HIDDEN MARKOV MODELS FOR AUTOMATIC SPEECH RECOGNITION

AND THEIR MULTIMODAL APPLICATIONS

Pascal Wiggers

August 2001

Knowledge Based Systems Group
Department for Technical Mathematics and Computer Science
Faculty of Information Technology and Systems
Delft University of Technology

Thesis committee:
Prof. dr. H. Koppelaar
Prof. dr. ir. E. J. H. Kerckhoffs
Dr. drs. L. J. M. Rothkrantz

TU Delft
Delft University of Technology

ACKNOWLEDGEMENTS

This report was made possible with the help of the Knowledge Based Systems Group of the Department for Technical Mathematics and Computer Science of the Delft University of Technology.
I would like to thank Leon Rothkrantz, who supervised and supported my work and with whom I had many interesting discussions that were not limited to the topic of speech recognition.
Further I would like to Thank Jacek Wojdel, who has helped me with my work and whose insights helped to solve many problems that occurred during the project.

ABSTRACT

State of the art automatic speech recognition systems reach acceptable levels of performance when used under laboratory conditions. In more realistic noisy environments however, their performance rapidly degrades. A possible solution to this problem lies in the use of multiple modalities for speech recognition; the audio signal is augmented by for example lipreading signals or information on facial expressions.
Open question is how and at what point during the recognition process to integrate multiple modalities in a speech recognizer.
This report describes the development of a large vocabulary speaker independent continuous speech recognizer for the Dutch language using Hidden Markov Toolkit and the Polyphone database of recorded Dutch speech. This recognizer can be used as a starting point for a multimodal recognizer based on multimodal Hidden Markov models. Furthermore a number of models for multimodal recognition are presented and a number of experiments on the incorporation of other modalities in the speech recognizer are described and tested.

TABLE OF CONTENTS

1 INTRODUCTION

Speech is the most natural means of communication between humans; it can be done without any tools or any explicit education. It is one of the first skills we learn to use. Babies quickly learn how to react to the voice of their mother and they even quicker learn to produce noise when they are in need. When speaking with somebody, one does not have to focus on this person; one can look in a different direction or even perform some other task while communicating.
Speech is also the most important way of communicating. It has always been; before mankind invented writing, the spoken word was the only way of passing knowledge. Ancient poets like Homer and Ovidius originally wrote their still famous epic poems for recitation not for reading. Despite all our novel ways of communicating, like e-mail and chat, speech is still the number one means of communication, a fact once again proven by the immense popularity of cellular phones.

So it is only logical that machine interface designers in their quest for a natural man-machine interface have turned to automatic speech recognition and speech production as one of the most promising interfaces. In the last 30 years researchers from areas like psychology, linguistics, electrical engineering and computer science have worked on this subject. While the first systems could only differentiate between 'yes' and 'no', currently, some professional tools for automatic speech recognition (ASR) are commercially available from leading companies like IBM, Lernhout & Hauspie and Philips.

The Holy Grail of speech recognition is a speaker independent system that recognizes in real-time natural, fluent, spontaneous, continuous speech and which is capable of telling speech and background noise apart. The current generation of systems is nowhere near this ideal. The state of the art in the field of speech recognition is defined by two types of systems. In the first place dictation systems that are capable of automatically transcribing dictated texts. These systems rely heavily on the fact that written texts adhere strictly to the rules of grammar and they do not allow for hesitations or mispronunciations. The second type of recognition system is the category of command recognizing and dialog based systems, like telephone inquiry applications that are commercially available now. In these systems the interaction between human and machine is guided by a dialog and at any point in time only a limited vocabulary is used. These systems usually try to spot some keywords from this vocabulary instead of trying to recognize complete sentences. Both types of systems perform well as long as they are used exactly by the book in a silent environment. However, performance rapidly degrades in (more realistic) noisy environments.

Humans seem to recognize speech not only by listening, but also by using clues from other modalities, like gestures, facial expressions and lip movements. This becomes especially evident in noisy environments where people tend to look at the face of a speaker during a conversation. An open question is whether it would be possible to create better and more robust speech recognizing systems by integrating information from other modalities in the recognition system. Or more general how to build multimodal automatic communication systems.

In the group knowledge based systems at Delft University of Technology there is a project running on the development of multimodal interfaces. A number of systems are being developed:

- A system for automatic recognition of facial expressions.

- A system for automatic generation of facial expressions.
- A system for automatic lipreading.

To integrate unimodal systems, it's necessary to have unimodal systems. The group invested much research in Dialog Management but was lacking an ASR-system for the Dutch Language. Up to now an executable of Philips Speech Mania has been used, which is not very satisfactory since its source cannot be changed.
There was a need to develop a speech recognizer and to find a way to integrate this recognizer with other modalities. The goal of the current thesis project is:

- To conduct a literature study on automatic speech recognition in general and to determine how to develop an ASR-engine.
- To study HTK (Hidden Markov Toolkit), a toolkit that supports the development of automatic recognition systems based on Hidden Markov Models. It had to be determined how the toolkit works and how it can be utilized in the development of an ASR-system.
- To develop an ASR-engine, using HTK that can be used for future projects.
- To develop a mixed Hidden Markov model for multimodal recognition.
- To test the mixed HMM.

This report describes the results of the master thesis project. The second chapter describes speech recognition in general. Chapter three gives an overview of the Hidden Markov toolkit that was used to develop an automatic speech recognition system. The development of this speech recognizer is described in chapter four.
Chapter five describes the theory behind the integration of unimodal recognition systems in a multimodal recognition system. It presents a number of possible integration techniques. Chapter six describes a number of experiments that were conducted using the multimodal models.

2 SPEECH RECOGNITION

This chapter describes the theory of speech recognition. It starts with a description of the difficulties in recognizing speech. Next an overview of the entire process is given and the subsystems of a recognizer are identified. Each of these components is then described. Especially the Hidden Markov Model, which constitutes the core of the recognition process, is described in detail.

2.1 INTRODUCTION

Why is automatic speech recognition such a difficult problem that, after 30 years of research, it is still not solved? At first sight it may seem just a matter of classifying sounds using some typical characteristics of these sounds. This approach, called the acoustic approach was indeed tried, but only with limited results. Finding explicit characteristics of speech sounds that suffice to classify them proved extremely hard.
A second approach to recognizing, that does not directly rely on a set of characteristics is the statistical pattern recognition approach, which has already been successfully been applied to problems like the automatic recognition of handwriting. The pattern recognition approach did prove to be fruitful, but only after advanced models were developed.

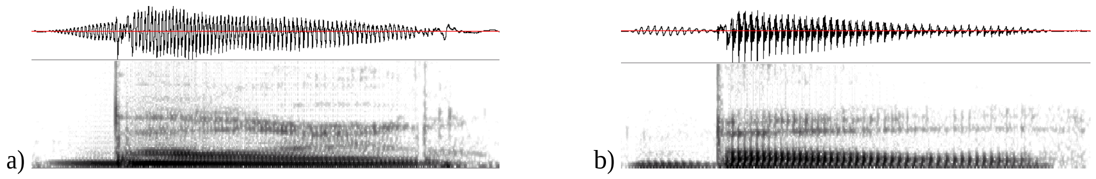


Figure 2.1 - The word *ball* spoken by two different speakers: (a) female and (b) male

Figure 2.1 shows why the simple pattern recognition approach of classifying signals with a similar shape as the same sound, is not powerful enough to perform speech recognition. The figures show the word *ball* spoken by two different persons. The upper half of the figure shows the raw speech waveform, the lower half shows processed versions of the signal that highlight its formants, which are characteristic for a sound. So speech is person dependent, which is no surprise, as different voices sound different. Figure 2.2 shows the phoneme /e/ spoken by the same person in a number of words.

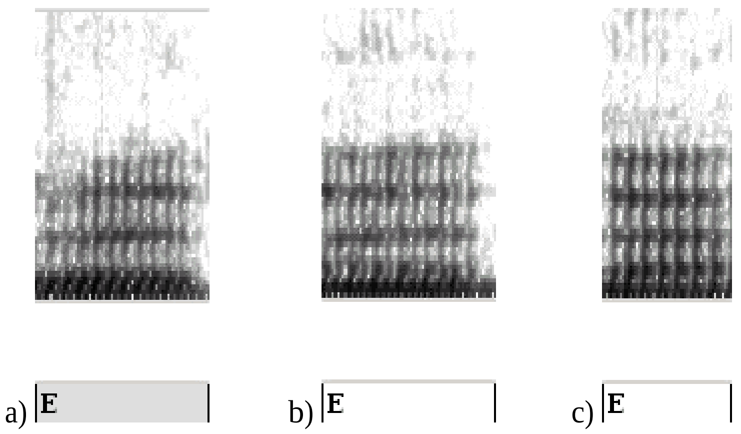


Figure 2.2 - The phoneme /e/ in three different contexts: (a) *let's*, (b) *phonetic* and (c) *sentence*.

These pictures look very different. In this case actually context is involved, the exact sound of a speech unit, called a phoneme, depends on its neighboring phonemes. The exact shape of the speech signals also depends on the speed with which is spoken and the mood and the temper of the speaker.

2.2 FUNDAMENTALS OF SPEECH RECOGNITION

2.2.1 Components of a speech recognizer

Figure 2.3 schematically shows the speech recognition problem: someone produces some speech and we want to have a system (the box in the figure) that automatically translates this speech, a pressure waveform that is, to a written transcription.

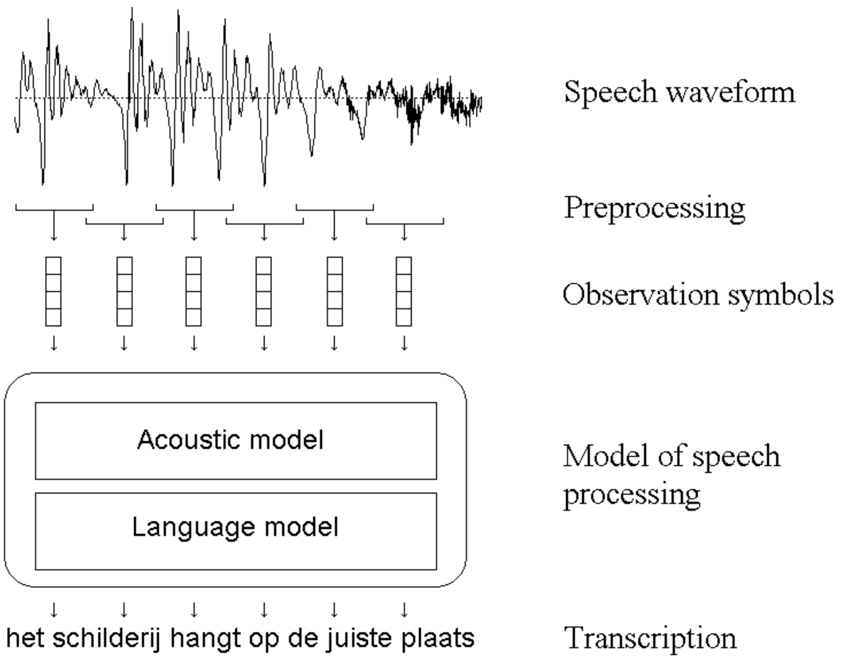


Figure 2.3 - overview of the speech recognition problem

It is possible, after some preprocessing, to represent the speech signal as a sequence of observation symbols $\mathbf{O} = o_1o_2\ldots o_T$ that belong to a set $\mathbf{O}$ of symbols. If, in addition, we have a vocabulary V of all the words $w_i,\ 1 \le i \le |V|$, that can be uttered. Then mathematically the speech recognition problem comes down to finding the word sequence $\hat{\mathbf{W}}$ having the highest probability of being spoken, given the acoustic evidence $\mathbf{O}$, thus we want to solve

$$\hat{\mathbf{W}} = \arg\max_{\mathbf{W}} P(\mathbf{W}|\mathbf{O}) \tag{2.1}$$

Unfortunately, this equation is not directly computable since the number of possible observation sequences is sheer inexhaustible, unless there is some limit on the duration of the utterances and there is a limited number of observation symbols. But Bayes formula gives:

$$P(\mathbf{W}|\mathbf{O}) = \frac{P(\mathbf{W})P(\mathbf{O}|\mathbf{W})}{P(\mathbf{O})} \tag{2.2}$$

Where $P(\mathbf{W})$, called the language model, is the probability that the word string $\mathbf{W}$ will be uttered and $P(\mathbf{O}|\mathbf{W})$ is the probability that when word string $\mathbf{W}$ is uttered the acoustic evidence $\mathbf{O}$ will be observed, the latter is called the acoustic model. The probability $P(\mathbf{O})$ is usually not known but for a given utterance it is of course just a normalizing constant and can be ignored. Thus to find a solution to formula (2.1) we have to find a solution to:

$$\hat{\mathbf{W}} = \arg\max_{\mathbf{W}} P(\mathbf{W})P(\mathbf{O}|\mathbf{W}) \tag{2.3}$$

Consequently, a speech recognizer consists of three components: a preprocessing part that translates the speech signal into a sequence of observation symbols, a language model that tells us how likely a certain word string is to occur and an acoustic model that tells us how a word string is likely to be pronounced. In the next sections these three subsystems will be described.

2.2.2 Acoustic processing

The first step in speech recognizer design is to decide what acoustic data **O** will be observed. Therefore a front end is needed that will transform the original waveform into a sequence of symbols o_i with which the recognizer will deal. Strictly speaking this is not necessary, speech recognition could be done by performing pattern recognition algorithms directly on the speech signal, as this signal contains all information. But as mentioned before there are many possible variations in a speech signal and visually similar waveforms do not necessarily indicate perceptually similar sounds. Therefore some preprocessing may be useful to reduce the amount of noise introduced by the environment and the recording hardware and to reduce correlation in the input signal and to extract relevant features.
Many different ways to extract meaningful features have been developed, some based on acoustic concepts or knowledge of the human vocal tract and psychophysical knowledge of the human perception. Much work is done in the field of signal processing; the most important methods here are Linear Predictive Coding and Mel Frequency Cepstral Analysis. An impression of these two techniques is given below.

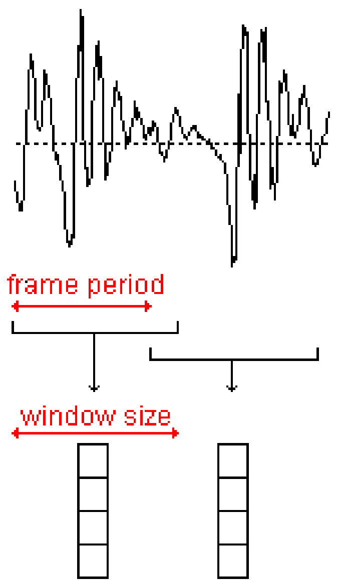


Figure 2.4 - feature extraction

Figure 2.4 illustrates the overall feature extraction process. A sampled waveform is converted into a sequence of parameter vectors at a certain frame rate. A framerate of 10 ms is usually taken, because a speech signal is assumed to be stationary for about 10 ms. The segment of a waveform that is used to determine each parameter vector is referred to as a window. The window size and the frame rate are independent. Normally, the window size will be larger than the frame rate, so that successive windows will overlap to make up for discontinuity in the signal introduced by the discrete sampling of the signal.

2.2.2.1 Linear predictive coding

Linear predictive coding is one of the most useful methods for encoding good quality speech at a low bit rate. LPC starts with the assumption that the speech signal is produced by a buzzer at the end of a tube. The space between the vocal chords, called the glottis, produces the buzz, which is characterized by its intensity (loudness) and frequency, which determines the pitch of the sound. The vocal tract, that is the combination of the throat and the mouth, forms a tube, which is characterized by its resonances, called formants.
LPC analyzes the speech signal frames by estimating the formants, removing their effects from the speech signal, and estimating the intensity and frequency of the remaining buzz. The process of removing the formants is called inverse filtering, and the remaining signal is called the residue. The basic problem of LPC is to determine the formants from the speech signal. The solution is a difference equation, called a linear predictor, which expresses each sample of the signal as a linear combination of previous samples. The coefficients of the difference equation, the prediction coefficients, characterize the formants. These coefficients are estimated by minimizing the mean-square error between the predicted signal and the actual signal.

2.2.2.2 Mel-frequency cepstral analysis

One of the more common techniques of studying a speech signal is via the power spectrum. The power spectrum of a speech signal describes the frequency content of the signal over time. The first step towards computing the power spectrum of the speech signal is to perform a Discrete Fourier Transform (DFT). A DFT computes the frequency information of the equivalent time domain signal. Since a speech signal contains only real point values, we can make use of this fact and use a real-point Fast Fourier Transform (FFT) for increased efficiency. The resulting output contains both the magnitude and phase information of the original time domain signal.
Psychophysical studies have shown that human perception of the frequency content of sounds, either for pure tones or for speech signals, does not follow a linear scale. This research has led to the idea of defining subjective pitch of pure tones. Thus for each tone with an actual frequency *f*, measured in Hz, a subjective pitch is measured on a scale called "Mel" scale. As a reference point, the pitch of a 1kHz tone, 40 dB above the perceptual hearing threshold, is defined as 1000 Mels. Other subjective pitch values are obtained by adjusting the frequency of a tone such that it is half or twice the perceived pitch of a reference tone with a known Mel frequency. Feature extraction based on Mel Frequency Cepstral Coefficients (MFCC) utilizes the power spectrum of which center frequency and bandwidth are scaled by subjective measure, Mel. The cepstral coefficients are then computed by taking the logarithm of the power spectrum and transforming this log spectrum to the cepstral domain using an inverse Discrete Fourier Transform. This signal can be further decorrelated by taking the cosine of the signal. Usually, first and second derivatives are taken from the cepstral coefficients and added to the speech vector.

2.2.3 Language modeling

Formula 2.3 requires that we are able to compute for every word string **W** the a priori probability $P(\mathbf{W})$ that the speaker wishes to utter **W**. As **W** is a string, these probabilities can be decomposed as follows:

$$P(\mathbf{W}) = \prod_{i=1}^{m} P(w_i | w_1, w_2, ..., w_{i-1}) \qquad (2.4)$$

Where $P(w_i | w_1, w_2, ..., w_{i-1})$ is the probability that w_i will be spoken given that the words $w_1 w_2 ... w_{i-1}$ were said previously. The choice of w_i thus depends on the entire history of the discourse sofar. In reality it is impossible to estimate these probabilities even for moderate values of *i* as most of these histories would be unique. For a vocabulary of size $|V|$ there are $|V|^{i-1}$ different histories. for example even for a relatively small vocabulary of 1000 words and $i=3$ there would be one billion different histories. The number of probabilities to compute can be reduced drastically by considering only limited histories, for example using only the last three words, will effectively map the histories to a limited number of equivalence classes. Doing so will preserve most of the model's ability to predict the next word, as most words do not really depend on a large history. Besides, as every language has a grammar, which sets rules to the word order, for a large vocabulary most of the possible word strings will never occur.
A simple but effective approach is the bigram, this model uses only one step histories.

$$P(\mathbf{W}) = P(w_1) P(w_2 | w_1) .... P(w_m | w_{m-1}) \qquad (2.5)$$

It is now easy to estimate the probabilities $P(w_i | w_{i-1})$ simply by counting the number of times each word pair occurs in a representative corpus of strings. One problem with this procedure is that the training corpus might be missing some words, which do occur in the vocabulary the recognizer uses. To these words would be assigned an estimated probability of zero. Therefore, usually a small portion of the probability distribution is reserved for words out of the vocabulary that do not occur in the training corpus.
Most present-day speech recognizers take it one step further and use the trigram language model that has two word history equivalence classes. This language model is surprisingly powerful but requires large amounts of training data to provide values for all probabilities $P(w_i | w_{i-1}, w_{i-2})$ as can be seen in the above example. Furthermore the model tends to get very large as all the words have to be listed and for each word all possible two-word histories must be provided.
To circumvent these problems some extensions to the basic statistical language models are possible. To make up for data sparseness the trigram frequencies can be smoothed by interpolating trigram, bigram and unigram relative frequencies.

$$P(w_i | w_{i-1}, w_{i-2}) = \lambda_1 f(w_i) + \lambda_2 f(w_i | w_{i-1}) + \lambda_3 f(w_i | w_{i-1}, w_{i-2}) \qquad (2.6)$$

