e-002},
        {1.162380e-002,1.000000e-006,1.189733e-002,2.455728e-
002,6.742771e-003,8.982036e-003},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.996008e-002},
        {7.476463e-002,1.000000e-006,1.735586e-002,1.000000e-
006,1.000000e-006,8.483034e-002},
        {4.711122e-002,1.014415e-001,3.050268e-002,3.240790e-
002,6.079287e-003,4.890220e-002},
        {6.022923e-002,1.532785e-001,1.812507e-002,2.319834e-
002,1.300265e-002,5.988024e-003},
        {1.000000e-006,2.706533e-002,6.510236e-003,2.312614e-
002,5.969812e-003,1.397206e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006},
        {1.000000e-006,2.926586e-002,1.000000e-006,5.815091e-
003,1.000000e-006,8.682635e-002},
        {1.958740e-002,3.050097e-002,2.319470e-002,8.254432e-
003,3.545923e-003,7.385230e-002},
        {4.690599e-002,2.129140e-002,2.293047e-002,1.061185e-
002,1.316723e-002,6.886228e-002},
        {1.018119e-001,3.675978e-002,3.190320e-002,5.313772e-
002,3.845798e-002,8.083832e-002},
        {2.398939e-002,3.155482e-002,2.417055e-002,6.939807e-
003,1.792203e-002,1.896208e-002},
        {1.000000e-006,5.813532e-003,1.000000e-006,1.000000e-
006,1.000000e-006,2.994012e-003}
};

double PrBel[7]={7.895551e-001,4.509694e-001,4.014443e-001,2.426600e-
001,1.286838e-001,3.209441e-001,1.000000e+000};

double PnBel[7]={2.104449e-001,5.490306e-001,5.985557e-001,7.573400e-
001,8.713162e-001,6.790559e-001,1.000000e-006};

double ProBel[24][7]={
        {6.054029e-001,1.290406e-002,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,5.479119e-001},
        {1.783466e-001,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,2.013664e-001},
        {6.562191e-002,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.124495e-001},
        {1.825626e-002,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,6.246186e-003},
        {7.500873e-002,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,9.860105e-001,4.854795e-003},
```

```
        {9.310562e-003,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,4.571498e-003},
        {1.414210e-002,1.000000e-006,1.000000e-006,3.137790e-
002,1.000000e-006,1.000000e-006,1.371883e-002},
        {4.136883e-003,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,4.909249e-003},
        {2.882964e-003,1.000000e-006,1.515552e-001,1.000000e-
006,1.000000e-006,1.398713e-002,1.000000e-006},
        {3.247513e-003,3.460959e-001,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,2.267504e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.000000e-006},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.000000e-006},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.000000e-006},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.000000e-006},
        {1.000000e-006,1.621509e-002,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,3.066603e-003},
        {1.000000e-006,6.197779e-002,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.676216e-002},
        {6.505457e-003,1.000000e-006,1.000000e-006,3.720871e-
001,1.000000e-006,1.000000e-006,1.000000e-006},
        {1.713737e-002,1.000000e-006,1.000000e-006,1.000000e-
006,9.999946e-001,1.000000e-006,1.000000e-006},
        {1.000000e-006,1.000000e-006,1.000000e-006,5.671642e-
001,1.000000e-006,1.000000e-006,3.051529e-003},
        {1.000000e-006,2.639583e-002,6.802158e-001,2.936741e-
002,1.000000e-006,1.000000e-006,1.236830e-002},
        {1.000000e-006,2.092240e-001,1.682238e-001,1.000000e-
006,1.000000e-006,1.000000e-006,2.148054e-002},
        {1.000000e-006,1.607270e-001,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,4.571500e-003},
        {1.000000e-006,1.664570e-001,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.999555e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.000000e-006}
};