Where the weights satisfy $\lambda_1 + \lambda_2 + \lambda_3 = 1$.
Another, similar method to make up for the lack of training data used in many speech recognizers is backing-off. Arguing that if there is enough evidence the trigram frequency is a good estimate for the probability, if not the system should back-off and rely on bigrams and if there is not enough evidence even for those, unigrams should be used. Of course it is possible to

create any *N*-gram, *N* > 3, but with it comes a combinatorial increase in model complexity and size and an unsatisfiable hunger for training data.
The big advantage of statistical language models is that they are relatively simple to obtain. All that is needed are some representative texts and for large vocabulary recognition tasks these models may be the only realistic option. However, creation of a representative corpus of text may be a difficult and time-consuming effort. If there is an existing human-human interface for the area of interest a vocabulary can be created using the transcriptions of these dialogs. If a new ASR application has to be designed for which there is no human counterpart we can simulate speech recognizing systems by using Wizard of Oz studies to create a corpus. For smaller, more specialized tasks or dialogs constraints on the form of the utterances can be found, that is a grammar can be defined. Within this grammar we can still use statistics to attach probabilities to the different paths through the grammar. These grammars offer a powerful language model because of the constraints they impose. However, it is important to note that there is no need for these grammars to be completely accurate. They can be open to counter examples, because their purpose is to distribute probability among different futures and not exact analysis of the utterances (as would be the case in for example speech synthesis). In the example below a grammar for a simple telebanking application, that can be used to manage one's bank account and make financial transactions by telephone, is defined, using EBNF notation.

```
daytime    ::= 'morgen' | 'middag' | 'avond'
greeting   ::= 'goede' daytime
digit      ::= 'een' | 'twee' | 'drie' | 'vier' | 'vijf' | 'zes' | 'zeven'
               |'acht' | 'negen'
digit0     ::= 'nul' | digit
number     ::= digit0 digit0 digit0 digit0 digit0 digit0 digit0 digit0
tenfolds   ::= 'twintig' | 'dertig' | 'veertig' | 'vijftig' | 'zestig' |
               'zeventig' | 'tachtig' | 'negentig'
hondreds   ::= [ digit ] 'honderd'
amount     ::= [hondreds] [ digit 'en' ] tenfolds
please     ::= 'alstublieft' | 'alstjeblieft' | 'graag' | 'willen'
want       ::=  'ik' ('wil' | 'wilde' | 'zou graag')
mine       ::= ['van'] 'mijn'
account    ::= ['prive'] ['bank'] rekening
specific   ::= [mine] account [['nummer'] number]
money      ::= ( ['wat'] 'geld' ) | (amount ('guldens' | 'piek'))
           open       ::= 'een nieuwe' account 'openen' [please]
close      ::= specific 'sluiten'
deposit    ::= money ( ( 'op' specific 'storten' ) |
                       ( 'storten op' specific) ) ;
withdrawal ::= money ( ( 'van' specific 'opnemen' ) |
                       ('opnemen van' specific) )
transfer   ::= money 'van' specifict ['naar' specific] 'overmaken'
info       ::= ( 'saldo van' specific 'opvragen') |
               ('naar het saldo van' specific 'informeren')
command ::= [greeting] want ( open | close | deposit | withdrawal |
            transfer | info) [please]
```

Figure 2.5 - a EBNF grammar for a telebanking application

The application is intended to recognize commands like:

```
"Goede morgen, ik zou graag drieenveertig guldens van mijn priverekening
opnemen, alstublieft"

"Ik wilde honderdzevenentwintig guldens overmaken naar bankrekening een
twee drie vier vijf zes zeven acht"
```

But a closer inspection of the grammar reveals that it also allows for sentences like:

```
"Goede middag, ik graag nul piek op van mijn rekening storten, graag"
```

that do not conform to the rules of Dutch grammar. Semantically this sentence does not make much sense either. But the point is that this sentence is highly unlikely ever to be uttered, so there is no need to make the grammar overly complicated just to prevent this sentence from occurring. And actually, it is not even desirable, as spoken language does not adhere to grammar rules per se. Making them too restrictive may lead to even bigger problems, because the system then starts recognizing things that were never said. For example, in the telebanking application the following sentence is not unlikely to be spoken:

```
"Goede morgen, uh…, goede middag, ik zou graag drieentwintig, nee toch maar
vijftig gulden naar de bank rekening van de TU Delft over willen maken."
```

Because of the hesitations and some unknown words the above grammar cannot recognize this sentence. It will, however, recognize parts of the sentence and may for example conclude that the caller wants to transfer twenty-three guilders to some bank account. It will then try to fit the sounds in `TU Delft` to some account number. Thus, the judgment whether a sentence makes sense is better left up to the application that uses the speech interface.

2.2.4 Acoustic modeling

The acoustic model determines what sounds will be produced when a given string of words is uttered. Thus for all possible combinations of word strings **W** and observation sequences **O** the probability $P(\mathbf{O}|\mathbf{W})$ must be available. This number of combinations is just too large to permit a lookup, in the case of continuous speech its even infinite. It follows that these probabilities must be computed on the fly, so a statistical acoustic model of the speakers' interaction with the recognizer is needed. The total process modeled involves the way the speaker pronounces the words of **W**, the ambience (room) noise, the microphone placement and characteristics and the acoustic processing performed by the signal processing front end. The most frequently used model these days is the Hidden Markov model but other models are possible, for instance artificial neural networks. The next paragraph describes the Hidden Markov model in detail.

2.3 HIDDEN MARKOV MODELS

2.3.1 Introduction

The Hidden Markov model (HMM) is a very powerful mathematical tool for modeling time series. It provides efficient algorithms for state and parameter estimation, and it automatically performs dynamic time warping for signals that are locally squashed and stretched. It can be used for many purposes other than acoustic modeling.
Hidden Markov models are based on the well-known Markov chains from probability theory that deal with a class of random processes having one-step memory. That is, they have the so-called Markov property: the probability that the system will be in a certain state at the next time step only depends on the current state. A Markov chain can be viewed as a finite state process with a probabilistic transition function. In the Markov model each state corresponds to one observable event. But this model is too restrictive, for a large number of observations the size of the model explodes, and the case where the range of observations is continuous is not covered at all. Therefore the Hidden Markov concept extends the model by making the observation a probabilistic function of the state, this allows more freedom to the random process while

avoiding a substantial complication to the basic structure of Markov chains. The resulting model is a doubly embedded stochastic process in which the underlying stochastic process is not directly observable but can be observed only through another set of stochastic processes that produce the sequence of observations. The Hidden Markov model is defined as follows:

1. The number of states N.
2. The number of distinct observation symbols M.
3. An output alphabet $\mathrm{O} = \{\mathrm{o}_1, \mathrm{o}_2, ...\mathrm{o}_M\}$.
4. A state space $\mathrm{Q} = \{q_1, q_2, ...q_n\}$.
5. A probability distribution of transitions between states $\mathbf{A} = \{a_{ij}\}$ where
   $a_{ij} = P(q_{t+1} = j \mid q_t = i) \quad 1 \le i, j \le N$
6. An observation symbol probability distribution $\mathbf{B} = \{b_j(k)\}$ in which
   $b_j(k) = P(o_t = \mathrm{o}_k \mid q_t = j) \quad 1 \le k \le C, 1 \le j \le N$
7. The initial state distribution $\pi = \{\pi_i\}$ where $\pi_i = P(q_1 = i) \quad 1 \le i \le N$
8. To indicate the complete parameter set of a model the notation $\lambda = (\mathbf{A}, \mathbf{B}, \pi)$ is used.

The observation probability distributions are usually all defined on the same domain, so in theory each state is capable of generating every possible observation symbol.

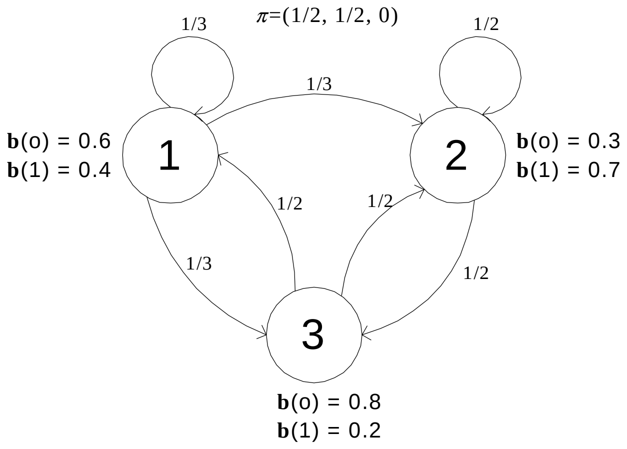


Figure 2.6 - a simple discrete Hidden Markov model

Figure 2.6 shows a three state HMM with discrete probability distributions attached to each state, capable of generating the symbols 0 and 1.

2.3.2 HMM generator of observations

Given appropriate values of *N*, *M*, **A**, **B**, π and an alphabet the HMM can be used as a generator to give an observation sequence $\mathbf{O} = o_1 o_2 \ldots o_T$ by performing the following steps:

1. Choose an initial state $q_1 = i$ according to the initial state distribution π.
2. Set $t = 1$.
3. Choose $o_t = \mathrm{o}_k$ according to $b_j(k)$.

4. $t = t + 1$, transit to a new state $q_{t+1} = j$ according to a_{ij}.
5. Return to step 3 if $t < T$.

But if recognition is to be performed with a Hidden Markov model the observation sequence is already available and we want to know how well a given model matches a given observation sequence, that is we want to know the probability that the observation sequence was produced by the model λ.
Now, when we have an observation sequence and know which model is most likely to generate the sequence, it is not clear how the model generates that sequence, because the underlying state sequence is hidden. Since practically any state sequence could generate each observation sequence there is no correct state sequence to be found here. All we can do is try to solve this problem as best as possible and seek the most likely state sequence given the observation sequence and the model.
Another important issue that has to be addressed is how to obtain and optimize the model parameters to maximize $P(\mathbf{O}|\lambda)$. This is called the training problem. The next three sections discuss each of these fundamental questions concerning Hidden Markov models in turn.

2.3.3 Forward algorithm

When we consider a fixed state sequence $\mathbf{q}$ of length T, the probability of the observation sequence given the model λ and sequence $\mathbf{q}$ is stated by

$$P(\mathbf{O}|\mathbf{q},\lambda) = \prod_{t=1}^{T} P(o_t|q_t,\lambda) = b_{q_1}(o_1)b_{q_2}(o_2)...b_{q_T}(o_T) \tag{2.7}$$

(Assuming statistical independence of observations).

There are N^T possible state sequences $\mathbf{q}$, each with probability:

$$P(\mathbf{q}|\lambda) = \pi_{q_1}a_{q_1q_2}a_{q_2q_3}...a_{q_{T-1}q_T} \tag{2.8}$$

The probability of the observation sequence $\mathbf{O}$ given the model is now obtained by taking the joint probability of 2.7 and 2.8 and summing over all possible state sequences, giving:

$$P(O|\lambda) = \sum_{\forall\mathbf{q}} P(\mathbf{q}|\lambda)\prod_{t=1}^{T} P(o_t|q_t,\lambda) \tag{2.9}$$

$$= \sum_{q_1,q_2,...,q_N} \pi_{q_1}b_{q_1}(o_1)a_{q_1q_2}b_{q_2}(o_2)...a_{q_{T-1}q_T}b_{q_T}(o_T) \tag{2.10}$$

As can be seen from equation 2.10 this calculation of $P(\mathbf{O}|\lambda)$ involves on the order of $2TN^t$ calculations since at every time t=1, 2, ... T there are N^t possible state sequences and for each such state sequence about $2T$ calculations are required. So this calculation is computationally infeasible. Therefore a more efficient procedure is needed. Fortunately, such a procedure exists; it is called the forward procedure. For this procedure the probability $\alpha_t(i)$ is defined as the probability of being in state *i* at time *t* and having observed the partial observation sequence $o_1 o_2 \ldots o_t$ sofar, given the model λ.

$$\alpha_t(i) = P(o_1 o_2 ... o_t, q_t = i | \lambda) \tag{2.11}$$

Now $\alpha_t(i)$ can be computed inductively, as follows:

1. initialization

$$\alpha_1(i) = \pi_i b_i(o_1) \quad 1 \le i \le N \tag{2.12}$$

2. Induction

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij} \right] b_j(o_{t+1}) \quad 1 \le t \le T-1, t \le j \le N \tag{2.13}$$

3. Termination:

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{2.14}$$

Step one initializes the forward probabilities as the joint probability of state *i* and initial observation o_1. The induction step follows from the fact that the product $\alpha_i(t) a_{ij}$ is the probability of the event that $o_1 o_2 ... o_t$ are observed and state *j* is reached at time *t*+1 via state *i* at time *i*. Summing this product over all the *N* possible states, $i \; 1 \le i \le N,$ at time *t* results in the probability of *j* at time *t*+1 with all of the accompanying previous partial observations. Now all that is left to do is to take account for observation o_{t+1} in state *j*, this is done by use of the probability $b_j(o_{t+1})$.

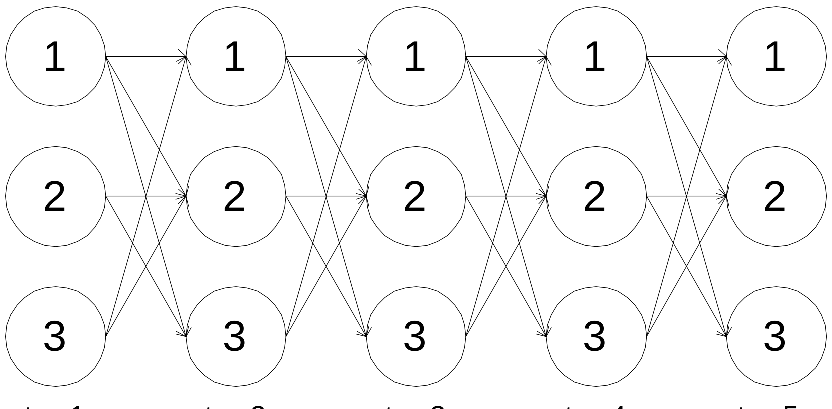


Figure 2.7 - The forward algorithm visualized by a trellis

The forward algorithm can be visualized by a trellis that shows the time evolution of the process. It consists of *T* time stages each having *N* nodes (the model states) connected by arcs that show the possible state sequences. Figure 2.7 shows a trellis for the output sequence 10110 generated by the HMM of figure 2.6. This figure shows the main thought behind the forward algorithm, because there are only *N* nodes at each time slot in the trellis all possible state sequences will

remerge into these N nodes no matter how long the observation sequence. At time t each calculation only involves the N previous values of $\alpha_{t-1}(i)$, because each of the N grid points van be reached from only the N grid points at the previous time slot. So this procedure only requires on the order of N^2T calculations. Rather than $2TN^t$ as required by the direct calculation.
There is one practical remark to make on the forward procedure: the probabilities $P(\mathbf{O} \mid \lambda)$ may become extremely small, in particular if the observation sequence becomes longer or if the observation sequence is very unlikely for the model. To avoid underflow, usually the logarithm of $P(\mathbf{O} \mid \lambda)$ is computed.

2.3.4 Viterbi algorithm

Finding the most likely state sequence $q_1q_2...q_T$ given an observation sequence $\mathbf{O}=o_1o_2...o_T$ boils down to maximizing $P(\mathbf{q}|\mathbf{O},\lambda)$ which is equivalent to maximizing $P(\mathbf{q},\mathbf{O}|\lambda)$. Therefore we define $\delta_t(i)$ as the best score (highest probability) along a single path, at time t, which accounts for the first t observations and ends in state i.

$$\delta_t(i) = \max_{q_1,q_2,..q_{t-1}} P(q_1q_2...q_{t-1}, t = i, o_1o_2...o_t \,|\lambda) \qquad (2.15)$$

We can calculate $\delta_t(i)$ using a recursive procedure similar to the forward algorithm, but this time using a maximization over previous states instead of a summing procedure. The optimal path can be found by keeping track of the argument i that maximized $\delta_t(j)$ in equation 2.15

1. Initialization

$$\delta_1(i) = \pi_i b_i(o_1) \quad 1 \le i \le N \qquad (2.16)$$

$$\psi_1(i) = 0 \qquad (2.17)$$

2. Recursion

$$\delta_t(j) = \max_{1\le i\le N}\left[\delta_{t-1}(i)a_{ij}\right]b_j(o_t) \quad 2 \le t \le T, 1 \le j \le N \qquad (2.18)$$

$$\psi_t(j) = \arg\max_{1\le i\le N}[\delta_{t-1}(i)a_{ij}] \qquad (2.19)$$

3. Termination

$$\tilde{P} = \max_{1\le i\le N}[\delta_T(i)] \qquad (2.20)$$

$$\tilde{q}_T = \arg\max_{1\le i\le N}[\delta_T(i)] \qquad (2.21)$$

4. Path backtracking

$$\tilde{q}_t = \psi_{t+1}(\tilde{q}_{t+1}) \quad t = T-1, T-2, ..., 1 \qquad (2.22)$$

Once again the algorithm can be visualized using a trellis. Figure 2.8 shows the result of applying the Viterbi algorithm to the HMM of figure 2.6 to find the state sequence corresponding to
O = 10110.

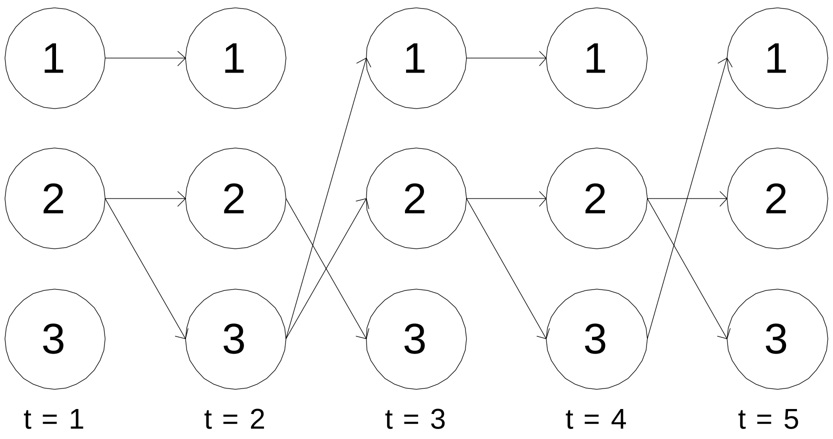


Figure 2.8 - The Viterbi algorithm visualized by a trellis

The calculations below show how the Viterbi path through the trellis is found.

At *t* = 1 the δ variables are initialized:

$$\delta_1(1) = \pi_1 b_1(1) = \frac{1}{2}\times 0.4 = 0.2$$
$$\delta_1(2) = \pi_2 b_2(1) = \frac{1}{2}\times 0.7 = 0.35$$
$$\delta_1(3) = \pi_3 b_3(1) = 0\times 0.8 = 0$$

At *t* = 2:

$$\delta_1(1)a_{11} = 0.2\times\frac{1}{3} = 0.6667$$
$$\delta_1(2)a_{21} = 0.35\times 0 = 0$$
$$\delta_1(3)a_{31} = 0\times\frac{1}{2} = 0$$

Thus $\delta_2(1) = \delta_1(1)a_{11}b_2(0) = 0.02667$

similarily:

$$\delta_2(2) = \delta_1(2)a_{22}b_2(0) = 0.0525$$
$$\delta_2(3) = \delta_1(2)a_{23}b_3(0) = 0.14$$

$$\delta_3(1) = \delta_2(3)a_{31}b_1(1) = 0.0037$$
$$\delta_3(2) = \delta_2(3)a_{32}b_2(1) = 0.01715$$
$$\delta_3(3) = \delta_2(2)a_{23}b_3(1) = 0.0049$$

$$\delta_4(1) = \delta_3(1)a_{31}b_1(1) = 0.0037$$
$$\delta_4(2) = \delta_3(2)a_{22}b_2(1) = 0.0715$$
$$\delta_4(3) = \delta_3(2)a_{23}b_3(1) = 0.0049$$

$$\delta_5(1) = \delta_4(3)a_{31}b_1(0) = 0.00147$$
$$\delta_5(2) = \delta_4(2)a_{22}b_2(0) = 0.0025725$$
$$\delta_5(2) = \delta_4(2)a_{23}b_3(0) = 0.00686$$

At *t* = 5 $\delta_5(3)$ gives the highest probability, the trace back gives state sequence **q** = {2,3,2,2,3} as a solution. It should be clear from the trellis that the Viterbi algorithm takes O(N^2T) calculations. By taking the logarithms of the model parameters, the Viterbi algorithm can be implemented without the need for any multiplications.
As an aside it can be noted that the Viterbi algorithm is not the only possible way to find the optimal state sequence, since there are several possible optimality criteria. For example we could also decide to choose the states q_t that are individually most likely at each time *t*, which would maximize the expected number of correct individual states but may result in an invalid state sequence. Furthermore the algorithm relies heavily on the Markov property of the underlying model. At each time step *t* it assumes that the most likely path into the current state will be part of the most likely path over the entire model through this state.

The next section describes a way of estimating the parameters of a HMM, but the Viterbi algorithm can also be used to find an (initial) estimate of the models state distribution parameters, given an observation sequence $\mathbf{O} = o_1o_2\ldots o_T$ and a particular HMM whose parameters are to be estimated. The procedure that accomplishes this is called Viterbi alignment. It consists of two steps. . In the first step the Viterbi algorithm is used in the normal way, most likely sequence of states $\mathbf{q} = q_1q_2\ldots q_n$ that generated $\mathbf{O}$ is found. In the second step once again the concept of a HMM as generator of speech vectors is used. Each state q_i is thought of as having generated the corresponding subset $\mathbf{O}_i$ from the observation sequence. Now $\mathbf{q}$ effectively segments $\mathbf{O} = \mathbf{O}_1||\mathbf{O}_2||\ldots||\mathbf{O}_n$. These subsets can then be used to calculate the state means and variances.

2.3.5 Baum-Welch algorithm

The third problem of HMMs is to determine a method to obtain or adjust the model parameters $\lambda = (\mathbf{A}, \mathbf{B}, \pi)$. We would like to have a method that has the following maximum likelihood property:

$$\hat{\lambda} = \arg\max_{\lambda} P(\mathbf{O}|\lambda) \tag{2.23}$$

$\hat{\lambda}$ would allow the HMM to account best for the observed data $\mathbf{O}$. What we really want is for $\hat{\lambda}$ is to account for as yet unseen data, so the best we can do here is try to use a training set that is as representative as possible. There is no known way to analytically solve for the model parameter set that maximizes the probability of the observation sequence. But we can choose $\lambda = (\mathbf{A}, \mathbf{B}, \pi)$ such that its likelihood $P(\mathbf{O}|\lambda)$ is locally maximized using a sample of the type of data the model is supposed to generate. This can be done by using an iterative procedure known as the Baum-Welch algorithm or Forward-Backward algorithm. To get to this re-estimation algorithm we first have to define a backward variable $\beta_t(i)$ analogues to the forward variable $\alpha_t(i)$ from section 2.3.3:

$$\beta_t(i) = P(o_{t+1}o_{t+2}...o_T|q_t = i, \lambda) \tag{2.24}$$

That is the conditional probability that $o_{t+1}o_{t+2}\ldots o_T$ are observed and the system starts in state $\mathbf{q}_j$ at time t, given the model λ. $\beta_t(i)$ can be computed inductively:

1. Initialization

$$\beta_T = 1 \quad 1 \le i \le N \tag{2.25}$$

2. induction

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j) \quad 1 \le i \le N \tag{2.26}$$

The initialization step arbitrarily defines $\beta_t(i)$ to be 1 for all i.

Now we can define the probability of being in state i at time t given the entire observation sequence and the model as:

$$\gamma_t(j) = \frac{\alpha_t(i)\beta_t(j)}{P(\mathbf{O}|\lambda)} \tag{2.27}$$

And the probability of being in state *i* at time *t* and state *j* at time *t*+1, given the model and the observation sequence by:

$$\begin{aligned}\xi(i,j) &= \frac{P(q_t = i, q_{t+1} = j, \mathbf{O} | \lambda)}{P(\mathbf{O}|\lambda)} \\ &= \frac{\alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{P(\mathbf{O}|\lambda)}\end{aligned} \tag{2.28}$$

If $\xi_t(j)$ is summed over the time index *t*, the expected number of times that state *i* is visited is obtained or equivalently the number of transitions made from state *i*. Similarly, summation of $\xi_t(i, j)$ over *t* can be interpreted as the expected number of transitions from state *i* to state *j*. Now, using the concept of counting event occurrences, we can estimate a_{ij} as the expected number of transitions from state *i* to state *j* normalized by the expected number of transitions from state *i*:

$$\tilde{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{2.29}$$

And similarly, $b_j(k)$ can be estimated by dividing the expected number of times in state *j* at which symbol o_k was observed by the expected number of times the system is in state *j*:

$$\tilde{b}_j(k) = \frac{\sum_{t=1, o_t = \mathrm{Q}_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \tag{2.30}$$

The initial state distribution for state *j* is simply equal to the expected frequency with which state *j* is visited:

$$\tilde{\pi}_j = \gamma_1(j) \tag{2.31}$$

Since the left hand side of these equations also appears on the right hand side we have to use an iterative procedure to improve the model parameters. After starting with an initial guess for **A**, **B**, and π, λ is used in place of λ in each iteration until the values stop changing within certain limits.

Up to this point the algorithms were explained terms of a finite alphabet of observation symbols and thus discrete probability functions could be used to generate these observations. But as mentioned in section 2.2 in speech recognition continuous signal observation vectors are used. Older system, however, did use a limited codebook of observation symbols. During preprocessing vectors were then mapped to one of the observation symbols using a *k*-means segmentation. In modern state of the art systems the observation vectors are put directly into the Hidden Markov models, therefore multivariate continuous probability density functions are needed to model the distribution of the observations. The form of the density functions then becomes

$$b_j(o_t) = N(o_t, \mu_j, \mathbf{U}_j) \tag{2.32}$$

Where N is an elliptical symmetric density (e.g. Gaussian) with mean vector μ_j and covariance matrix $\mathbf{U}_j$ for state *j*. It can be shown that the re-estimation formulas for these parameters are of the form:

$$\mu_j = \frac{\sum_{t=1}^{T} \gamma_t(j) o_t}{\sum_{t=1}^{T} \gamma_t(j)} \tag{2.33}$$

$$\mathbf{U}_j = \frac{\sum_{t=1}^{T} \gamma_t(j)(o_t - \mu_j)(o_t - \mu_j)'}{\sum_{t=1}^{T} \gamma_t(j)} \tag{2.34}$$

2.3.6 HMM topology for speech recognition

Given a sufficiently large and representative training set, the parameters of a HMM can be estimated as described in the last section, but there is no (known) way to automatically estimate the transition structure of a HMM. Thus the topology of an HMM has to be designed using knowledge of the situation and to some extent, designer's intuition.

The first question that requires an answer here is what kind of unit one HMM should represent. Many choices can be made, for example phrases, words, syllables, phonemes or other sub word units. In fact it is possible to use HMMs to model any unit of speech. Even if the speech units are poorly selected, HMMs have the ability to absorb the suboptimal characteristics within the model parameters; this might of course limit the performance of the system.

Words seem to be the most natural units to model, because they are what we want to recognize and the language model also uses words as basic units. Indeed, recognizers that use word-level models perform rather well. Part of this success is due to the fact that they are able to capture within-word phoneme coarticulation effects. Actually it is shown [1] that because of these effects, the larger the unit, the better the recognition will be. However, as there are many unique words, training data is needed for each of these words, making this kind of system not easily extendible. So for large vocabulary natural speech recognition word units are not really an option. But for small well-defined vocabularies, for example a set of commands, they are well suited. Usually left-to-right model topologies are used in which the number of states depends on the number of phonemes in the word. One state per phoneme is a good rule of thumb.

If sub word units are used, data can be shared among words. This way not all the dictionary entries have to be present in the training phase, the vocabulary of the system is then easily extendible. There are many possible sub word units, typical models include syllable models or linguistically defined sub word units, such as phone models and acoustically defined sub word models, called fenone models.

Linguistically defined sub word models use human specific knowledge for partitioning the parameter space. Acoustically defined sub word units use automatic algorithms to explore the acoustic similarities. Hybrid models, using both acoustic and linguistic knowledge also exist.

Phone models are the most used sub word units. Since there are only 40 to 50 phonemes in languages like Dutch and English, HMMs based on phone models can be adequately trained. Most topologies used in speech recognition are based on the assumption that there are three phases in the pronunciation of a phone. In the first phase the vocal tract is changing shape to pronounce the phone, this is called the on-glide of the phone. In this phase there may be some overlap with the preceding phone. In the second phase the sound of the phone is assumed to be pure and in the third phase the sound is released and the vocal tract starts to transit to the next phone. This is called the off-glide, some overlap with the next phone may occur here, the process is schematically shown in figure 2.9.

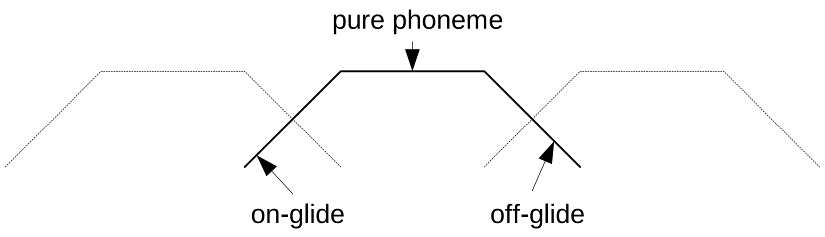


Figure 2.9 – Three phases of a phone

This suggests that at least three states should be used in a phone HMM. Furthermore as a word or even a sentence can be seen as a large string of phones, a left-to-right model structure seems to be necessary, back-loops do not fit very well in this picture.
Adding more states means introducing more parameters and thus more degrees of freedom. Variations in a phoneme can be modeled more accurate but this also introduces a need for more training data to avoid undertraining. And a model should not be too large, a five state model does not work for phones that only occupy three time frames, so in larger models there should always be a 'short-cut' that can handle the shortest example in the training data.

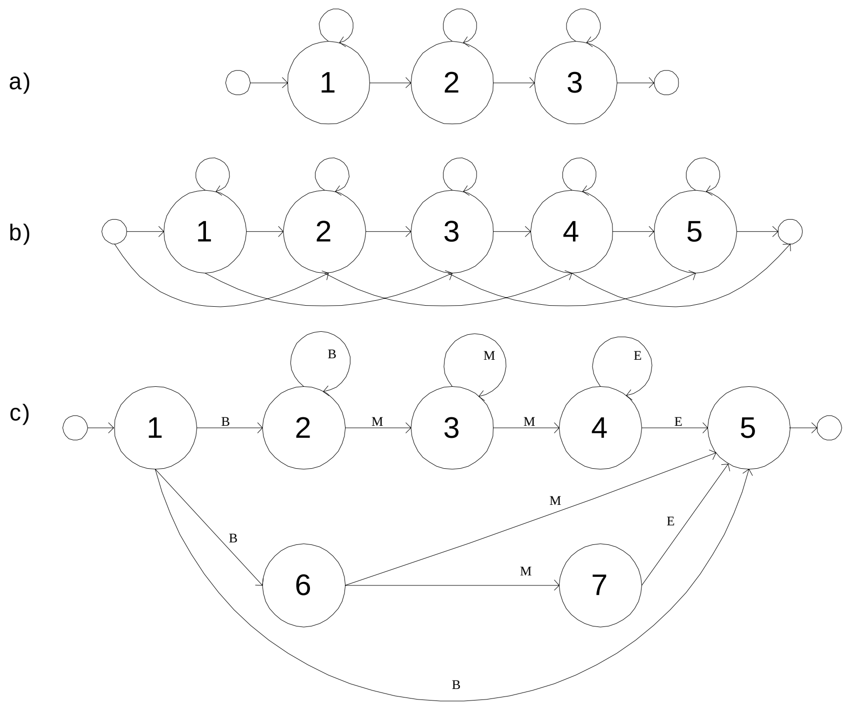


Figure 2.10 - Model topologies for phoneme units

Figure 2.10 shows three model topologies that have successfully been used in various speech recognizers. The first model (2.10a) is a simple three state left-right model with state dependent output probabilities. The first and last smaller circles in the figure represent entry and exit states, these are so called null-states, they do not generate observations and are only used to concatenate the models. The second model (2.10b) has five states, but provides transitions that skip the succeeding state, therefore it is possible to pass through only three states. This model also has state dependent output probabilities. The last model (2.10c) has seven states and twelve transitions with transition dependent output probabilities. Three groups of output probabilities are tied, corresponding with the three phases in a phoneme. In the figure the begin phase is marked with B the middle phase with M and the end phase with E. As a consequence this model only has three different probability distribution functions.

2.3.7 Fitting the pieces together

When we have a language model and a set of word level or phone level HMMs how do these pieces fit together? To answer this question we have to notice that a language model can be seen as a network of states (the words) connected by transitions with probabilities attached to them. In other words a language model can be seen as a Hidden Markov Model. Taking advantage of the fact that embedding HMMs into an HMM leads to a new HMM we can replace the word states by the corresponding word or phone level HMMs resulting in one huge HMM. In case of phone level HMMs, the phone models have to be concatenated to form a word, before substituting.
Now the Viterbi algorithm can be used to find the most likely path through this composite HMM. This path will then lead through a sequence of words that specify the recognized word string. Actually, this method is not guaranteed to find the most likely word sequence equation 2.3 demands that for each candidate word string **W** the probability of the set of paths that correspond to that **W** is found, and then the word whose set of paths has the highest probability is identified. But in practice it is very rare that the word string corresponding to the most probable set of paths is the one actually spoken but the one corresponding to the most probable path is not.

2.3.8 The Beam search

Problem with this approach is that for practical vocabularies even simple grammars result in a large composite model that has just too many states, making the Viterbi algorithm very slow. But it is possible to reduce the state space without compromising the results too much. This is done by a beam search, hypotheses that are below a certain probability level are pruned. At each trellis stage *i* the maximum probability P_{i-1}^{m} of the states at stage *i*-1 is determined $P_{i\text{-}1}^{m} = \max \delta_{i-1}(q)$.
This value serves as a basis for a dynamic threshold:

$$\tau_{i-1} = \frac{P_{i-1}^{m}}{K} \tag{2.35}$$

Where *K* is a suitable chosen constant. Then all states q' on trellis level *i*-1 are eliminated such that $\delta_{i\text{-}1}(q') = 0$ for all states q' that satisfy $\delta_{i\text{-}1}(q') < \tau_{i\text{-}1}$. This purge of improbable paths reduces drastically the number of states entering the comparison implied by the max function in the recursion 2.15 without significantly affecting the values $\delta_i(q)$.

2.3.9 Token passing algorithm

To find the recognized word sequence at the end of the Viterbi search more information is needed beyond the log probability of the path normally calculated by the Viterbi algorithm. An efficient alternative formulation of the algorithm called the token passing model allows for the incorporation of other (meta) information.
In this algorithm each state *i* of the composite HMM at time *t* holds a single movable token which contains, amongst other information, the partial log probability $\delta_t(i)$. This token represents a partial match between the observation sequence $o_1 o_2 \ldots o_t$ and the model subject to the constraint that the model is in state *j* at time *t*. The recursion step of equation 2.18 is then replaced by the equivalent token passing step which is executed at each time frame *t*. The key steps in this algorithm are as follows:

1. Pass a copy of every token in state *i* to all connecting states *j*, incrementing the log probability of the copy by $\log[a_{ij}] + \log[b_j(o_t)]$
2. Examine the token in every state and discard all but the token with the highest probability.

The history of a token route through the network may be recovered efficiently as follows, for each word instance in the composite HMM a word end null-state is added and every token carries a pointer called a word end link. When a token is propagated from the exit state of a word (indicated by passing through a word end node) to the entry state of another, that transition represents a potential word boundary. Hence a record called a Word Link Record is generated in which is stored the identity of the word from which the token has just emerged and the current value of the token's link. A pointer to the newly created WLR then replaces the token's actual link, effectively creating a linked list of word boundaries.

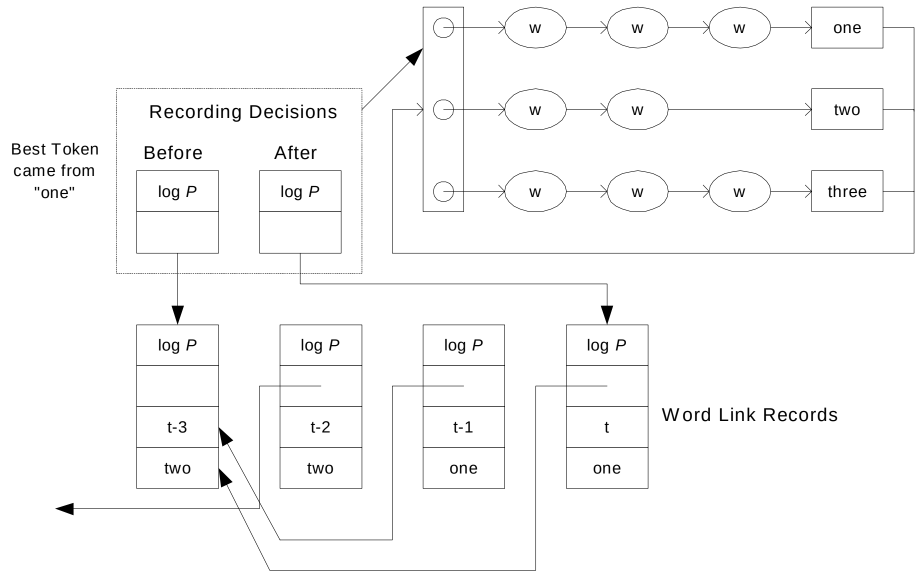


Figure 2.11 token passing using Word Link records

Once all of the unknown speech has been processed the WLRs attached to the link of the best matching token can be traced back to give the best matching sequence of words. At the same time the positions of the word boundaries can also be extracted if required.

2.3.10 N-best search

The Viterbi algorithm finds the most likely single path through the word network. As mentioned before this is not guaranteed to be the optimal solution, and even if it were it still does not have to be the right solution. Therefore it is often desirable to find the *N* most likely word sequences given the observed acoustic data **O**. For instance, to reprocess the data using more refined models or to parse the alternate hypotheses using syntactic or semantic expert system rules.
The required *N*-best search would differ from the Viterbi search in one aspect only: Instead of retaining only one path leading into each trellis state, each trellis state is split into *N* states, one for each of the *N* most likely paths leading into the unsplit state from the split state of the trellis' previous stage.
The *N*-best search can be incorporated in the token passing algorithm by saving more than just the best token at each word boundary. The resulting lattice *N*-best algorithm is sub optimal, because the use of a single token per state limits the number of different token histories that can be maintained. This limitation can be avoided by allowing each model state to hold multiple tokens and regarding tokens as distinct if they come from different preceding words. This gives a class of algorithms called word *N*-best, which has been shown empirically to be comparable in performance to an optimal *N*-best algorithm [9].

2.3.11 Adaptive training

Although the training and recognition techniques described in this chapter can produce high performance recognition systems when trained on a large corpus of speech data, these systems can be improved upon by customizing the HMMs to the characteristics of a particular speaker. By collecting data from a speaker and training a model set on this speaker's data alone, the speaker’s characteristics can be modeled more accurately. Such systems are known as speaker dependent systems, and on a typical word recognition task, may have half the errors of a speaker independent system. The drawback of speaker dependent systems is that a large amount of data (typically hours) must be collected in order to obtain sufficient model accuracy.
Rather than training speaker dependent models, adaptation techniques can be applied. In this case, by using only a small amount of data from a new speaker, a good speaker independent model set can be adapted to better fit the characteristics of this new speaker. This can be done using maximum likelihood linear regression (MLLR), a technique that computes a set of linear transformations by solving a maximization problem using the Expectation-Maximization technique. These transformations will reduce the mismatch between an initial model set and the adaptation data by shifting the means and alter the variances in the initial system so that each state is more likely to generate the adaptation data.
Model adaptation can also be done using a Bayesian approach, in which the speaker independent model parameters and the feature vectors in the adaptation evidence are combined to estimate a new model set. To know how much the model parameters should be changed by the evidence an occupation likelihood for the model state is needed. This can be obtained by running the Viterbi algorithm. Now the next logical step is of course to integrate the Bayesian adaptation approach in the Viterbi algorithm, this way the model set can be adapted on-line during recognition. Each time an utterance is recognized the system is adapted a little more, so the system incrementally gets tailored to the speaker's voice.

3 THE HIDDEN MARKOV TOOLKIT

Subject of this chapter is the Hidden Markov Toolkit, a collection of software libraries and tools for manipulating Hidden Markov models. The architecture of the toolkit will be described and an overview of the tools will be given.

3.1 INTRODUCTION

It may be evident from the last chapter that implementing the algorithms needed to perform the different stages of speech recognition involves quite some work, especially since a lot of optimizations have to be realized to make the algorithms efficient enough for practical use. Fortunately, there are several toolkits available now that implement the algorithms needed in speech recognition, allowing a recognizer designer to focus on the important tasks involved in building a speech recognizer like data preparation, training and recognizers evaluation. One of these toolkits is the Hidden Markov Toolkit (HTK).

HTK is a portable software toolkit for building and manipulating systems that use continuous density Hidden Markov models. It has been developed by the Speech Group at Cambridge University Engineering Department.
HMMs can be used to model any time series and the core of HTK is similarly general purpose. However, HTK is primarily designed for building HMM based speech processing tools, in particular speech recognizers. In can be used to perform a wide range of tasks in this domain including isolated or connected speech recognition using models based on whole word or sub-word units, but it is especially suitable for performing large vocabulary continuous speech recognition.
HTK includes nineteen tools that perform tasks like manipulation of transcriptions, coding data, various styles of HMM training including Baum-Welch re-estimation, Viterbi decoding, results analysis and extensive editing of HMM definitions. HTK tools are designed to run with a traditional command-line style interface, each tool has a large number of required and optional arguments and most tools require one or more script files. Figure 3.1 shows some examples of HTK tools in action.

Example A. a training iteration

```
HERest -T 1 -p 8 -B -C d:\htk\P\config1.cfg -S d:\htk\P\train8.scr
       -I d:\htk\P\plab\triph8.mlf -H ts1\hmms.mmf -M ts2 -d ts1
       -t 250.0 150.0 1000.0 -s stat.txt triph.lst
```

Example B. recognition

```
HVite -T 1 -n 3 3 -s 9 -C ..\wdnexp.cfg -H t17m3\hmms.mmf
      -H t15m3\mono.mmf -l * -i rec.mlf -w d:\htk\P\wdsbi.lat
      -S ..\test.scr d:\htk\P\poly2.dic ctriph4m.lst
```

Figure 3.1 - some HTK commands

Although this style of command-line working results in complex commands, that actually have more resemblance with programming languages than with commands, it has the advantage of making it simple to write shell scripts or programs to control HTK tool execution. Furthermore it allows the details of system construction or experimental procedure to be recorded and documented.

3.2 HTK SOFTWARE ARCHITECTURE

Much of the functionality of HTK is build into library modules, they ensure that every tool interfaces the outside world in exactly the same way and they provide a programming environment for the creation of custom tools or the integration of recognizer functionality in an application. Figure 3.2 shows the library modules and their purpose. User input/output and interaction with the operating system is controlled by the library module *HShell* and all memory management is controlled by *HMem*. General mathematical support is provided by *Hmath* and the signal processing operations needed for LPC and MFCC speech analysis are in *HSigP*.
Each of the file types required by HTK has a dedicated interface module. *HLabel* provides the interface for label files, HLM for language model files, *HNet* for networks and lattices, *HDict* for dictionaries, HVQ for VQ codebooks and *HModel* for HMM definitions.
All speech input and output at the waveform level is via *HWave* and at the parameterized level via *HParm*. As well as providing a consistent interface, *HWave* and *HLabel* support multiple file formats allowing data to be imported from other systems. Direct audio input is supported by *HAudio* and simple interactive graphics is provided by *HGraf*. *HUtil* provides a number of utility routines for manipulating HMMs while *HTrain* and HFB contain support for the various HTK training tools. *HAdapt* provides support for the various HTK adaptation tools. Finally, *HRec* contains the main recognition processing functions.