double PnoBel[24][7]={
        {1.055920e-001,2.293675e-002,1.243228e-002,2.334151e-
002,2.507113e-002,7.189908e-002,8.289612e-002},
        {7.794787e-002,9.152474e-002,1.178373e-001,1.884761e-
001,2.082944e-001,5.412985e-001,9.443861e-003},
        {2.320222e-001,1.215411e-001,8.281883e-002,1.098020e-
002,1.885843e-002,2.133195e-002,9.024134e-002},
        {3.822177e-002,4.328985e-003,1.000000e-006,5.526999e-
003,1.075804e-002,1.000000e-006,7.450157e-002},
        {2.898694e-002,3.090015e-002,4.948472e-002,7.891411e-
002,3.485022e-001,1.391079e-001,1.049318e-003},
        {6.076129e-003,5.503766e-003,1.000000e-006,1.000000e-
006,5.813641e-003,1.000000e-006,3.462749e-002},
        {1.095730e-001,7.225189e-003,6.714471e-002,5.525175e-
002,1.000000e-006,1.000000e-006,1.678909e-002},
        {3.600830e-002,5.337421e-002,7.686420e-002,8.741192e-
002,6.692847e-002,4.632382e-002,1.154250e-002},
        {6.955273e-003,5.692100e-002,9.563560e-003,1.144972e-
002,1.697033e-002,1.743704e-002,3.147954e-003},
        {6.552280e-002,9.691112e-002,7.232530e-002,4.928754e-
002,1.298106e-002,6.040412e-003,9.863589e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,8.394544e-003},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,8.394544e-003},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,5.351522e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,2.203568e-002},
        {3.131975e-002,7.682629e-003,1.000000e-006,5.744698e-
003,1.000000e-006,1.000000e-006,2.308499e-002},
        {2.960845e-002,2.204911e-002,2.534066e-002,1.000000e-
006,1.000000e-006,1.000000e-006,8.499475e-002},
        {1.047631e-002,1.000000e-006,6.750474e-003,1.303630e-
001,4.017702e-002,1.000000e-006,3.567681e-002},
        {1.163104e-002,4.070594e-002,7.135973e-002,2.265406e-
001,1.526725e-001,9.324387e-002,7.345226e-003},
        {1.244025e-002,3.599872e-002,1.156179e-001,2.049147e-
002,1.752575e-002,1.153388e-002,5.876180e-002},
        {7.320233e-002,1.735221e-001,2.463850e-001,7.117705e-
002,5.230368e-002,3.496262e-002,8.184680e-002},
        {9.121095e-002,1.500665e-001,2.055004e-002,1.000000e-
006,1.162426e-002,1.107422e-002,8.289612e-002},
        {7.670133e-003,2.885228e-002,5.875275e-003,1.744190e-
002,1.000000e-006,1.000000e-006,5.981112e-002},
        {2.553459e-002,4.995567e-002,1.964987e-002,1.760127e-
002,1.151833e-002,5.746444e-003,2.518363e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,2.518363e-002}
};

double PrOpnieuw[8]={5.666058e-001,4.029735e-001,3.899737e-
001,5.186205e-001,4.558876e-001,2.074026e-001,2.925630e-
001,1.000000e+000};

double PnOpnieuw[8]={4.333942e-001,5.970265e-001,6.100263e-
001,4.813795e-001,5.441124e-001,7.925974e-001,7.074370e-
001,1.000000e-006};

double ProOpnieuw[24][8]={
        {4.192557e-001,1.000000e-006,1.000000e-006,9.718933e-
001,1.000000e-006,1.000000e-006,1.000000e-006,2.904815e-001},
        {7.644898e-002,1.000000e-006,7.680290e-001,2.232774e-
002,1.000000e-006,1.000000e-006,1.000000e-006,2.626015e-001},
        {8.677877e-002,1.000000e-006,1.195969e-001,9.288882e-
006,1.000000e-006,1.000000e-006,1.092115e-001,1.272431e-001},
```