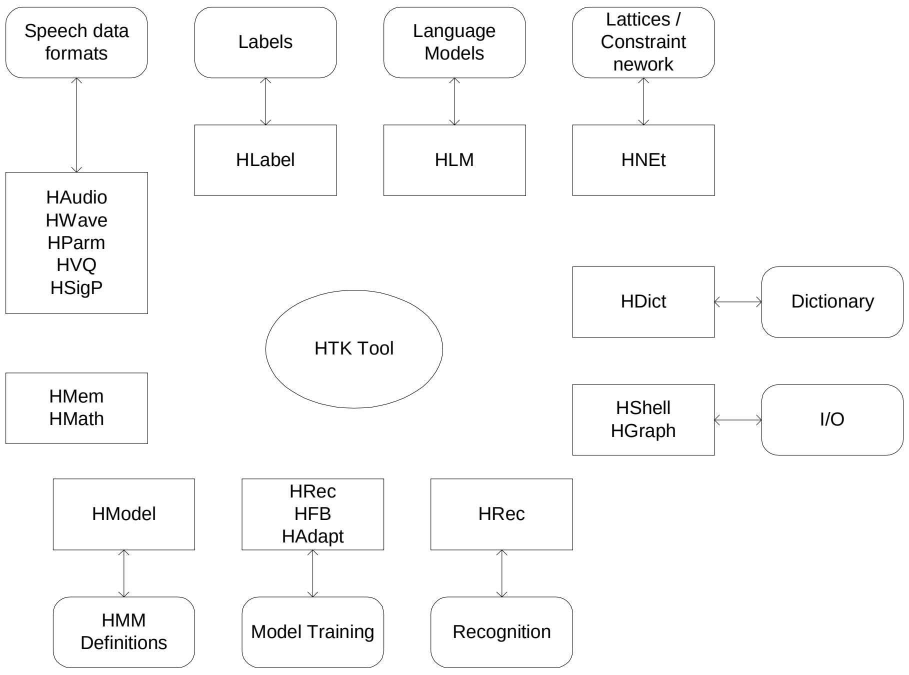


Figure 3.2 HTK libraries

Fine control over the behavior of these library modules is provided by setting configuration variables or by configuration files that contain a complete list of configuration variables.

3.3 OVERVIEW OF THE HTK TOOLS

Figure 3.3 gives an overview of the HTK tools subdivided into groups according to the processing stages in which they are used. These tools all have a similar interface, which is described, in the next subsection. A short description of each tool is given in the subsequent sections.

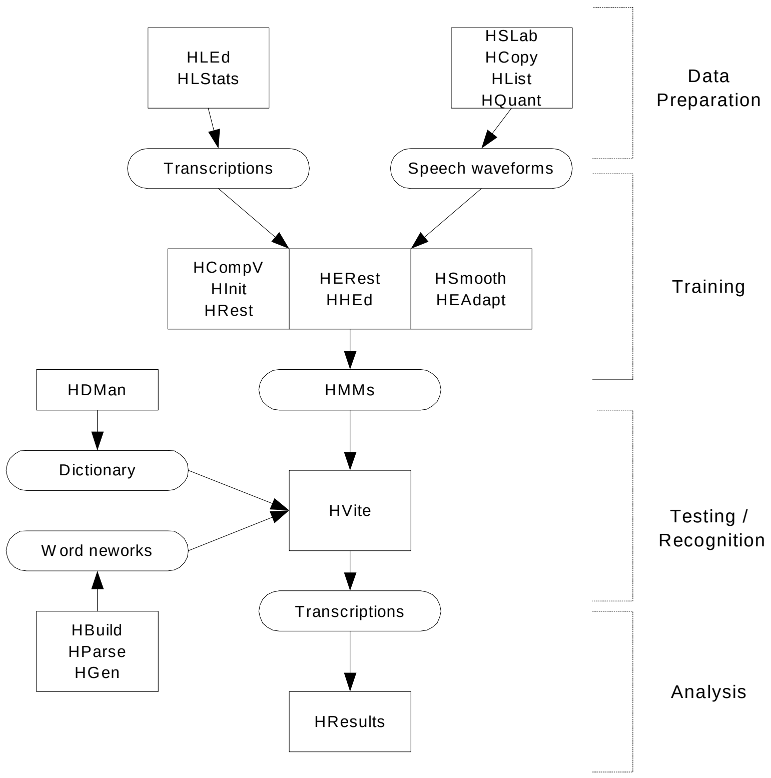


Figure 3.3 The HTK tools

3.3.1 Generic Properties of a HTK Tool

As was said in the introduction HTK tools are designed to run with a traditional command-line style interface. The layout of the commands is the same for all tools. Each tool has a number of required arguments plus optional arguments. The latter are always prefixed by a minus sign. As an example, the following command would invoke the mythical HTK tool called *HFoo*

```
HFoo -T 1 -f 34.3 -a -s myfile file1 file2
```

This tool has two main arguments called *file1* and *file2* plus four optional arguments. Options are always introduced by a single letter option name followed where appropriate by the option value. The option value is always separated from the option name by a space. Thus in the example, the value of the -f option is a real number, the value of the -T option is an integer number and the value of the -s option is a string. The -a option has no following value and it is used as a simple flag to enable or disable some feature of the tool. Options whose names are a capital letter have the same meaning across all tools. For example, the -T option is always used to control the trace output of a HTK tool.

In addition to command line arguments, the operation of a tool can be controlled by parameters stored in a configuration file. Configuration files are indicated by the -C option. For example, if the command

```
HFoo -C config -f 34.3 -a -s myfile file1 file2
```

is executed, the tool *HFoo* will load the parameters stored in the configuration file config during its initialization procedures. Configuration parameters can sometimes be used as an alternative to using command line arguments. Most tools, especially editing tools, like *HHed* which edits Hidden Markov models, or *HLEd* which edits transcription files also need a script that tells the tool which steps to perform and how to edit the data provided to it. The available scripting commands are tool specific.

3.3.2 Data Preparation Tools

HSLab is an interactive label editor for manipulating speech label files. It can be used both to record the speech and to manually annotate it with any required transcriptions. An example of using *HSLab* would be to load a sampled waveform file, determine the boundaries of the speech units of interest and assign labels to them. Alternatively, an existing label file can be loaded and edited by changing current label boundaries, deleting and creating new labels. *HSLab* is the only tool in the HTK package, which provides a graphical user interface.
Although all HTK tools can parameterize waveforms *on-the-fly*, in practice it is usually better to parameterize the data just once. The tool *HCopy* is used for this. As the name suggests, *HCopy* is used to copy one or more source files to an output file. Normally, *HCopy* copies the whole file, but a variety of mechanisms are provided for extracting segments of files and concatenating files. By setting the appropriate configuration variables, all input files can be converted to parametric form as they are read-in. Thus, simply copying each file in this manner performs the required encoding.
The tool *HList* can be used to check the contents of any speech file and since it can also convert input on-the-fly, it can be used to check the results of any conversions before processing large quantities of data.
HLEd is a script-driven label editor which is designed to make transformations to label files, like translating word level label files to phone level label files, merging labels or creating triphone labels. *HLEd* can also output files to a single *Master Label File* MLF, which is usually more convenient for subsequent processing.
HLStats can gather and display statistics on label files and where required, *HQuant* can be used to build a VQ codebook in preparation for building discrete probability HMM system.

3.3.3 Training Tools

HTK allows HMMs to be built with any desired topology. HMM definitions can be stored externally as simple text files and hence it is possible to edit them with any convenient text editor. With the exception of the transition probabilities, all of the HMM parameters given in the prototype definition are ignored. The purpose of the prototype definition is only to specify the overall characteristics and topology of the HMM. The actual parameters will be computed later by the training tools. Sensible values for the transition probabilities must be given but the training process is very insensitive to these. An acceptable and simple strategy for choosing these probabilities is to make all of the transitions out of any state equally likely.
If segmented transcriptions are available the tools *HInit* and *HRest* provide isolated word style training using the fully labeled data as bootstrap data.
HInit can be used to provide initial estimates of whole word models in which case the observation sequences are realizations of the corresponding vocabulary word. Alternatively, *HInit* can be used to generate initial estimates of *seed* HMMs for sub-unit based speech recognition. In this latter case, the observation sequences will consist of segments of continuously spoken

training material. *HInit* will cut these out of the training data automatically by simply giving it a segment label. In both of the above applications, *HInit* normally takes as input a prototype HMM definition, which defines the required HMM topology i.e. it has the form of the required HMM except that means, variances and mixture weights are ignored. The transition matrix of the prototype specifies both the allowed transitions and their initial probabilities. Transitions, which are assigned zero probability, will remain zero and hence denote non-allowed transitions. *HInit* estimates transition probabilities by counting the number of times each state is visited during the alignment process.

HRest performs basic Baum-Welch re-estimation of the parameters of a single HMM using a set of observation sequences. *HRest* can be used for normal isolated word training in which the observation sequences are realizations of the corresponding vocabulary word or it can be used for isolated model training for sub-unit based speech recognition. In this latter case, the observation sequences will consist of segments of continuously spoken training material. *HRest* will cut these out of the training data automatically by simply giving it a segment label. In both of the above applications, *HRest* is intended to operate on HMMs with initial parameter values estimated by *HInit*.

HERest is used to perform a single re-estimation of the parameters of a set of HMMs using an embedded training version of the Baum-Welch algorithm. Training data consists of one or more utterances each of which has a transcription in the form of a standard label file (segment boundaries are ignored). For each training utterance, a composite model is effectively synthesized by concatenating the phoneme models given by the transcription. Each phone model has the same set of accumulators allocated to it as are used in *HRest* but in *HERest* they are updated simultaneously by performing a standard Baum-Welch pass over each training utterance using the composite model.

The tool *HHEd* is a HMM definition editor which will clone models into context-dependent sets, apply a variety of parameter tyings and increment the number of mixture components in specified distributions.

To improve performance for specific speakers the tools *HEAdapt* and *HVite* can be used to adapt HMMs to better model the characteristics of particular speakers using a small amount of training or adaptation data.

3.3.4 Recognition Tools

HTK provides a single recognition tool called *HVite*, which uses a token passing algorithm like the one described in the previous chapter to perform Viterbi-based speech recognition. *HVite* takes as input a network describing the allowable word sequences, a dictionary defining how each word is pronounced and a set of HMMs. It operates by converting the word network to a phone network and then attaching the appropriate HMM definition to each phone instance. Recognition can then be performed on either a list of stored speech files or on direct audio input. *HVite* can support cross-word triphones and it can run with multiple tokens to generate lattices containing multiple hypotheses. It can also be configured to rescore lattices and perform forced alignments.

The word networks needed to drive *HVite* are stored using the HTK standard lattice format. This is a text-based format and hence word networks can be created directly using a text-editor. However, this is rather tedious and hence HTK provides two tools to assist in creating word networks. Firstly, *HBuild* allows sub-networks to be created and used within higher level networks. Hence, although the same low level notation is used, much duplication is avoided. Also, *HBuild* can be used to generate word loops and it can also read in a backed-off bigram language model and modify the word loop transitions to incorporate the bigram probabilities.

As an alternative to specifying a word network directly, a higher level grammar notation can be used. This notation is based on the Extended Backus Naur Form (EBNF) used in compiler specification The tool *HParse* is supplied to convert this notation into the equivalent word network. Whichever method is chosen to generate a word network, it is useful to be able to see examples of the *language* that it defines. The tool *HSGen* is provided to do this. It takes as input a network and then randomly traverses the network outputting word strings. These strings can then be inspected to ensure that they correspond to what is required. *HSGen* can also compute the empirical perplexity of the task.
Finally, the construction of large dictionaries can involve merging several sources and performing a variety of transformations on each sources. The dictionary management tool *HDMan* is supplied to assist with this process.

3.3.5 Analysis Tool

A tool called *HResults* compares recognition results with original transcriptions. It uses dynamic programming to align the two transcriptions and then counts substitution, deletion and insertion errors. Options are provided to ensure that the algorithms and output formats used by *HResults* are compatible with those used by the US National Institute of Standards and Technology (NIST). As well as global performance measures, *HResults* can also provide speaker-by-speaker breakdowns, confusion matrices and time-aligned transcriptions. For word spotting applications, it can also compute *Figure of Merit* (FOM) scores and *Receiver Operating Curve* (ROC) information.

4 DEVELOPMENT OF A SPEECH RECOGNIZER

This chapter describes the design and construction of a speech recognizer for the Dutch language. First the requirements for the system are stated and the development platform and tools are described. Next an outline of the steps needed to build a speech recognizer is given. Subsequent sections explain how these steps were performed. For each step the actions that were actually taken are explained to ensure that this description can be used as a guide for similar projects. Whenever appropriate some background theory is also explained, the emphasis here is on the differences between practice and the theory described in chapter two. The final sections give an evaluation of the system.

4.1 INTRODUCTION

This chapter describes the design and construction of a speech recognizer for the Dutch language that is capable of recognizing sounds recorded by a normal desktop computer microphone. There are many possible types of speech recognizers, ranging from task-specific word recognizers or word spotting systems to large vocabulary recognizers for natural free speech. The system that was build here was intended to be as general as possible, so that it can be used, with slight modifications and some adaptive training, as a speech interface for many different, more specific tasks and applications. An example of such an application is the telebanking system introduced in 2.2.3 that can be used to conduct financial transactions by telephone. Moreover the system should also provide a baseline system for further research on the subject of speech recognition. An example for this category, and the immediate cause for building this system, is the integration of multiple modalities in a Hidden Markov based speech recognizer as described in part II.
To meet these requirements the recognizer is designed to recognize large vocabulary continuous speech and is speaker independent. To make sure that the systems vocabulary was not limited in any way to some fixed set of words it was decided to create a phoneme based recognizer. This way the system is easily extendible. A different vocabulary only involves a change of language model and possibly adding some words to the pronunciation dictionary.
The final system uses context dependent models, but as this system was build and refined incrementally, actually a whole set of recognizers has been created, ranging from a simple monophone recognizer to a sophisticated multiple mixture triphone system.

4.2 THE DEVELOPMENT ENVIRONMENT

The system was developed on the Microsoft Windows platform using a variety of tools, programs and scripts. The first set of scripts that was written can be described as meta scripts, because they contain the calls to the different tools and provide these tools with their numerous attributes, scripts and with the output obtained from other tools. This approach was taken to make reproduction of the entire process easy. This is not just useful for future attempts to build similar systems, but it also turned out to be a necessity for keeping the process manageable and for recovering from errors.
By far the largest set of tools and software libraries was taken from the HTK toolkit described in the last chapter. The choice of using HTK, rather than implementing the speech recognition algorithms in some general purpose programming language was based on two facts.

1. Although implementing the basic algorithms, necessary for performing speech recognition, like the Viterbi algorithm, is straightforward, given the formulas in chapter two, it is much harder, and thus more time consuming, to create implementations efficient enough to be

useful in practice. As will be seen later on in this chapter training a system using the highly optimized HTK tools still takes vast amounts of time.

2. HTK offers a nice and flexible data structure for defining and manipulating Hidden Markov models. The main advantage of this structure is that it uses a text based definition language, which allows for manual manipulation of the models or manipulation by custom made tools. A property that proved very valuable, not only during the construction of the speech recognizer, but also in later experiments that were concerned with the integration of models from different modalities in the speech recognizer.

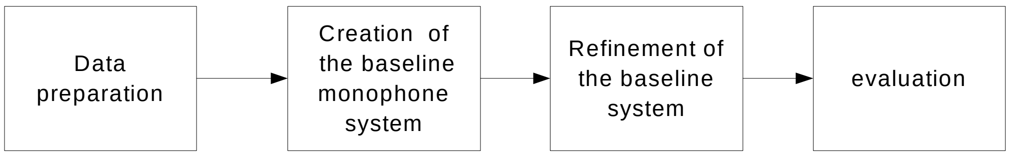
As was described in the last chapter most HTK tools are script-driven. More than 100 of these scripts were created to control the development process. A number of these scripts were created by programs written in C++ that filled the gaps in the development process not covered by one of the tools. This includes a program for data selection, some tools for creating a initial acoustic model set and a tool for creating scripts for triphone clustering.
Further explains of the tools will be given in the next sections, when appropriate.

4.3 OUTLINE OF THE DEVELOPMENT PROCESS

As was described in Chapter two a Hidden Markov model based speech recognizer consists of three parts.

1. A preprocessing part, which extracts relevant features from the speech signal.
2. A language model that tells how likely a certain word string is to occur.
3. An acoustic model, which computes how a word string is likely to be pronounced.

This subdivision gives some clues on the decisions that have to be made and the steps that have to be performed to build a recognizer. In the case of the preprocessor decisions have to be made concerning the data format of the audio streams and on the type of preprocessor that is to be used. To build a language model a vocabulary has to be chosen and it should be decided what kind of grammatical structure of the language model should impose on the utterances.
With regard to the acoustic model decisions have to be made concerning the unit of speech on model represents, the number of states in such a model, the way these states are connected and concerning the type of distribution functions. These acoustic models should then be trained and refined using a set of examples. So a training data set has to be prepared.


Data preparation
Creation of the baseline monophone system
Refinement of the baseline system
evaluation
Figure 4.1 the development process

As shown in figure 4.1 the overall development process can be divided in four stages. In the data preparation stage training data is collected and prepared and the design decisions concerning the models are taken. In the initial training stage, a simple, baseline speech recognition system is build. In the refinement stage this systems is then incrementally refined to make it more advanced and robust. In the final stage the performance of the system is evaluated. The next four sections describe how these four stages were performed to build the speech recognizer.

4.3 Data preparation

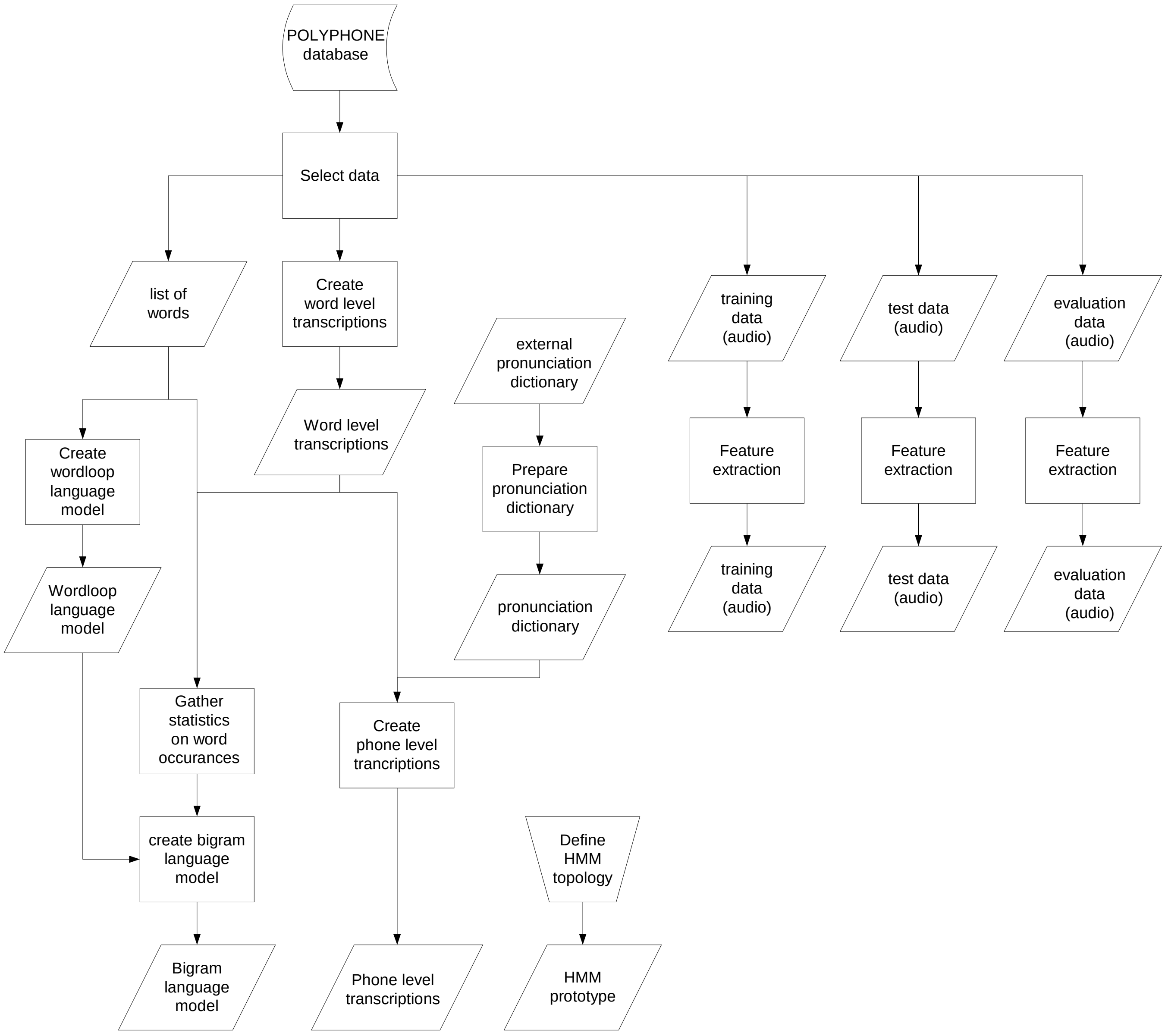


Figure 4.2 - Steps performed during data preparation

The data preparation phase involves a number of inter-related tasks a shown in figure 4.2. Obviously speech data is needed for training, testing and evaluating the models in the process, therefore the first step involved selection of appropriate speech recordings. For each of these recordings a textual transcription was created. Further a language model was created and a HMM topology was defined. The exact steps and their relations are now described in detail.

4.3.1 Data selection

In order to build a robust large vocabulary speech recognizer it is crucial that all acoustic (sub-word) models receive enough training examples, taking into account the many variations that can occur in speech, due to, for example, variation in speed of speech, different gender and different accents. It follows that a training corpus should be sufficiently large, consisting of speech samples

spoken by men and women from different dialect regions. Since the sound of a sub-word unit is also influenced by its context the speech samples should include all phonemes in as many different phonetic contexts as possible. In practice this means that, depending on the quality of the samples and the complexity of the acoustic models, about ten- to thirty-thousand (phonetically rich) sentences are necessary.
Recording such a data set is a major undertaking involving sub-tasks like selection of participants, recording the data and the most time-consuming parts post-processing and transcribing the data. Fortunately, many of these speech corpora are now commercially available. One of the corpora available for the Dutch language is the Polyphone database, which is used to select the training data for the recognizer as it contains speech samples that fulfill the requirements stated above. It is rather large; it contains telephone speech from 5050 different speakers and 222075 speech files, based on 44 or in some cases 43 items per speaker. The speakers in the database were selected from all geographic regions. The ratio between male and female speakers is almost fifty-fifty. The utterances contain all Dutch phonemes in as many phonetic contexts as the designers of the database could find. But the set also has some disadvantages that all stem from the fact that the Polyphone database was recorded with automatic voice-interactive telephone services in mind. Most speech files therefore contain examples of phrases useful for this kind of applications, this includes street names, bank-accounts, numbers and answers to yes-no questions. Training a large vocabulary recognizer on these samples may result in a recognizer that performs well on recognizing numbers and 'yes' and 'no' but which generalizes very poor to other words. To avoid these problems only nine items per person were used, five phonetically rich sentences and four application sentences. The latter group consists of sentences that contain an application word, that is, a word that is often used in normal speech. Since the polyphone database is recorded over a telephone line many samples are of poor quality, or contain background noise or even background speech. To ensure well-trained models that can be used for recognizing speech using for example a PC microphone spoken by an average person, only sessions that fit the following profile were used:

- Men and women from all regions.
- Only native speakers that did not live abroad (this may be a bit rigorous, as not all foreigners have an accent, and a small accent shouldn't be a problem, but it avoids manually going through thousands of utterances).
- No utterances that contain background noise or background speech. But mouth noises, like smacking, sniffing or loud breaths and verbal hesitations like uh are allowed.
- Only sentences that are assessed to have quality level OK in the polyphone database, which means that the text can be clearly understood, i.e. no stuttering, no disturbing hesitations and no mispronunciations or foreign pronunciations.

Manually selecting utterances that adhered to this profile is practically impossible. The data in the polyphone database is stored on 10 CD-roms each with about 500 directories, called sessions, with one speaker per session. Information concerning the speakers and the utterances is listed in three large plain text format tables. No data selection tools whatsoever are included in the database.
To overcome this problem a program was written, called *Poly2Htk*, which automatically selects a subset of the polyphone database according to some selection criteria it is provided with.
This program provides a command line interface, in a similar fashion as the HTK tools. At this command line a number of options can be specified:

- A subset of the database that should be searched.
- A caller profile: gender, age interval, dialect region, education level, foreign / lived abroad.
- A subset of the utterances in each session.

- A quality profile: Cordless phones allowed, allowed types of noise, allowed levels of quality.
- A number of output options.

The program creates based on the selected options a number of scripts needed by some HTK tools in later steps, which will be described in relevant sections. Furthermore it creates a list of all the words in the selected set and a either file containing all the transcriptions of the selected utterances index by the name of the utterances or one file per utterance with the same name as the utterance but a different file type. The program also creates unique filenames for the selected utterances.

Three different data sets were extracted from the Polyphone database this way. A training set, a development set, which was used for testing and fine tuning during development of the system and an evaluation test set to evaluate the final performance of the system. The development set contained persons and sentences that did not occur in the training set. And the evaluation test set contained persons that did neither occur in the training or development set, but its phonetically rich sentences did also occur in the training set.

```
TRAIN - PolyPhone to HTK Summary:

4022 sessions processed
  2883 sessions OK
  1139 sessions skipped

25954 utterances processed
  22626 utterances OK
  1632 utterances skipped because of bad transcriptions
  1689 utterances skipped because of bad sound quality

  7 utterances missing

DEVELOP - PolyPhone to HTK Summary:

500 sessions processed
  340 sessions OK
  160 sessions skipped

3060 utterances processed
  2673 utterances OK
  200 utterances skipped because of bad transcriptions
  187 utterances skipped because of bad sound quality

EVAL - PolyPhone to HTK Summary:

528 sessions processed
  368 sessions OK
  160 sessions skipped

3313 utterances processed
  2885 utterances OK
  198 utterances skipped because of bad transcriptions
  229 utterances skipped because of bad sound quality

  1 utterances missing
```

Figure 4.3 results Polyphone 2 HTK

4.3.2 Feature vectors

The audio files on the Polyphone CDs are saved as waveforms. In particular, they are stored in compressed CITT A-law format, the format used in telephone systems. The first step in speech recognition is the extraction of feature vectors. Theoretically this can be done on the fly during training, as is the case during real-time recognition. But for training purposes it makes sense to do this once and for all and save the feature vectors in a file, as each utterance is processed a number of times during training.
Preprocessing of the data files was done in two stages. First, the original waveform files were translated to intermediate audio files in the SPHERE format. This was done by the *Praat* program, an audio analysis program developed at the University of Amsterdam, which provides extensive scripting possibilities. The scripts that guided this program were created by *Poly2Htk* during data selection. This first step was necessary, because the program used in the second step, the HTK tool *HCopy*, cannot read the A-law audio format. *HCopy* is a tool for translating audio files to feature vector files using one of the many variants of either linear predictive coding or Mel scale cepstral coefficients. It takes as its parameters a list summing all the files it has to copy and the new filenames they should be copied to and a script containing information on input and output formats. The filename list was provided also by *Poly2Htk*.
The utterances were encoded to Mel-frequency cepstral coeffient vectors. This choice was based on literature and earlier experiments. Each vector contained twelve cepstral coefficients with log energy and delta and acceleration coefficients added, all scaled around zero by subtracting the cepstral mean from all vectors. Which resulted in 39 dimensional feature vectors. A sampling rate of 10 ms was used and a overlapping window of 25 ms.

4.3.3 Phoneme set

As was already mentioned in the introduction, the basic speech units in the system are phonemes. This implies that a phoneme set should be chosen. Unlike the situation for example with the characters in the alphabet, there is no such thing as the phoneme set for a given language. Existing sets differ in the number of details they provide. For example the so-called plosive sounds, like */p/* could be modeled as one sound or as two sounds. Namely a closure */cp/*, that is a short period of silence because the mouth is closed before a plosive is uttered and the */p/* sound that is produced as soon as the mouth is opened.
However, there are a few standard phoneme sets, used for example in dictionaries. One of those is the SAMPA notation, which covers all distinguishable phoneme sounds in the Dutch language. It provides 43 different phonemes, which is a fairly reasonable number for the use of a model set for speech recognition.
The phonemes from the SAMPA set were adopted as phoneme set in this project, but they were renamed for convenience in later steps. Table 4.1 gives an overview of the phonemes including an example for each phoneme to show which sound it represents.
Furthermore three other phonemes were added. The first, *sil*, models (longer periods) of silence, these silences occur between sentences or when a person is not speaking at all. The second phoneme, *sp*, also represents silence, but only periods of short duration. This is the kind of silence that occurs between words. The reason that a distinction was made between these two types of silence is that they really represent two different phenomena. One represents periods of pure silence (which in this context always means background and environment noise) that can greatly vary in length while the other represents optional periods of silence shorter than the

duration of a phoneme. Because these silences mainly occur between words there is a slight influence from the neighboring sounds on this phoneme.
The reader may have noticed that utterances containing mouth noise were not excluded during data selection. This was done on purpose, to enable the creation of the final phone that was added, *mn*, which models all kinds of mouth noise and verbal hesitation. The idea behind the inclusion of this phone was that real, natural speech always contains mouth noise and modeling this may improve the results in real-life environments.

Table 4.1 Phone set

SAMPA notation	used notation	example	phonetic transcription
I	i	pit	p i t
E	e	pet	p e t
A	a	pat	p a t
O	o	pot	p o t
Y	y	put	p y t
@	at	gemakkelijk	g at m a k at l at k
i	ie	vier	v ie r
y	yy	vuur	v yy r
u	u	voer	v u r
a:	aa	naam	n aa m
e:	ee	veer	v ee r
2:	eu	deur	d eu r
o:	oo	voor	v oo r
Ei	ei	fijn	f ei n
9y	ui	huis	h ui s
Au	ou	goud	x ou t
E:	eh	crème	k r eh m
9:	euh	freule	f r euh l at
O:	oh	roze	r oh z at
p	p	pak	p a k
b	b	bak	b a k
t	t	tak	t a k
d	d	dak	d a k
k	k	kap	k a p
g	gg	goal	gg oo l
f	f	fel	f e l
v	v	vel	v e l
s	s	sein	s ei n
z	z	zijn	z ei n
x	x	toch	t o x
G	g	goed	g u t (also: x u t)
h	h	hand	h a n t
Z	zj	bagage	b a g aa z at
S	sh	show	sh oo u
m	m	met	m e t
n	n	net	n e t
N	nn	bang	b a nn
l	l	land	l a n t
r	r	rand	r a n t
w	w	wit	w i t
j	j	ja	j aa

4.3.4 Transcriptions and pronunciation dictionary

To monitor the progress that is made during training and to evaluate the performance of the final system textual transcriptions of what was really said in the test and evaluation utterances were needed. As was already mentioned these were created by *Poly2Htk*. But there is more to it, the acoustic models represent phones, so in the training phase examples of phones will be needed to adjust the parameters of these models. But the examples in the set of training files constitute complete sentences, that is, a string of phones. To ensure that the right part of an utterance is used to train a model a transcription of the utterance at the phone level is needed. These phone level transcriptions can be created from the word level transcriptions by using a pronunciation dictionary that gives for each word the sequence of phones that make up the pronunciation of this word. This is exactly the way it was done during this project. First of all a pronunciation dictionary had to be prepared. With the polyphone database comes a dictionary that includes almost all words used in the polyphone utterances. This dictionary was used as a starting point for the pronunciation dictionary. Some changes had to be made to the format of the dictionary because the HTK transcription editor *HLEd* needs a particular form of dictionary.
First all characters were transformed to lowercase and punctuation symbols (\" and \') were changed and in some cases removed because HTK has difficulty with transcriptions containing certain non-alphanumeric characters. This resulted in some strange words like ‘patient’, ‘reeel’, ‘savonds’ but this is not really a problem, as there is no good reason why the spelling internal to the system should be the proper spelling (as it is output to a human reader). Next stress-markers and syllable information were removed from the dictionary and the phonetic transcriptions were transformed to the right format, using the new phone names.
Having a phone set, a dictionary and word level transcription files the phone level transcriptions could be prepared. It turned out that the word level transcriptions contained words that did not occur in the dictionary. Words, for which the transcription was clear, because they were similar to word already in the dictionary, were added to the dictionary, otherwise the transcriptions and corresponding utterances were removed from the data set (in total 7 items were removed).
Using the word level label files the phone set and the dictionary, the phone level label files were created, using the *HLed* tool. This tool takes a list of all label files that are to be processed, a dictionary and a script containing the commands that should be executed as its arguments. It this particular case it substituted each word in the label files with the first phonetic transcription it found for this word in the dictionary. The phone level transcriptions obtained this way were unsegmented, that is they did not contain any information on the time intervals at which a phoneme is spoken. This means that there is no way to know exactly which part of an utterance represents a certain phone and should be used to train this phones HMM. This makes model training a little harder, than when segmented data would be available, but not impossible as will be described in the sections on training.

4.3.5 Language models

The language model is a critical part of a speech recognition system. It tells the system how likely it is that a certain string of words is uttered. By doing so, it imposes a grammar onto the system. Furthermore, the language model is the glue that holds together the set of acoustic models in the word network that is ultimately used for recognition. In this project a number of language models at various levels of sophistication was created for each of the data sets.
First simple wordlist grammars were generated, using the wordlists provided by *Poly2Htk*. These word networks are not really grammars at all, since every word is equally likely and a sentence is

just a string of these words. In order to put some grammatical constraints on the language model, the grammars were transformed to backed-off bigram language models. Using the word level transcriptions and statistics on the number of occurrences of each word and of each word combination were calculated according to the following formulas:

$$p(i,j) = \begin{cases} (N(i,j) - D)/N(i) & \text{if } N(i,j) > t \\ b(i)p(j) & \text{otherwise} \end{cases} \tag{4.1}$$

Where *N(i,j)* is the number of times word *j* follows word *i* and *N(i)* is the number of times that word *i* appears. *D* is the so-called discount constant, which is used to deduct a small part of the available probability mass from the higher bigram counts to and distribute it amongst the infrequent bigrams.
When a bigram count falls below the threshold *t*, the bigram is backed-off to the unigram probability suitably scaled by a backed-off weight *b(i)* in order to ensure that all bigram probabilities for a given history sum to one. The unigram probability is computed using:

$$p(i) = \begin{cases} N(i)/N & \text{if } N(i) > u \\ u/N & \text{otherwise} \end{cases} \tag{4.2}$$

Where *u* is a constant called unigram floor count and *N* is the total number of words:

$$N = \sum_{i=1}^{L} \max(N(i), u) \tag{4.3}$$

.

The backed-off weight follows from:

$$b(i) = \frac{1 - \sum_{j \in B} p(i,j)}{1 - \sum_{j \in B} p(i)} \tag{4.4}$$

In which *B* is the set of all words for which *p(i,j)* has a bigram.

These statistics were then used to create backed-off bigram language models for the training, test and evaluation sets, using the *HDMan* tools which translated the gathered statistics into HTK Standard Lattice Format, that are used for storing word models and multiple hypotheses from the output of a speech recognizer. For testing purposes a simple phoneme language model was also created.

4.3.6 HMM prototype

The last step in data preparation was the selection of a Hidden Markov Model topology for the acoustic models. Since a phoneme based recognizer was build a model represents a phoneme. A topology consisting start and end states states and three emitting states, using single Gaussian density functions, was chosen. The states were connected in a left-to-right way, with no skip transitions. The model is shown in figure 4.4

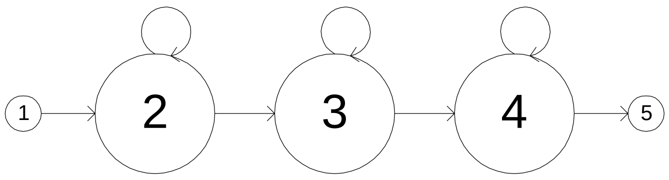


Figure 4.4 - The acoustic HMM topology

This is a rather simple model topology, but earlier experiments with three and five state models showed that the three state model performs as well, and in some cases even slightly better than the five state model, despite the fact that is has fewer parameters. These experiments showed that it works best to start with a simple model with only a few parameters and then increasing the complexity of the model as training progresses. So this is the approach that was also taken during this project.
Another advantage of the simple model topology is, that it is easier to integrate with lipreading models, that are, due to the lower video frame rate, usually small. As described in part II such integration experiments were planned with the speech recognizer build here, that's why this argument did count at this stage.

4.4 TRAINING

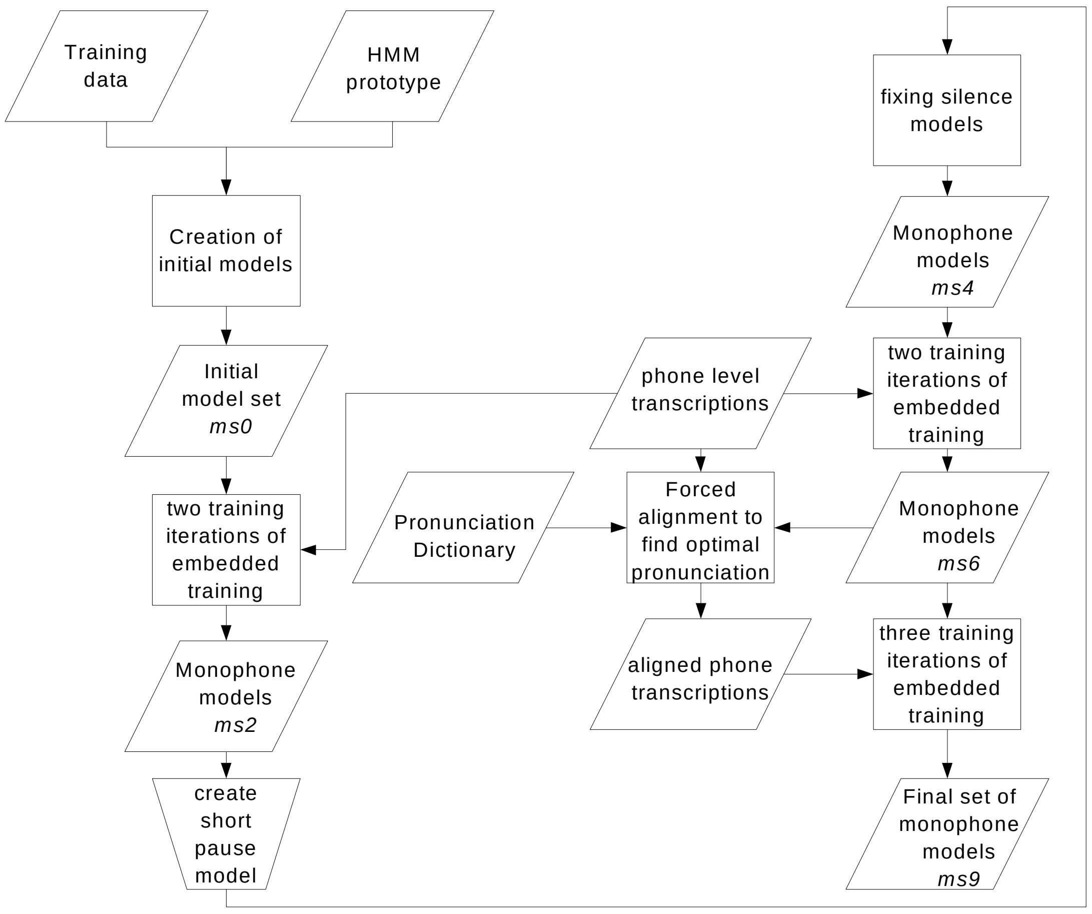

Training data
HMM prototype
Creation of initial models
Initial model set *ms0*
two training iterations of embedded training
Monophone models *ms2*
create short pause model
Pronunciation Dictionary
phone level transcriptions
Forced alignment to find optimal pronunciation
aligned phone transcriptions
fixing silence models
Monophone models *ms4*
two training iterations of embedded training
Monophone models *ms6*
three training iterations of embedded training
Final set of monophone models *ms9*

Figure 4.5 - Steps performed during training

With all data sources in place the actual training of the acoustic Hidden Markov Models could start. Figure 4.5 shows the steps that were taken to train a monophone speech recogizer. First, a set of acoustic models was created and trained. Next the system was made more robust by adding sophisticated silence models. Thereupon the system was trained a little more and Viterbi alignment was used to find the most likely transcriptions. These transcriptions were then used to train the monophone system. These steps will now be described in detail.

4.4.1 Initial models

As explained in chapter 2.5 training of HMMs can be done using the Baum-Welch algorithm, but to ensure proper and fast convergence of the models sensible initial values have to be calculated for the transitions parameters and the means and variances of the state density functions before the Baum-Welch algorithm can be used. HMMs are pretty sensitive to initial values so this stage is crucial for the entire process.
Good initial models can be obtained by using the concept of a HMM as a generator of speech vectors. The training examples of the phones corresponding to the model whose parameters are to be estimated can be viewed as the output of this model. Thus if the state that generated each vector in the training data was know, then the unknown means and variances could be estimated by averaging all the vectors associated with each state.
This principle can be implemented by using the Viterbi algorithm in an iterative scheme. In every iteration the Viterbi algorithm is used to find the most likely state sequence corresponding to each training example, then the HMM parameters are re-estimated. This process is repeated until no further increase in the log likelihood calculated by the Viterbi algorithm is obtained.
Drawback of this algorithm is that it requires segmented transcription data to collect all examples corresponding to a certain model. As mentioned before, in the case of the Polyphone data segmented labels were not available, therefore an alternative initialization technique was used, called the flat start scheme. Which comes down to an uniform segmentation of the data, by computing the global mean and variance for each feature and setting all the Gaussians of all the models to have this mean and variance. For this purpose a tool called *HCompVSet* was written. This tool uses code from the HTK tool HcompV to compute the global mean and variance over the entire data set. Subsequently it creates definitions for each model in the set.
Apart from the model set a variance floor vector was created which was equal to 0.01 times the global variance. This vector was used to set a floor on the variances estimated in subsequent steps. If a variance might fall below this floor it will be set equal to the floor variance.