```
        {3.521911e-002,1.000000e-006,5.886773e-003,1.000000e-
006,1.493295e-001,1.000000e-006,2.434479e-001,8.005736e-002},
        {1.000000e-006,2.129992e-001,1.064846e-001,1.000000e-
006,1.000000e-006,1.000000e-006,1.902714e-001,5.867960e-002},
        {1.000000e-006,1.345842e-002,1.000000e-006,1.000000e-
006,1.000000e-006,1.250011e-001,1.000000e-006,8.132274e-003},
        {9.279084e-003,1.370188e-001,1.000000e-006,1.000000e-
006,7.535075e-002,1.000000e-006,1.000000e-006,6.684742e-002},
        {1.000000e-006,4.382579e-002,1.000000e-006,1.000000e-
006,3.614858e-002,1.000000e-006,1.000000e-006,2.262924e-002},
        {1.000000e-006,2.568555e-002,1.000000e-006,5.766750e-
003,1.000000e-006,1.000000e-006,1.000000e-006,7.096344e-003},
        {1.000000e-006,4.305803e-002,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.000000e-006,5.961642e-003},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.000000e-006,1.534857e-003},
        {6.773599e-002,1.000000e-006,1.000000e-006,1.000000e-
006,8.143420e-002,1.000000e-006,1.160567e-001,1.040215e-002},
        {4.002338e-002,1.000000e-006,1.000000e-006,1.000000e-
006,1.107988e-001,1.000000e-006,1.000000e-006,1.000000e-006},
        {4.891784e-002,1.000000e-006,1.000000e-006,1.000000e-
006,2.058149e-001,1.000000e-006,1.000000e-006,1.839553e-002},
        {8.945586e-002,1.000000e-006,1.000000e-006,1.000000e-
006,2.003055e-001,2.998882e-002,1.263019e-001,2.626640e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,7.359018e-003,3.453049e-001,1.681058e-002,3.098845e-003},
        {1.000000e-006,1.325461e-001,1.000000e-006,1.000000e-
006,1.000000e-006,2.080793e-005,1.000000e-006,1.776171e-003},
        {5.194129e-003,2.258481e-001,1.000000e-006,1.000000e-
006,1.000000e-006,3.475350e-001,1.000000e-006,2.663081e-003},
        {1.000000e-006,5.735717e-002,1.000000e-006,1.000000e-
006,8.412116e-003,6.665218e-002,1.000000e-006,1.000000e-006},
        {1.000000e-006,1.081997e-001,1.000000e-006,1.000000e-
006,6.998130e-003,8.549233e-002,1.121434e-006,1.000000e-006},
        {1.051240e-002,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,8.468513e-002,1.000000e-006},
        {2.223533e-002,1.000000e-006,1.000000e-006,1.000000e-
006,2.081726e-002,1.000000e-006,1.000000e-006,1.000000e-006},
        {2.668240e-002,1.000000e-006,1.000000e-006,1.000000e-
006,1.387817e-002,1.000000e-006,2.811449e-002,1.000000e-006},
        {6.225839e-002,1.000000e-006,1.000000e-006,1.000000e-
006,8.334929e-002,1.000000e-006,8.509577e-002,6.132632e-003}
};

double PnoOpnieuw[24][8]={
        {1.886523e-001,9.408418e-002,4.312297e-001,4.690097e-
001,1.096072e-001,1.511848e-002,2.583564e-002,2.268761e-002},
        {1.809416e-001,4.065832e-001,2.497852e-001,1.000000e-
006,1.426216e-001,2.495298e-001,4.224625e-001,5.061082e-002},
        {5.776717e-002,4.378730e-002,1.460424e-001,1.138616e-
001,2.454826e-002,3.267655e-002,3.866306e-002,8.726003e-003},
        {3.484326e-002,5.640459e-002,1.026279e-001,1.123557e-
001,2.784585e-002,2.303754e-002,3.495866e-002,2.705061e-002},
        {1.729351e-001,2.867784e-001,1.000000e-006,5.644393e-
002,2.298991e-001,1.677597e-001,3.466346e-001,1.308901e-002},
        {9.899397e-003,1.000000e-006,5.375873e-002,4.281765e-
006,1.489954e-002,1.000000e-006,1.000000e-006,3.490401e-003},
        {1.278059e-001,2.799811e-002,1.655503e-002,1.345656e-
001,5.964462e-002,1.393378e-001,9.176535e-003,7.591623e-002},
        {3.384636e-002,6.557244e-003,1.000000e-006,6.831578e-
002,5.462772e-002,8.059936e-002,1.000000e-006,5.148342e-002},
        {1.000000e-006,1.173180e-002,1.000000e-006,2.172043e-
002,7.050503e-003,1.000000e-006,7.902547e-003,8.115183e-002},
        {5.819917e-003,1.000000e-006,1.000000e-006,1.000000e-
006,3.553817e-002,1.164630e-002,1.000000e-006,5.497382e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.000000e-006,6.980803e-003},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.000000e-006,4.013962e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,2.344524e-002,6.282723e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,7.874468e-003,1.000000e-006,1.000000e-006,2.356021e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,9.052779e-002,1.000000e-006,1.000000e-006,1.134380e-002},
        {6.976664e-002,1.000000e-006,1.000000e-006,1.000000e-
006,8.199058e-003,1.000000e-006,1.612154e-002,7.766143e-002},
        {2.055217e-002,2.952213e-002,1.000000e-006,6.064704e-
003,2.069950e-002,1.082817e-001,3.844876e-002,7.853403e-002},
        {4.990753e-002,1.000000e-006,1.000000e-006,7.263342e-
003,7.461056e-002,1.364430e-001,2.384014e-002,5.584642e-002},
        {6.129360e-003,2.492358e-002,1.000000e-006,1.039310e-
002,1.000000e-006,1.000000e-006,1.000000e-006,4.363002e-003},
        {8.363840e-003,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,2.320911e-002,6.694541e-003,6.980803e-002},
        {3.276813e-002,1.162781e-002,1.000000e-006,1.000000e-
006,9.180541e-002,6.892665e-003,1.000000e-006,3.839442e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.000000e-006,2.268761e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,5.813961e-003,6.457243e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,5.465167e-003,1.000000e-006,5.410122e-002}
};