4.4.2 Embedded re-estimation

Once the initial model set is available the re-estimation of the model parameters can start. Standard procedure would be to process each model in turn by selecting the training examples corresponding to the model and using the Baum-Welch re-estimation algorithm to update the models. The HTK tool *HREst* takes this approach. But this algorithm also requires segmented labels to locate the examples corresponding to a model and thus could not be used in the case of the polyphone recognizer. And actually, it was not even desirable to use the standard algorithm in this case, since the recognizer was supposed to recognize continuous speech and not isolated words or phonemes. Therefore an embedded re-estimation strategy was used, that simultaneously updated all of the HMMs in the system using all of the training data. In the embedded training algorithm each training utterance is processed in turn, the associated transcription is used to construct a composite HMM which spans the whole utterance. This composite HMM is made by concatenating instances of the phone HMMs corresponding to each label in the transcription.

The Forward-Backward algorithm is then applied and the sums needed to form the weighted averages are accumulated. When all of the training files have been processed, the new parameter estimates are formed from the weighted sums and the updated HMM set is created.
This embedded procedure is implemented in the HTK tool *HERest* which performs exactly one iteration of the algorithm each time it is ran, in contrast to the standard procedure which iterates until convergence is reached. In the case of large data sets a single iteration of embedded training may take several hours to compute. Therefore, *HERest* provides many optimizations to speed up the training process. It is capable of pruning the **A** and **B** matrices using a beam search-like approach. By this means, a factor of 3 to 5 speed improvement and a similar reduction in memory requirements can be achieved. Furthermore *HERest* includes features to allow parallel operation where a network of processors is available. The training set can be split into separate chunks that are processed in parallel on multiple processors.
For the initial models created in the last step the re-estimation process was performed twice. Pruning was not enabled during any of the re-estimation cycles in the entire process to ensure that all models were trained as optimal as possible given the data. But, although only one processor was used, the data set was split in eight subsets of about 2500 utterances each. This way the available processor time could better be utilized, as smaller continuous time spans were needed. Inspection of the source code of *HERest* revealed that it loaded its, rather large, scripts completely into memory. By splitting up the data sets the scripts were also split up in a number of smaller scripts. This reduced the memory usage of *HERest* and the time taken to load data in to memory, making it considerably faster.

4.4.3 Fixing the silence models

After the models received a little training and roughly started to take shape the system was adapted to make it more robust. One of the models created by *HCompVSet*, is the silence model *sil*. This model thus has the same topology as the other phones. But it is supposed to take care of periods of silence that can vary greatly in length, from a few milliseconds up to a few seconds, that is. Therefore a transition was added from the second state to the fourth and back from the fourth state back to the second. This had to be done after the system received some training during re-estimation, because otherwise the *sil* model might have absorbed a large part of the utterance.

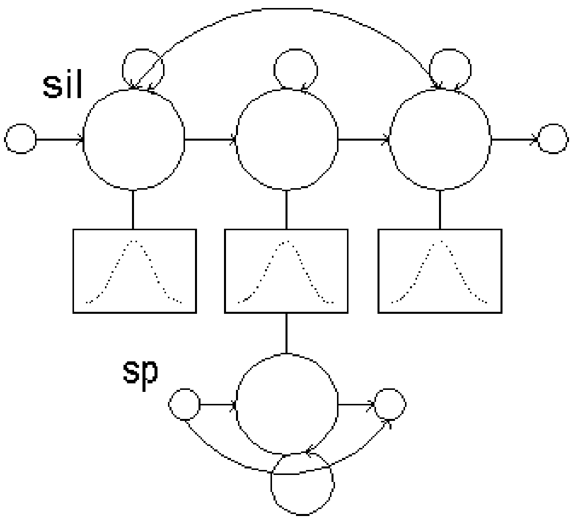


Figure 4.6 - the sp model is tied to the middle state of the sil model

Training sofar was done without the short pause model *sp*. This was done because of the special nature of the model. This model is meant to take care of optional silences between words, to make sure that these periods of silence are not assigned to other models during training, which would compromise these models. But if it were included right from the start in the training process exactly the opposite might happen. In some cases there may be no or not enough silence between words, then the *sp* model would pick up some vectors belonging to its neighboring phones. Since the *sp* model has a very short duration and practically all combinations of phones can occur as its neighbors. This may result in a very poor model that might suppress the performance of the whole system, as it is one of the most frequently occurring models. So in this particular case it is beneficial to explicitly define what type of data this model is supposed to represent. A nice way to do this, is to tell the system that the *sp* model resembles the silence model *sil*. This was realized by creating a one state *sp* model, which has a direct transition from entry to exit node, a so called *tee*-model. Its one and only state was then tied to the middle state of the silence model, so these two states now shared the same set of parameters, as shown in figure 4.6. After fixing the silence models two re-estimation cycles were performed.

4.4.4 Alignment

Many words, particularly function words, have more than one pronunciation. In the construction of the phone level label files in the data preparation phase the first pronunciation encountered in the dictionary for each word was chosen. This was not necessarily the right pronunciation. To fix this problem the models created so far were used to create new transcriptions using pronunciations that fitted the acoustic data embodied in the models. This was done by performing Viterbi alignment.
For each utterance a network including all alternative pronunciations in parallel was constructed, using the corresponding word level label file and the dictionary. The Viterbi decoder, implemented in the tool *HVite* then searched and segmented the best matching path through the network and constructed a lattice which included model alignment information. This lattice/path was then converted to a new segmented phone level transcription. Subsequently another two passes of the re-estimation procedure were performed.

4.4.5 Monophone system

Another final re-estimation cycle resulted in a set of models sufficiently trained to pass for a simple speech recognizer. To check the performance of this system the Viterbi decoding algorithm was used, implemented by the tool *HVite*, which uses a token passing approach similar to the one described in chapter two, to perform recognition on the development test set.
The transcriptions output by the Viterbi algorithm were compared to the original word level transcription files using the *HResult* analysis tool, which uses a dynamic programming-based string alignment procedure that is fully interchangeable with the one used in the standard US NIST scoring package. The analysis tool computes the percentage of words correctly recognized as

$$Correct = \frac{H}{N} \times 100\% \qquad (4.5)$$

In which H is the number of labels recognized correct and N is the total number of labels.
The word accuracy, which takes into account the fact that some of the words classified as correct may be in fact insertion errors, is computed by

$$Accuracy = \frac{H - I}{N} \times 100\% \quad (4.6)$$

where *I* is the number of insertion errors.
In this test and in all other tests conducted during development described below, the first half of the development test set was used. This comprises 240 utterances spoken by about 100 different persons. The subset contained 1060 different words, a bigram language model containing these words was used in these tests.

Table 4.2 - recognition results monophone system

System	Percentage of words recognized	Word accuracy percentage
Initial model set after re-estimation (*Ms2*)	17.15%	-77.17%
System with fixed silence models (*Ms6*)	30.00%	-48.75
Final monophone system (*Ms9*)	38.34%	-28.42%

Table 4.2 shows the recognition results from the monophone system. To show the progress that has been made during the various steps the results from earlier steps are also included. Although the improvements made were considerable, the final systems recognized twice as much words as the initial system and the word accuracy has improved in a similar fashion, the overall results were very modest. The accuracy was even negative. Slight improvements could have been realized by further re-estimation cycles, but the bottleneck of this system is that it is a rather simple continuous speech recognizer. To obtain significant improvement in performance more advanced techniques to refine the system were necessary. These refinements are the subject of the next section.

4.5 REFINEMENT

The monophone system has a number of shortcomings. In the first place, all of the models have the same shape. Further, the system does not take into account linguistic effects like coarticulation, since it uses phoneme units, that are too fine grained to model these effects. Finally, the system does not make up for possible unbalances in the training data, some models may receive much training, while others may receive only little training because they only have a small number of examples. There are a number of ways to refine a speech recognizer system: mixture component splitting, HMM cloning, generalized parameter tying and data driven or tree based clustering. To improve the system a combination of several of these techniques was applied.

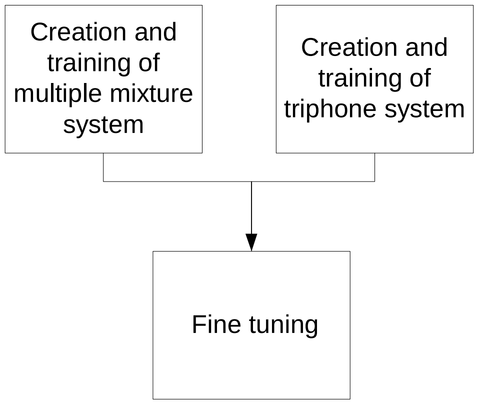
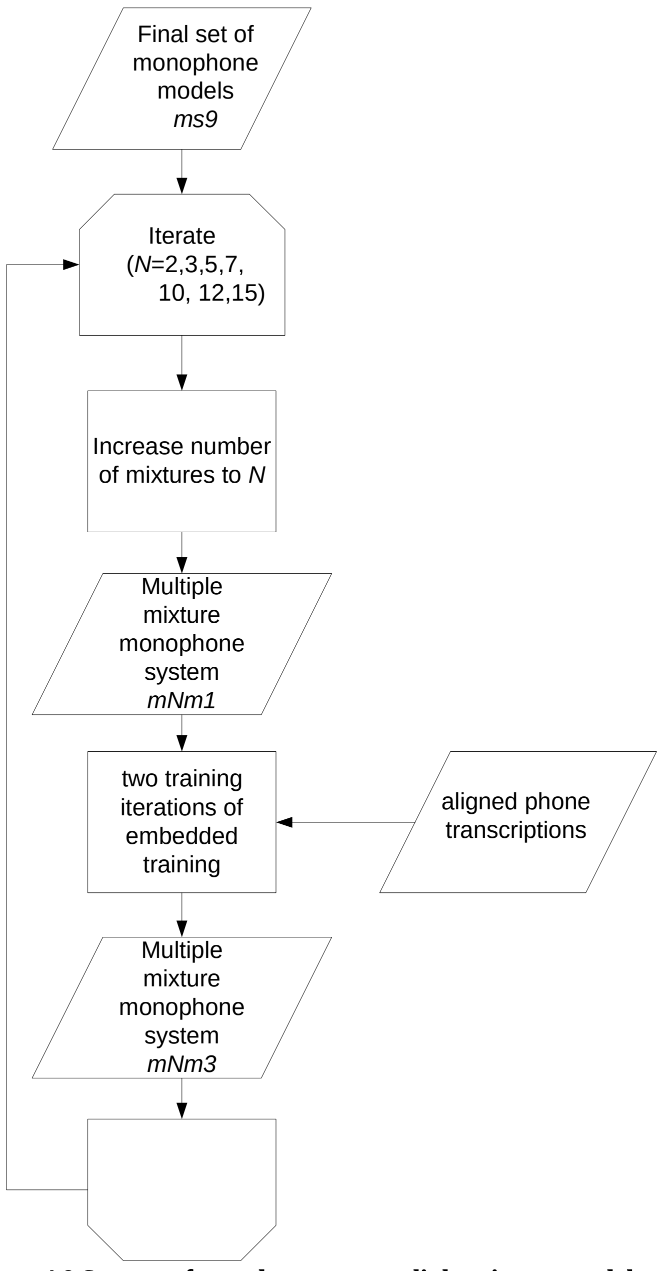

Creation and training of multiple mixture system
Creation and training of triphone system
Fine tuning

Figure 4.7 - Steps performed to refine the system

Figure 4.7 shows the sub-phases that were performed in this phase. Multiple mixture models were created to make up for model inaccuracies. Triphone models were created to better model coarticulation. These model sets were merged and the resulting model was fine-tuned to find the optimal combination of acoustic models and language model.

4.5.1 Multiple Mixtures



Final set of monophone models *ms9*
Iterate (*N*=2,3,5,7, 10, 12,15)
Increase number of mixtures to *N*
Multiple mixture monophone system *mNm1*
two training iterations of embedded training
aligned phone transcriptions
Multiple mixture monophone system *mNm3*

Figure 4.9 Steps performed to create muliple mixture models

The monophone system used Gaussian distributions to model the ‘observation functions’, but as a matter of fact these do not adequately reflect the speech process, They are not capable of modeling the variations in speech that are due to, for example, the difference in pitch between male and female voices. To overcome this problem more realistic density functions have to be found. Alternatively this can be achieved by using a large number of states based on simple densities like the Gaussian density. The latter option implies that an appropriate topology for the phone models has to be found, which is not an easy task. The first option can be achieved by a range of techniques from multivariate statistics, but an interesting solution lies in the use of Gaussian mixture densities, which can approximate any continuous probability density function in the sense of minimizing the error between two density functions. The Gaussian mixture density function is composed by taking the superposition of a number of Gaussian densities each with its own mean and variation and its own mixture weight, as shown in figure 4.10.

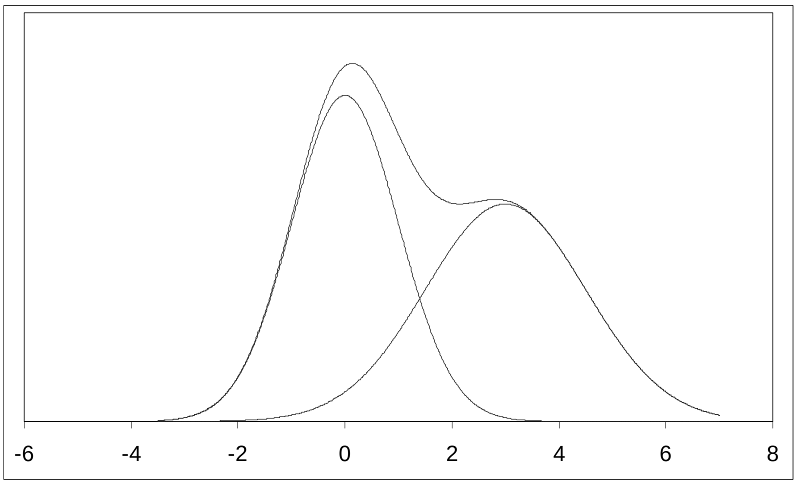


Figure 4.10 - a multiple mixture Gaussian

The observation probabilities are thus computed by

$$b_j(o_t) = \sum_{m=1}^{M} c_{jm} N\,(o_t, \mu_{jm}, \mathbf{U}_{jm}) \tag{4.7}$$

The big advantage of this approach is that with only some slight modifications the Baum-Welch algorithm can also estimate the mixture component weights c_{im}, because each M-component Gaussian mixture state can be viewed as a set of single Gaussian substates. Each with its ingoing transitions weighted by the corresponding mixture weight and its outgoing transition parameter equal to one, as can be seen in the figure below.

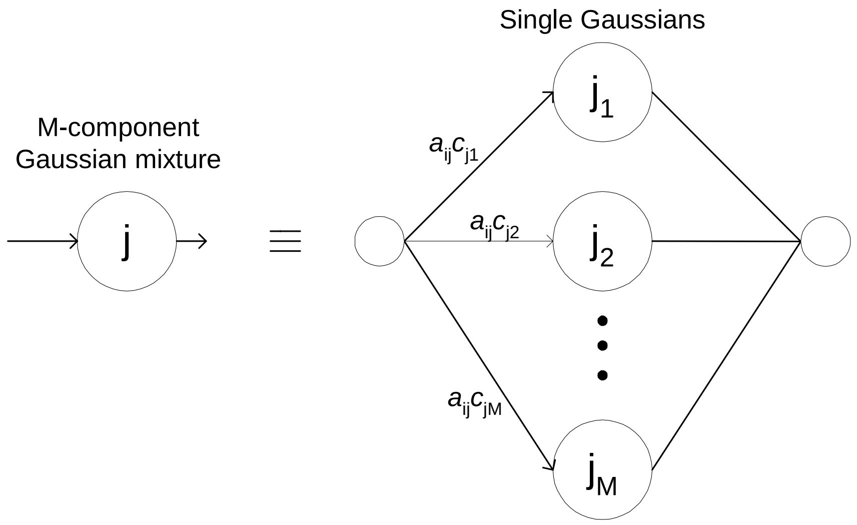


Figure 4.11 - a Gaussian mixture represented as a set of single Gaussians

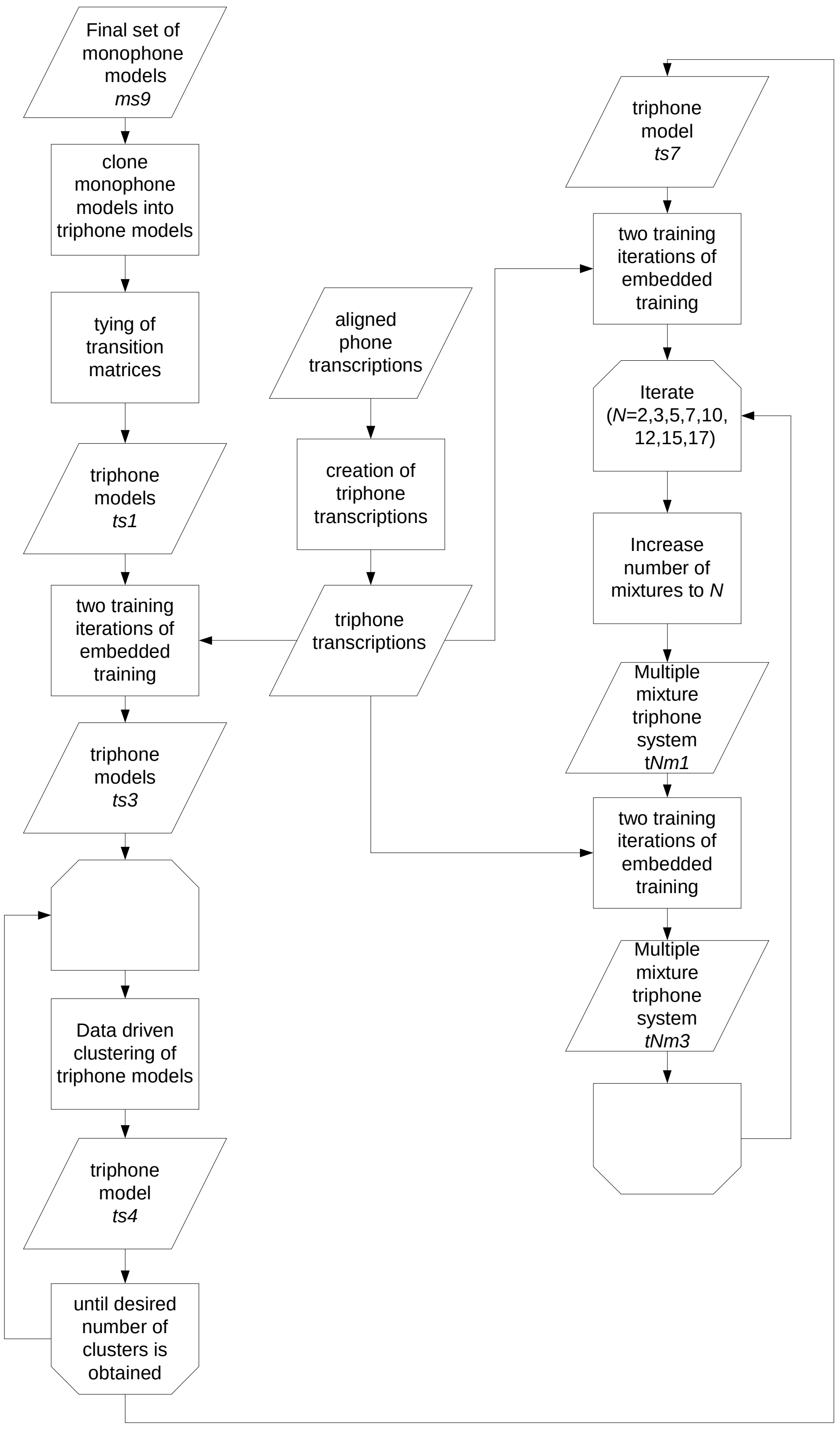
Different models and different states no longer have to have the same type of distribution. Therefore multiple mixture systems may improve recognition results considerably. However, mixture incrementing is not without dangers, the more mixtures are used the better the models fit to the training data, in the end this may result in models that overfit the training data and generalize poor to other data. During mixture incrementing some component weights may become very small, resulting in defunct mixture components. Defunct mixtures often indicate that not enough training data is available to further increase the mixtures of a model.
So the best strategy to adopt here is to increment the mixture components in stages, by incrementing by one or two mixtures a time, then re-estimating, checking recognition results on a test data set and incrementing the mixtures again until the optimum is found.

In figure 4.9 the steps that were performed to build a multiple mixture monophone system are shown. Taking the single Gaussian monophone system from the last section as a basis the mixtures were incremented in seven steps until a 15-mixture system was obtained. This was done by a script for the HMM editor *HHEd*. The script processes each state of the all the models one by one, for the distribution in a state it repeatedly splits the mixture with the largest weight until the required number of components is obtained. Splitting was performed by copying the mixture and dividing the weights of both copies by 2. The means were offset by plus or minus 0.2 times the variance to prevent that one component would be floored while the other would remain as before. To prevent defunct mixtures a floor mixture weight was defined, mixture weights were not allowed to fall below this floor. But during the first steps it turned out that the number of floored mixtures increased rapidly. To reduce this number and prevent overfitting the data the minimum number of examples necessary to allow for mixture incrementing of a model was also incremented in each step.