double GetP(int model,int symbol,int state,int kind)
{       switch( model ) {
        case HMM_0: return
((kind==0)?(Pr0[state]*Pro0[symbol][state]):(Pn0[state]*Pno0[symbol][
state]));
        case HMM_1: return
((kind==0)?(Pr1[state]*Pro1[symbol][state]):(Pn1[state]*Pno1[symbol][
state]));
        case HMM_2: return
((kind==0)?(Pr2[state]*Pro2[symbol][state]):(Pn2[state]*Pno2[symbol][
state]));
```

```
        case HMM_3: return
((kind==0)?(Pr3[state]*Pro3[symbol][state]):(Pn3[state]*Pno3[symbol][
state]));
        case HMM_4: return
((kind==0)?(Pr4[state]*Pro4[symbol][state]):(Pn4[state]*Pno4[symbol][
state]));
        case HMM_5: return
((kind==0)?(Pr5[state]*Pro5[symbol][state]):(Pn5[state]*Pno5[symbol][
state]));
        case HMM_6: return
((kind==0)?(Pr6[state]*Pro6[symbol][state]):(Pn6[state]*Pno6[symbol][
state]));
        case HMM_7: return
((kind==0)?(Pr7[state]*Pro7[symbol][state]):(Pn7[state]*Pno7[symbol][
state]));
        case HMM_8: return
((kind==0)?(Pr8[state]*Pro8[symbol][state]):(Pn8[state]*Pno8[symbol][
state]));
        case HMM_9: return
((kind==0)?(Pr9[state]*Pro9[symbol][state]):(Pn9[state]*Pno9[symbol][
state]));
        case HMM_M: return
((kind==0)?(PrM[state]*ProM[symbol][state]):(PnM[state]*PnoM[symbol][
state]));
        case HMM_BEL: return
((kind==0)?(PrBel[state]*ProBel[symbol][state]):(PnBel[state]*PnoBel[
symbol][state]));
        case HMM_OPNIEUW: return
((kind==0)?(PrOpnieuw[state]*ProOpnieuw[symbol][state]):(PnOpnieuw[st
ate]*PnoOpnieuw[symbol][state]));
        }
}

void Rescale(int model,int col) {
        int i ; scaling[col]=0.0;
        for(i=0;i<NS[model];i++) scaling[col]+=alphas[col][i];
        for(i=0;i<NS[model];i++) alphas[col][i]/=scaling[col];
}

double SetAlpha(int model, WORD *seq) {
        double accum; int i,j;
        for(i=0;i<TRELLIS;i++)
                for(j=0;j<NS[model];j++) alphas[i][j]=0.0;
        alphas[0][0]=1.0;
        Rescale(model,0);
        for(i=0;i<LENGTH;i++) {
                for(j=NS[model]-1;j>0;j--) {
                        accum=alphas[i][j]*GetP(model,seq[i],j,0);
                        alphas[i+1][j]=alphas[i][j-
1]*GetP(model,seq[i],j-1,1)+accum;
                }
                alphas[i+1][0]=alphas[i][0]*GetP(model,seq[i],0,0);
                Rescale(model,i+1);
        }
        return alphas[LENGTH][NS[model]-1];
}

double Score( int model,WORD *seq ) {
        double log_alpha; int i ;
        log_alpha=log(SetAlpha(model,seq));
        if(scaling[0]>0.0)
                for(i=0;i<TRELLIS;i++) log_alpha+=log(scaling[i]);
        return log_alpha;
}

int FinalCfr(WORD *sequence) {        double score[MODELS],max;
        int i,best,best2nd ;
        for(i=0,max=-10.0E10;i<MODELS;i++) {
                score[i]=Score(i,sequence);
                if(score[i]>max) {max = score[i];best=i;}
        }
        if(max<MIN_SCORE) return (-best);
        for(i=0,max=-10.0E10;i<MODELS;i++)
                if(score[i]>max && i!=best) {max=score[i];best2nd=i;}
        if( (score[i]-max)<MIN_DSCORE ) return (-best);
        return best;
}
```

Appendix 1.4: Source code of a single model

Header file (Bel.h)

```
/*********************************************************/
/*    Model     : Bel                                    */
/*    Source    : D:\Temp\Afstuderen\Classifier\Final\Bel.hmm*/
/*    Copyright : M. van Veelen (C)1997                  */
/*********************************************************/
#ifndef BEL
#define BEL

#ifndef WORD
#define WORD unsigned int
#endif

#define BEL_LENGTH    16
#define BEL_STATES    7
#define BEL_SYMBOLS   24
#define BEL_TRELLIS   17
#define BEL_WANTMEM   119

extern double TestBel(WORD *sequence);

#endif
/*        <<< End of header code for Bel >>>              */
```

Implementation (Bel.c)

```
/*********************************************************/
/*   Implementation part of model BEL                    */
/*********************************************************/
#include "Bel.h"
#include <math.h>

double PrBel[7]={7.895551e-001,4.509694e-001,4.014443e-001,2.426600e-
001,1.286838e-001,3.209441e-001,1.000000e+000};

double PnBel[7]={2.104449e-001,5.490306e-001,5.985557e-001,7.573400e-
001,8.713162e-001,6.790559e-001,1.000000e-006};

double ProBel[24][7]={
        {6.054029e-001,1.290406e-002,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,5.479119e-001},
        {1.783466e-001,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,2.013664e-001},
        {6.562191e-002,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.124495e-001},
        {1.825626e-002,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,6.246186e-003},
        {7.500873e-002,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,9.860105e-001,4.854795e-003},
        {9.310562e-003,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,4.571498e-003},
        {1.414210e-002,1.000000e-006,1.000000e-006,3.137790e-
002,1.000000e-006,1.000000e-006,1.371883e-002},
        {4.136883e-003,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,4.909249e-003},
        {2.882964e-003,1.000000e-006,1.515552e-001,1.000000e-
006,1.000000e-006,1.398713e-002,1.000000e-006},
        {3.247513e-003,3.460959e-001,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,2.267504e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.000000e-006},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.000000e-006},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.000000e-006},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.000000e-006},
        {1.000000e-006,1.621509e-002,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,3.066603e-003},
        {1.000000e-006,6.197779e-002,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.676216e-002},
        {6.505457e-003,1.000000e-006,1.000000e-006,3.720871e-
001,1.000000e-006,1.000000e-006,1.000000e-006},
        {1.713737e-002,1.000000e-006,1.000000e-006,1.000000e-
006,9.999946e-001,1.000000e-006,1.000000e-006},
        {1.000000e-006,1.000000e-006,1.000000e-006,5.671642e-
001,1.000000e-006,1.000000e-006,3.051529e-003},
        {1.000000e-006,2.639583e-002,6.802158e-001,2.936741e-
002,1.000000e-006,1.000000e-006,1.236830e-002},
        {1.000000e-006,2.092240e-001,1.682238e-001,1.000000e-
006,1.000000e-006,1.000000e-006,2.148054e-002},
        {1.000000e-006,1.607270e-001,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,4.571500e-003},
        {1.000000e-006,1.664570e-001,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.999555e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,1.000000e-006}
};

double PnoBel[24][7]={
        {1.055920e-001,2.293675e-002,1.243228e-002,2.334151e-
002,2.507113e-002,7.189908e-002,8.289612e-002},
        {7.794787e-002,9.152474e-002,1.178373e-001,1.884761e-
001,2.082944e-001,5.412985e-001,9.443861e-003},
        {2.320222e-001,1.215411e-001,8.281883e-002,1.098020e-
002,1.885843e-002,2.133195e-002,9.024134e-002},
        {3.822177e-002,4.328985e-003,1.000000e-006,5.526999e-
003,1.075804e-002,1.000000e-006,7.450157e-002},
        {2.898694e-002,3.090015e-002,4.948472e-002,7.891411e-
002,3.485022e-001,1.391079e-001,1.049318e-003},
```