Table 4.3 monophone mixture systems

Number of mixtures	Percentage of words recognized	Word accuracy percentage
2	40.11%	-25.42%
3	42.70%	-16.33%
5	45.74%	-7.57%
7	47.92%	-1.81%
10	51.54%	4.77%
12	54.50%	9.54%
15	56.81%	15.51%

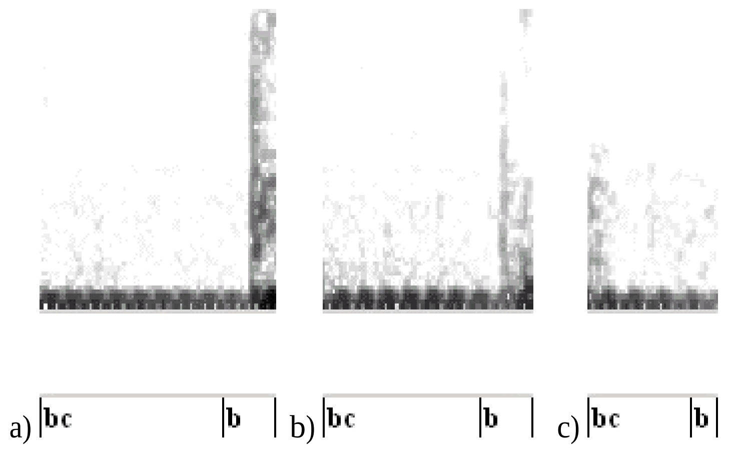
4.5.2 Context dependent models (triphones)



Final set of monophone models *ms9*
clone monophone models into triphone models
tying of transition matrices
triphone models *ts1*
two training iterations of embedded training
triphone models *ts3*
Data driven clustering of triphone models
triphone model *ts4*
until desired number of clusters is obtained
aligned phone transcriptions
creation of triphone transcriptions
triphone transcriptions
triphone model *ts7*
two training iterations of embedded training
Iterate (*N*=2,3,5,7,10, 12,15,17)
Increase number of mixtures to *N*
Multiple mixture triphone system t*Nm1*
two training iterations of embedded training
Multiple mixture triphone system *tNm3*

Figure 4.12 - the creation of multiple mixture triphone models

The phone models described in the systems so far did not depend in any way on their context. A phoneme is assumed to sound more or less the same in every situation. Actually, this is not the case since in normal speech, articulations are made quickly and consecutively and therefore modified by neighboring articulations, as can be seen in figure 4.13.



Figuur 4.13 - the phoneme b and corresponding closure bc in words (a) *big*, (b) *rubber* and (c) *tube*

To capture these effects, called coarticulations, models are needed that take into account the context of a phone. There are many ways to model coarticulations, like for example, modeling all context dependent phones, called allophones by using linguistic theories. But most present-day recognizers have settled on using triphones. Triphones model the context by taking into consideration the left and right neighboring phones. If two phones have the same identity but different left or right context they are considered as different triphones.
Before building a set of context dependent models it is necessary to decide whether cross-word triphones or word internal triphones are to be used. Cross-word triphones are more powerful, since they also take into account coarticulation between words, but because of the large number of possible different cross-word triphones they require large amounts of training data. There are far less word internal triphones, as words are not made up from random phone combinations, so their training data requirements are more modest, but still huge.
In this project it was decided to use word internal triphones. First triphone transcriptions were created that were needed to train the triphone system. This was done by translating the aligned transcriptions created in section 4.4.4 and by replacing the start and end phone in a word by a bi-phone and all other phones by triphones. The *sp*, *sil* and *mn* phones remained monophones. The triphone labels were of the form:

<left context>-<phone name>-<right context>

For example the transcription of the word `het` became:

```
h+e  h-e+t  e-t.
```

As a side effect of this process a list containing all triphones was created. This list was used to create a script for the HMM editor *HHEd*, that cloned each HMM as often as possible and renamed it to a triphone. So, at the end of this process all triphones were exact copies of the corresponding monophone.
This system contained 8570 triphones, each with 3 states and transition matrix and state distributions belonging to it, the number of parameters in this system is thus enormous. To deal

with this problem the number of parameters in the system had to be reduced, to find the right balance between compactness and acoustic accuracy of the individual models.
Parameter tying can do this, when two or more parameter sets are tied, all the owners of the tied set share the same set of parameter values. Figure 4.6 already showed how the states of two models were tied. The figure below shows at which points models can be tied.

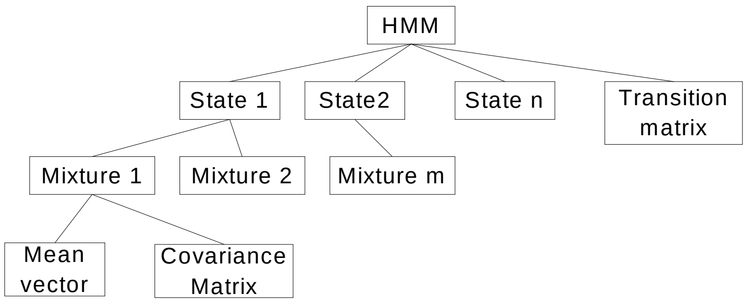


Figure 4.14 - Potential tie points

For context dependent models it can be beneficial to share one transition matrix across all variants of a phone rather than having a distinct transition matrix for each. And applying the argument that context will not greatly affect the center state of triphone models, one way to reduce the total number of parameters without significantly altering the models' ability to represent the different contextual effects might be to tie all of the center states across all models derived from the same monophone.
Although these explicit tyings can have some positive effect they are not very satisfactory. Tying all center states is too severe and the problem of undertraining for the left and right states remains. Therefore most triphone recognizers use a clustering technique to decide which states to tie. Basically there are two mechanisms: Data driven clustering and Tree-based clustering.
Data driven clustering uses a top-down hierarchical clustering procedure: Initially all states are placed in individual clusters. The pair of clusters which when combined would form the smallest resultant cluster are merged. This proces repeats until the size of the largest cluster reaches a certain threshold. The size of the clusters is defined as the greatest distance between any two states. A weighted Euclidean distance between the means of the (single) Gaussians is used.
On completion of the the clustering and tying procedures many of the models may be effectively identical, they are then, so-called generalized triphones. This effect can be exploited to reduce the number of physical HMM's by tying complete models. The resulting models are examples of a hybrid acoustic / linguistic sub-word models, mentioned in chapter 2.3.6.

Tree-based clustering uses phonetic knowledge to come to a clustering. A phonetic decision tree is constructed in which a yes/no question is attached to each node
Initially all states are places at the root node of the tree. Depending on each answer, the pool of states is successively split and this continues until the states have reached the leaf nodes. All states in the same leaf node are then tied. Figure 4.15 shows how the middle states of all triphones corresponding to phone /*at*/ are tied using a decision tree.
Tree-based clustering may lead to better results than data-driven clustering, but this is only true when enough and the right phonetic questions are formulated. This requires a lot of linguistic knowledge and experience, as this was not the subject of this project it was decided to use data driven clustering, in combination with tying of the transition matrices to allow for model tying.
To find the right balance between the number of models and their modeling accuracy about 15 different clusterings were tried, using different clustering-thresholds to find the right number of triphones. Out of these clusterings five different systems were build and each of them was re-

estimated twice. For each system first the transition matrices of the triphones that corresponded to the same monophone were tied.

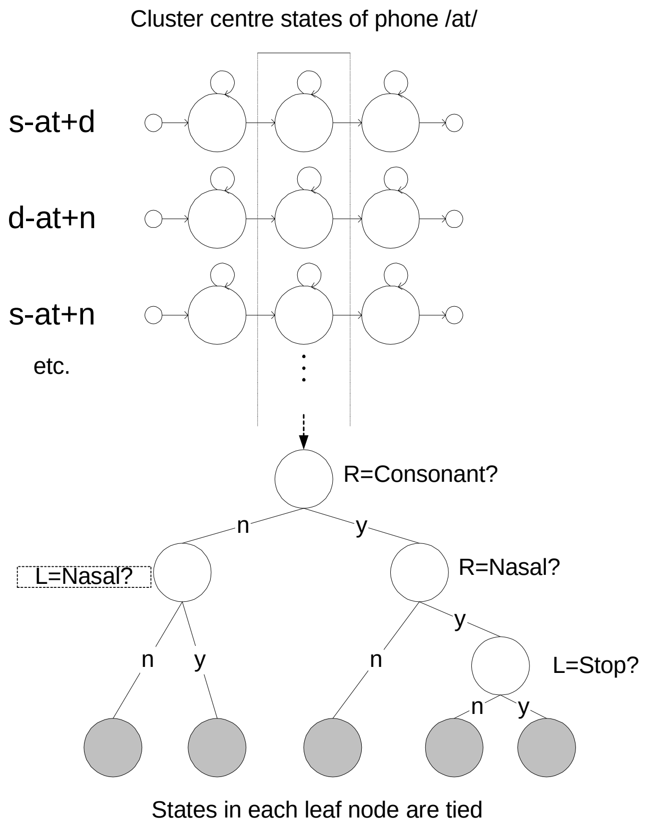

Cluster centre states of phone /at/
s-at+d
d-at+n
s-at+n
etc.
R=Consonant?
n
y
L=Nasal?
R=Nasal?
n
y
n
y
L=Stop?
n
y
States in each leaf node are tied

Figure 4.15 - Decision tree-based clustering

Then the models were clusters as described above and finally models that were effectively identical, because their states were in the same clusters and they had the same transition matrix, were tied. All this was done by a script for the HMM-editor HHEd. These scripts are rather long and complicated because all states and matrices that participate in the clustering have to be specified explicitly. Therefore a program, *TriScr*, was written that, given a list of triphone names and clustering parameters, automatically creates the script.

Table 4.4 - triphone systems

Triphone system	Number of triphones	Percentage of words recognized	Word accuracy percentage
Ts7c1	101	40.44%	-25.46%
Ts7c2	563	46.48%	-10.28%
Ts7c3	1050	49.57%	-4.77%
Ts7c4	2526	54.92%	5.68%
Ts7c5	8570	62.32%	18.59%

As can be seen from table 4.4 the results get better as more triphones are used, but unfortunately this also means a larger model set. The size of *Ts7c5* system, which used no clustering at all, was over 30 Mb, while the size of the *Ts7c1* systems was only 518 Kb. Furthermore, in the *Ts7c5* system there were more than 1000 models which had only one example in the training data, so the risk of overtraining was quite real with this system. The *Ts7c4*, which booked fairly reasonable results, had at least three models per cluster and in most case more. Each cluster had at least four

examples in the training data. This system seemed to provide a good balance between the number of parameters and the modeling accuracy. It contained less than one third of the original triphones, but was still large enough to model different contexts, therefore it was chosen to be further developed during the subsequent steps.
However, before these steps could be performed one problem had to be solved. A limitation of the data-driven clustering procedure is that it does not deal with triphones for which there are no examples in the training data, this may give rise to problems during recognition. This may be avoided by careful design of the training database (as long as word-internal triphones are used) but a little research showed that this was not an option in this case. The training data contained 8570 triphones out of 10205 in the dictionary. Using parts of the testing or evaluation database or relaxing the data selection requirements would only partially solve the problem, as the evaluation data set and the test data set together only contained 394 additional triphones. And doing so would of course have introduced new problems, since the test results would then be biased. Even if this would have helped this solution is still not very satisfactory because there is always the possibility that one day in some recognition job a triphone may show up that is not even in the dictionary. Then it would not be reasonable to claim that the resulting recognizer would be a general continuous speech recognizer.
To overcome these problems an approach that could be described as 'backed-off triphone approach' was thought out. In this approach the original monophone models augment the triphone models. During word network construction triphone models are used whenever available, otherwise the corresponding monophone is used. This is implemented by tying all triphones that have no model of their own to the corresponding monophone. This was done by manually manipulating the HMM definitions since there are no tools that support this particular type of tying. Essentially, the monophones became generalized triphones. But they are less specialized than the other generalized triphones because they contain all corresponding triphones that are now in other sets. Actually they are trained on all the triphones but the ones they represent, but being monophones they lack most context information, so they are general enough to cover the unseen triphones. This results in a robust recognizer that uses triphones most of the time and does not break down when an unknown triphone is encountered.
As with the monophone system the modeling accuracy of the generalized triphone system can be improved by incrementing the mixture components to get better density functions. This was done for the *Ts7c4* triphone system. Once again the mixtures were incremented by two or three at a time, with two re-estimation cycles between increments. The minimal number of examples needed to update a model was increased in each step to reduce the number of floored mixtures and prevent overfitting.

Table 4.5 Multiple mixture triphone systems

Number of mixtures	Percentage of words recognized	Word accuracy percentage
2	56.27%	8.02%
3	58.82%	15.63%
5	61.00%	22.21%
7	64.05%	26.82%
10	66.64%	30.93%
12	68.61%	34.92%
15	70.55%	38.46%
17	71.00%	39.57%

4.5.3 Fine tuning

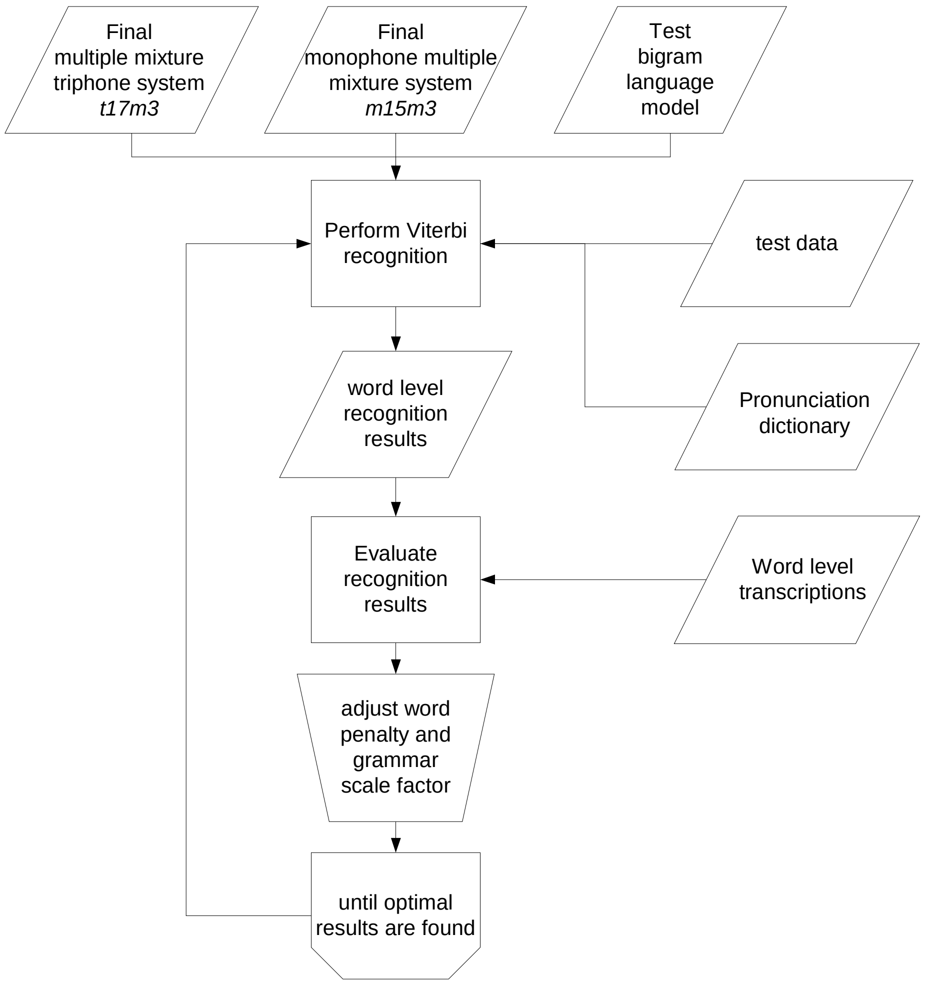

Final multiple mixture triphone system *t17m3*
Final monophone multiple mixture system *m15m3*
Test bigram language model
Perform Viterbi recognition
test data
word level recognition results
Pronunciation dictionary
Evaluate recognition results
Word level transcriptions
adjust word penalty and grammar scale factor
until optimal results are found

Figure 4.16 Fine tuning of the final system

Once the models were sufficiently trained the complete system was fine tuned using a held-out test set, the steps taken are shown in figure 4.16 . The relative levels of insertion and deletion errors can be controlled by adding a fixed word insertion penalty *p*. A negative value of *p* results in less word transitions. A large positive value of *p* would give many short words in the output. These kind of errors and substitution errors can be further controlled by using a grammar scale factor *s*. Every language model log probability *x* will be converted to *s*x-*p* before being added to the tokens emitted from the corresponding word-end node. The grammar scale factor regulates, in a way, the relative influences of the language model and the acoustic model. A grammar scale factor bigger than one reduces the number of insertion errors and tempers the relative influence of the language model with respect to the acoustic model. A grammar scale factor smaller than one increases the relative influence of the language model.
The final acoustic model used was a combination of the 17-mixture generalized triphone set and the 15-mixture monophone set. Varying the word insertion penalty did not improve the overall recognition results, in fact the system proved rather insensitive to this value. Large positive or negative insertion penalties resulted in a decrease in performance of about 2%. The grammar scale factor had a more positive effect. Increasing the grammar scale factor improved the

recognition results considerably. Its maximum was found at s = 9, giving the following results, for the final system:

Table 4.6 – recognition results

Percentage of words correct	95.27%
Word accuracy percentage	89.59%
Percentage of sentences correct	38.43%

4.6 EVALUATION

As can be seen in the last section the generalized triphone multiple mixture system performs rather well. Its results improved from 17% word recognition in the first step of the development process to 95% in the final step. The improvements in word accuracy were even more dramatic; the fist set of trained models had a word accuracy of -77.17%, while the final systems reached 89.59%. But all these results are based on a 1000 word test set, that was used during the process to decide which steps to take and to tune parameters like the cluster size in the triphone set, the number of mixtures and the grammar scale factor. So these results are in a way compromised and likely to be a bit too optimistic. To test the robustness of the system and to see how well it will perform on other data a number of evaluation tests was run.
First recognition was performed on part of the evaluation test set created in the data selection step. This set contained 100 sentences, each of which was spoken by a different person. The bigram wordnetwork used contained 5017 different words. The test gave the following result:

Table 4.7 – recognition results

Percentage of words correct	93.55%
Word accuracy percentage	88.76%
Percentage of sentences correct	32.56%

So the recognizer generalizes very well to speakers it was neither trained nor tuned on even when al larger word network is used.

4.6.1 Using a different data test set

Although the evaluation data did not occur in the training or development test set it also came from the Polyphone database, so it was recorded under similar conditions as the other two sets, in particular it was recorded over a telephone line. As mentioned in 4.1 the speech recognizer should recognize speech recorded by a PC microphone and it should be easily adaptable to specific tasks and applications. To test how well the recognizer generalizes to other data and other environments a small dataset was recorded at TU Delft using a digital video camera. This data set will be described in more detail in chapter 6, for the moment the following information is sufficient. The part of the data set used consisted of 5 different persons, all computer science students at TU Delft, four male students and one female student. From each person 4 or 5 recording sessions were used. Each session contained 23 sentences, ten of which were phonetically rich sentences, similar to those in the Polyphone database. The other sentences contained a sequence of short words, a sequence numbers, a spelled word or a command from a telebanking application that adhered to the grammar of Figure 2.5.
The dataset was split in a training set containing about 500 utterances, that is, about 100 per person and a test set containing 30 utterances. All utterances were stored in the CITT A-law audio format and subsequently MFCC feature vectors were extracted in the same way as described in section 4.3.2. First the system was tested without any further training, which gave the following results:

Table 4.8 – recognition results

percentage of words correct:	87.30%
word accuracy percentage	84.59%
percentage of sentences correct	36.36%

Although still reasonable, the performance clearly decreased in comparison to the performance obtained in the last two sections. These results were to be expected, since the data set was recorded using a video camera in a quiet laboratory, while the Polyphone on which the system was trained was recorded over a normal telephone line. So the ambient noise, which is modeled in the acoustic HMMs, will be quite different, which means that the acoustic vectors in the data set are still similar to those produced by the corresponding HMMs, but there is some distortion, causing some classification mistakes.
To make up for this effect, the system was adapted to the new situation by re-estimating once, using the training part from the recorded data set. This time recognition produced the following results:

Table 4.9 – recognition results

percentage of words correct:	96.76%
word accuracy percentage	95.41%
percentage of sentences correct	60.61%

The performance thus increased considerably. Actually, it is better than any of the results obtained in earlier tests, especially the word accuracy and percentage of correct sentences are very good. The explanation for these results lies in the fact that the system not only adapted to the background noise in this data set, but it also adapted to the voices of these five persons. In fact the system has become a speaker dependent system. the voices it is adapted to are recognized very well, but now recognition of other speakers might give some trouble. To show these effects two more tests were performed. In the first test the system was adapted using only 4 different persons. Recognition was then performed using data from the fifth person. The results below show that although the performance is not as good as the ones in the previous test, the system is still better than the unadapted system. So it adapted to the new environment. Since the person used in the test was not part of the training set, the system is still capable of generalizing and performing speaker independent recognition.

Table 4.11 – recognition results

percentage of words correct:	91.80%
word accuracy percentage	93.44%
percentage of sentences correct	47.62%

To show that the system adapted to the new environment recognition was performed once again on the Polyphone test set using the system that was adapted to four persons, giving the following dramatic results:

Table 4.12 – recognition results

percentage of words correct:	32.16%
word accuracy percentage	-21.44%
percentage of sentences correct	36.36%

Thus the system no longer recognizes the data it was developed with, it has completely adapted to the new environment. That the results are this bad is due to the fact that the polyphone data contains much more noise, since it is recorded over a telephone line, than the data set used here. Recognizing the PC recorded data with an unadapted Polyphone trained system worked because from the systems point of view these were just very high quality recordings. But from the point of view of the adapted system the polyphone data set contains very noisy recordings, indeed many sounds were classified as mouth noise.
As was mentioned, this set also contained sentences that adhered to the telebanking grammar from chapter 2. Using the system that was adapted to four persons recognition was performed on these sentences. A word network that implements the grammar from figure 2.5. was used. This gave rise to the following results:

Table 4.13 – recognition results

percentage of words correct:	75.30%
word accuracy percentage	72.59%
percentage of sentences correct	68.00%

One would expect the percentage of correct words to be higher as only a small vocabulary is used and the syntax of the sentences is constrained. A sentence by sentence inspection of the results showed what was going on here. In most cases the system did recognize the right sentence, but when it made a mistake it often recognized a completely wrong sentence in which only a few words were correct. So about 25 percent of all sentences are responsible for most word errors.