```
        {6.076129e-003,5.503766e-003,1.000000e-006,1.000000e-
006,5.813641e-003,1.000000e-006,3.462749e-002},
        {1.095730e-001,7.225189e-003,6.714471e-002,5.525175e-
002,1.000000e-006,1.000000e-006,1.678909e-002},
        {3.600830e-002,5.337421e-002,7.686420e-002,8.741192e-
002,6.692847e-002,4.632382e-002,1.154250e-002},
        {6.955273e-003,5.692100e-002,9.563560e-003,1.144972e-
002,1.697033e-002,1.743704e-002,3.147954e-003},
        {6.552280e-002,9.691112e-002,7.232530e-002,4.928754e-
002,1.298106e-002,6.040412e-003,9.863589e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,8.394544e-003},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,8.394544e-003},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,5.351522e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,2.203568e-002},
        {3.131975e-002,7.682629e-003,1.000000e-006,5.744698e-
003,1.000000e-006,1.000000e-006,2.308499e-002},
        {2.960845e-002,2.204911e-002,2.534066e-002,1.000000e-
006,1.000000e-006,1.000000e-006,8.499475e-002},
        {1.047631e-002,1.000000e-006,6.750474e-003,1.303630e-
001,4.017702e-002,1.000000e-006,3.567681e-002},
        {1.163104e-002,4.070594e-002,7.135973e-002,2.265406e-
001,1.526725e-001,9.324387e-002,7.345226e-003},
        {1.244025e-002,3.599872e-002,1.156179e-001,2.049147e-
002,1.752575e-002,1.153388e-002,5.876180e-002},
        {7.320233e-002,1.735221e-001,2.463850e-001,7.117705e-
002,5.230368e-002,3.496262e-002,8.184680e-002},
        {9.121095e-002,1.500665e-001,2.055004e-002,1.000000e-
006,1.162426e-002,1.107422e-002,8.289612e-002},
        {7.670133e-003,2.885228e-002,5.875275e-003,1.744190e-
002,1.000000e-006,1.000000e-006,5.981112e-002},
        {2.553459e-002,4.995567e-002,1.964987e-002,1.760127e-
002,1.151833e-002,5.746444e-003,2.518363e-002},
        {1.000000e-006,1.000000e-006,1.000000e-006,1.000000e-
006,1.000000e-006,1.000000e-006,2.518363e-002}
};

#define PRSBEL(i,j)    (ProBel[i][j]*PrBel[j])
#define PNSBEL(i,j)    (PnoBel[i][j]*PnBel[j])

#ifndef SHAREDALPHA
double scalingBel[BEL_TRELLIS] ;
double alphaBel[BEL_TRELLIS][BEL_STATES] ;
#else
#define  alphaBel     alphaShared
#define  scalingBel scalingShared
#endif

void RescaleBel(int col)
{       int i;
        scalingBel[col]=0.0;
        for(i=0;i<BEL_STATES;i++) scalingBel[col]+=alphaBel[col][i];
        for(i=0;i<BEL_STATES;i++) alphaBel[col][i]/=scalingBel[col];
}

double SetAlphaBel(WORD *seq)
{       double accum  ;
        int            i,j            ;
        for(i=0;i<BEL_TRELLIS;i++)
                for(j=0;j<BEL_STATES;j++) alphaBel[i][j]=0.0;
        alphaBel[0][0]=1.0;
        RescaleBel(0);
        for(i=0;i<BEL_LENGTH;i++) {
                for(j=BEL_STATES-1;j>0;j--) {
                        accum=alphaBel[i][j]*PRSBEL(seq[i],j) ;
                        alphaBel[i+1][j]=alphaBel[i][j-
1]*PNSBEL(seq[i],j-1)+accum ;
                }
                alphaBel[i+1][0]=alphaBel[i][0]*PRSBEL(seq[i],0) ;
                RescaleBel(i+1) ;
        }
        return alphaBel[BEL_LENGTH][BEL_STATES-1] ;
}

double TestBel(WORD *sequence)
{       double log_alpha ;
        int            i ;
        log_alpha = log(SetAlphaBel(sequence));
        if(scalingBel[0]>0.0){
                log_alpha = 0.0 ;
                for(i=0;i<BEL_TRELLIS;i++)
                        log_alpha+=log(scalingBel[i]);
        }
        return log_alpha ;
}
```