PART II

5 MODELS FOR MULTIMODAL RECOGNITION

This chapter explains why the inclusion of multiple modalities in a speech recognizer may lead to a system that is more robust in noisy environments. The problems in combining multiple modalities are described and a number of ways to integrate data from different modalities in a Hidden Markov based speech recognizer is presented.

5.1 INTRODUCTION

The speech recognizer described in the last chapter performed quite well, but it was tested on audio recordings that did not contain any disturbing background noise or background speech. When there are additional audio sources the performance of the recognizer rapidly degrades. This behavior is common to all present-day ASR systems. The problem is that they have no way of separating the speech signal from other sounds. When speech from multiple persons is picked up it is treated as if it were spoken by one person. However, most real-life environments where speech recognizing applications would be useful are by nature very noisy. For example, currently, a number of speech recognizing telephone inquiry systems is running, these systems do not only have to cope with noise introduced by the telephone channels but as cellular phones become more common also with all kinds of background noises.

To see how ASRs can be made more robust to noise we might want to take a look at the way humans process speech. This seems to be a multimodal process. For example, in a noisy environment people tend to look at the faces of their opponents to understand what is said. This suggests that humans are capable of lipreading. This is also shown by the fact that most people find it rather irritating to watch dubbed movies because the sounds produced do not match the lip movements.

Research has shown that people do indeed use visual information on lip movements to understand speech. This is most clearly demonstrated by the McGurk effect, where a spoken sound /ga/ is superimposed on the video of a person uttering /ba/. Most people perceive the speaker as uttering sound /da/. The visual modality is known to contain some complementary information to the audio modality and what is more important it is not affected by any noise in the environment.

This suggests that speech recognition may be improved by including information on lip movements, that is, by combining the speech recognizer with an automatic lip-tracker. It might also be beneficial to include information from other modalities as well, for example gestures, facial expressions or simply information produced by an other speech-preprocessing tool.

The question that remains is how additional data streams from different modalities can be integrated in the Hidden Markov framework used in speech recognition. The second part of the report presents a number of models that are capable of handling multiple modalities. In chapter five the models are described and in the next chapter some experiments on multimodal integration are discussed.

5.2 PROBLEMS IN COMBINING MULTIPLE MODALITIES

There are five main problems in combining multiple modalities:

1. The signals may have different dynamic ranges. For example the duration of a sound of a phoneme is usually shorter than the duration of the corresponding lip movement.
2. There may be a time offset between the signals. For example in lip reading the video signal usually start before the audio signal. This offset may be as large as 120 ms, almost the duration of a phoneme.
3. There may be a different number of distinguishable classes and therefore different model topologies. For speech phonemes constitute a good set of classes but for other modalities syllables, words or even phrases may be natural classes, for example gestures.
4. The signals may be sampled at different rates. In particular, video sample rates are usually slower than audio sample rates.
5. Some modalities may be more reliable than others. For example audio contains much more information on a speech signal than does visual speech, even in a noisy environment. In some cases the reliability may change dynamically over time. e.g. If people start talking in the background audio recognition may become less reliable.

5.3 CATEGORIES OF MULTIMODAL INTEGRATION

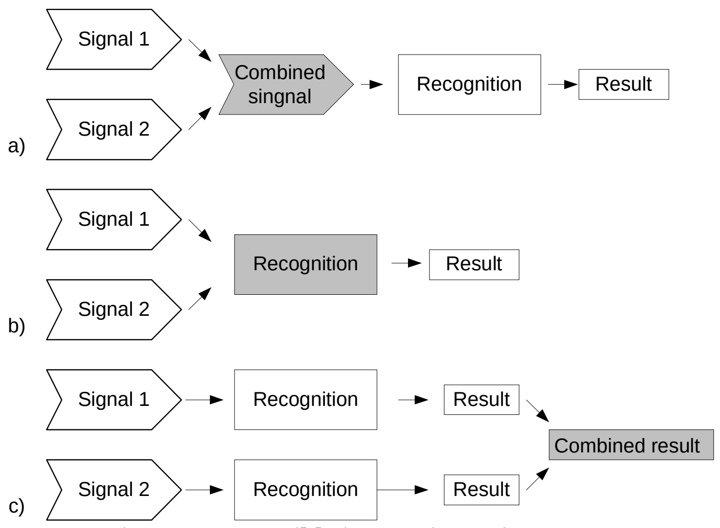


Figure 5.1 – possible integration points

There are three categories of multimodal integration. They differ in the point of the process where integration takes place. These constitute:

1. *Feature fusion* - features are extracted from the raw data and are combined before recognition.
2. *Decision fusion* or *model fusion* - the integration takes place within the HMM, at various possible stages. For example state level, phone level or word level.
3. *Late integration* - The different modalities are processed by different recognizers and subsequently their outputs are combined to get the final result.

Each of these categories will now be discussed in detail.

5.4 FEATURE FUSION

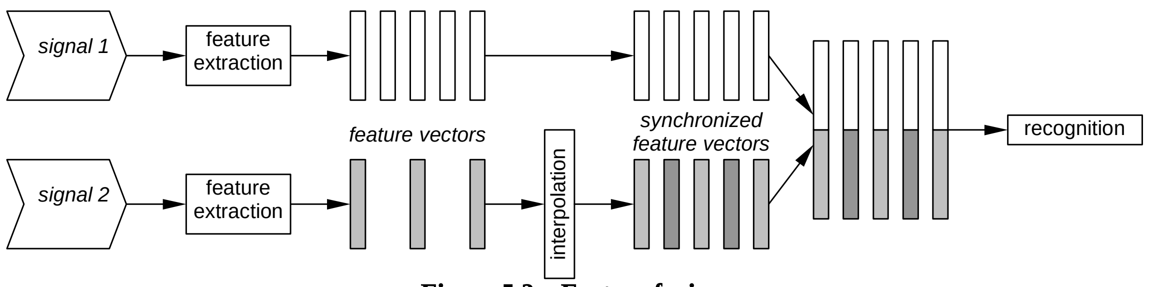


Figure 5.2 – Feature fusion

In feature fusion, features vectors from the different modalities are first extracted from the raw signals using appropriate preprocessors, like a LPC-decoder in the case of speech recognition, and then concatenated to generate a single vector on which an HMM based recognizer can be trained in the normal way.
In many cases, the frame rates of the different modalities do not have to be the same. As pointed out above video frame rates are typically lower than audio frame rates. This is usually circumvented by inter-frame interpolation.
A problem with this fusion technique is that it can lead to rather high dimensionalities, which can cause inadequate modeling because of the large number of parameters that are introduced this way. To avoid this problem more advanced methods seek to reduce the size of the concatenated vectors before training HMMs on it. This can be achieved by projecting the feature space to a lower dimension by means of statistical techniques like Linear Discriminant Analysis (LDA) or Principal Component Analysis (PCA). Alternatively a feature selection method from the field of pattern recognition can be used to find a subset of the features having the most discriminative power.
Training of a model on concatenated feature vectors can be done in two ways. The system can be trained from scratch or separate systems for each modality can be trained and then integrated to form the multi-modal system. Which can then further be trained using the concatenated vectors. The latter approach only works if the separate systems have the same model topology and no dimension reduction is performed after concatenating the vectors. But in can be advantageous if recognizers for one or more modalities are already available or if one of the modalities needs far more training or more training data to produce adequate models than others.
Feature fusion is a nice simple technique that uses the same type of models and algorithms as normal speech recognition. But is has some significant drawbacks. First of all it assumes vector synchrony between all its input data streams. Therefore it is not very well suited to handle signals with different dynamic ranges or with a time offset. The HMM concept is capable of partially catching such effects, given enough training data, but this usually results in weak sub-optimal models as most variations will be modeled as noise.
Secondly, the number of distinguishable classes and the model topology of the units in these classes have to be the same for each modality or the classes of one modality have to be superimposed on the others, possibly resulting in poor modeling.
Thirdly, this approach does not explicitly model the reliability of each modality. The influence the modalities get is mostly based on the relative lengths of their vectors.
Furthermore, the feature fusion technique cannot handle different classifications of different modalities as it uses a common recognizer for all of them. Whether this constitutes a problem depends of course on the application. The same is true for the property of the feature fusion algorithms, that they do not make any conditional independence assumptions between

modalities. The decision fusion techniques in the next section, on the other hand, do make this assumption.

5.5 DECISION FUSION

In decision fushion separate data streams are put into the recognizer. The integration takes place within the Hidden Markov models. This can be done at various levels, like the state level, the model level or for instance the phoneme or word level.

There are two basic kinds of couplings obtainable by extending the HMM framework. The weakest is when two independent processes are nominally coupled at the output, superimposing their outputs in a single signal. This is called the source separation problem, signals with zero mutual information are overlaid in a single channel e.g. different voices at a cocktail party or different instruments in a symphony.

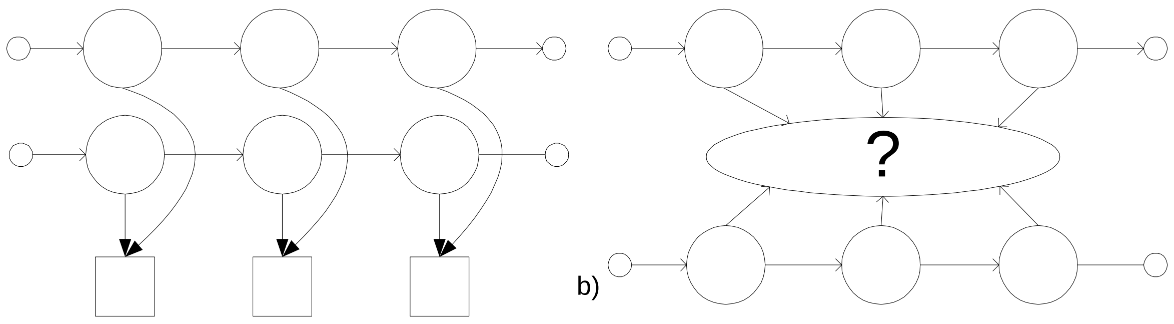


Figure 5.3 – a) nominally coupled b) dependent processes

In stronger couplings, the processes are dependent and interact by influencing each others' states. In this situation, called sensor fusion, multiple channels carry complementary information about different components of a system, e.g. acoustic signals from speech and visual features from lipreading.
Many problems tend to lie between the two extremes. Two processes may interact without wholy determining each other. Each process has its own internal dynamics and is influenced by others, possibly causally. A variety of architectures have been proposed in literature, differing in the way they model the influences of the processes on each other.
In this section a number of these models will be presented. The Cartesian product Hidden Markov model is probably the most intuitive approach but unfortunately also the most naive approach. The Factorial Hidden Markov model is an example of a weak coupled system while the Linked Hidden Markov model, the Hierarchical Hidden Markov model and the Coupled Hidden Markov model exhibit much stronger couplings. The Multistream and Product Multistream Hidden Markov models lie somewhere in between, modeling processes that are independent between certain boundary points.

5.5.1 The Cartesian product Hidden Markov model

The first solution that comes to mind to integrate two or more data streams that can be modeled using HMMs is to build one large model having the Cartesian product of all the states of the separate models as its states. This is shown in figure 5.4 But this naive solution is rarely satisfactory. The size of the model explodes, leading to high computation costs and the huge

amount of parameters in this model leads to overfitting. Often there is insufficient data for a large number of states, usually resulting in oversampling.
Even with the correct number of states and vast amounts of data, large HMMs often train poorly because the data is partitioned among states early and possibly incorrectly during training. The Markov independence structure then ensures that the data is not shared by states, thus reinforcing any mistakes in the initial partitioning. Systems with multiple processes have states that share properties and thus emit similar signals. The Markovian framework works against this systematically.

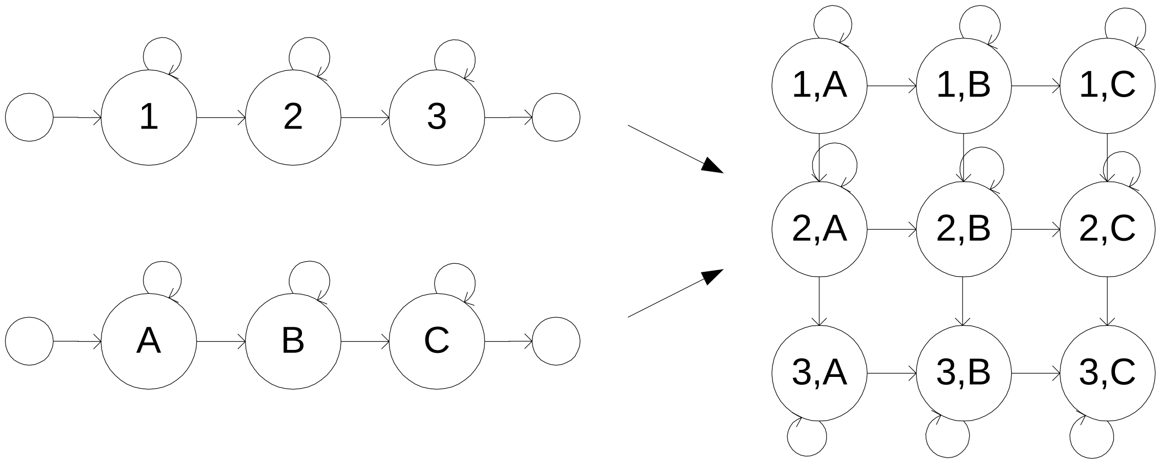


Figure 5.4 - two 3 states HMMs combined in a Product HMM

To model *K* processes each with *N* states would result in a product HMM with N^K composite states. The time complexity of the dynamic programming algorithm that lies at the heart of the forward-backward and Viterbi algorithms then becomes $O(TN^{2K})$, thus exponential in the number of states of the separate models. This becomes quickly intractable as the number of datastreams increases. Things are made worse by the fact that product HMMs tend to be densely connected.

All models in this category try to improve on the Cartesian Product HMM by placing constraints on the states or the transitions of this model. This results in models with compositional state representation that are more compact and clearer than the Cartesian Product HMM and which offer better modeling accuracy. Most techniques are only capable of improving on the computational costs of the Product HMM by resorting to approximations of the exact algorithms.

5.5.2 The Factorial Hidden Markov model

Factorial Hidden Markov models extent HMMs by allowing the modeling of several random processes that are loosely coupled. They were first described by Gharhamni and Jordan [28], who also provided several algorithms to efficiently learn the parameters of the models. In the original description FHMMs were mainly used as an advanced mathematical tool for modeling time series. But as the focus here is the applicability of FHMMs to multimodal speech recognition, they will be described as an extension to HMMs. To see how the Factorial Hidden Markov model extents the HMM it is convenient to use a different representation of HMMs, that shows the evolution of the state sequence, and the accompanying observations, with time as depicted in figure 5.5. The circles in the picture represent states while the boxes represent the output a state.

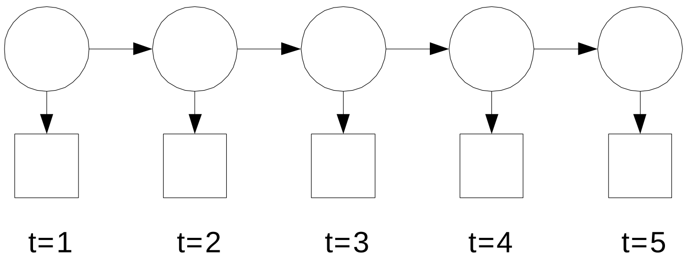


Figure 5.5 - HMM depicted rolled out in time - a dynamic belief network

Essentially, this is a dynamic belief network. Each node only depends on its direct parents, as indicated by the arrows. Now the Factorial HMM arises by forming a dynamic belief network composed of several layers, shown in figure 5.6.

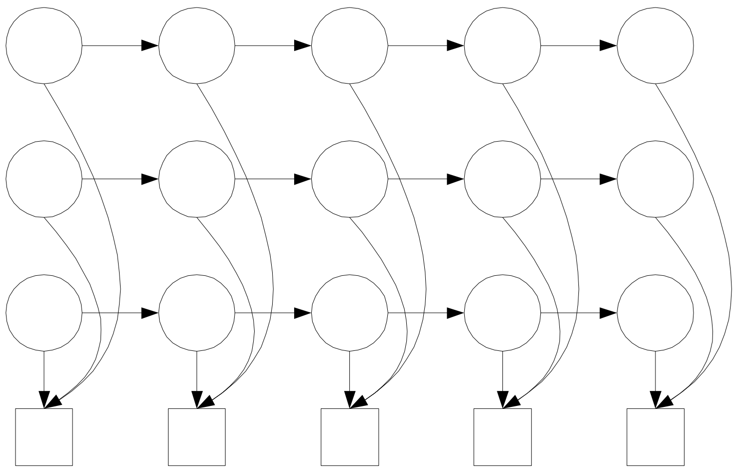


Figure 5.6 - FHMM as DBN

Each layer has independent dynamics but the output observation vector depends upon all the state variables of the current time step. This is achieved by letting the states be represented by a collection of state variables:

$$q_t = q_t^{(1)}, q_t^{(2)}, \ldots, q_t^{(K)} \tag{5.1}$$

Where the superscript is the layer index, with K being the number of layers. Each of these sub-states $q^{(i)}$ can take on $N^{(i)}$ values. For simplicity assume that $N^{(i)} = n$ for all i, rather than assuming that the number of possible states within each layer is different, although the model can easily be generalized to include these cases.
The complete state space now consists of the cross product of these sub-state variables, hence the name factorial Hidden Markov models. Placing no constraints on the state transition structure would result in a N^MxN^M transition matrix that would be equivalent to the naive cartesian product HMM, with the same exponential time complexity. Therefore the underlying state transitions are constrained by letting a sub-state depend only on its direct predecessors in the same layer, as in simple HMMs, and letting it evolve independent of sub-states in other layers. Thus making the different layers completely independent. Each of them forms a basic HMM, as

was already shown in figure 5.6. The transition probabilities for the entire system are now computed by taking the product of the transition probabilities:

$$a_{q_{t-1}q_t} = P(q_t \mid q_{t-1}) = \prod_{k=1}^{K} P(q_t^{(k)} \mid q_{t-1}^{(k)}) \tag{5.2}$$

The transition structure for this system can be represented as K distinct NxN matrices.
In a factorial HMM the observation at time step t depends on all the state variables at that time step. For continuous observations a possible form of the distribution function, as proposed in the original paper by Jordan, is linear Gaussian, i.e. the observation function $b(o_t)$ is a Gaussian whose mean is a linear combination of the state means.

$$b(o_t) = P(o_t \mid q_t) = \exp\left\{\frac{1}{2}(o_t - \sum_{k=1}^{K} \mu^{(k|q_t)})^T \mathbf{U}^{-1}(o_t - \sum_{k=1}^{K} \mu^{(k|q_t)})\right\} \tag{5.3}$$

In which $\mu^{(k|q_t)}$ is the mean of layer k given the meta-state q_t and $\mathbf{U}$ is a common tied covariance matrix.
A second method to compute the probability of the observation is simply the product of the Gaussian distributions of each layer. This is called the streamed method, as each layer of the FHMM now models a stream of the observation vector.

$$b(o_t) = \prod_{k=1}^{K} N\,(o_t, \mu^{(k|q_t)}, \mathbf{U}^{(k|q_t)}) \tag{5.4}$$

Estimation of the parameters of a factorial HMM can be done via the expectation maximization algorithm, Baum-Welch algorithm in the case of HMMs, that fixes the current parameters and computes posterior probabilities over the hidden states (E step). And then uses these probabilities to maximize the expected loglikelihood of the observations as a function of the parameters in the M step.
The M step for factorial HMMs is simple and tractable. The calculations for the initial values π and for the state transition matrices are even identical to the ones in the Baum-Welch algorithm. But, unfortunately, the exact E step for factorial HMMs is computationally intractable. This is due to the fact that the hidden state variables at one time step, although marginally independent, become conditionally dependent given the observation sequence. Since $b(o_t)$ is a function of all the state variables, the probability of a setting of one of the state variables will depend on the settings of the other state variables. The naive exact algorithm of translating the factorial HMM into an equivalent HMM with KN states and using the forward-backward algorithm, has time complexity O(TN^{2K}) like the Product HMM and is thus intractable.
The problems of finding the most likely model or the most likely path, which is really a set of sub-paths in this case, through a model, are intractable for the same reason. There are several approximations proposed in literature using for example a Monte Carlo sampling procedure or an approximating of the distribution function over the hidden variables $P(q_t \mid o_t)$ by a tractable distribution $Q(q_t)$ assuming that all state variables are independent. In the latter group of algorithms there exists an approximation that is both tractable and preserves much of the probabilistic structure of the original system. In this scheme the factorial HMM is simply approximated by K uncoupled HMMs. Within each HMM efficient and exact learning is implemented via the classical forward-backward algorithm.

A dynamic programming procedure to compute the Viterbi path using this approach still requires O(Tn^{2n}) steps, because corresponding subpaths have to be found in each layer of substates. As a practical alternative an iterative approach can be used that returns a possibly suboptimal path in polynomial time. The iteration is based on a subroutine that finds the optimal path of hidden states through the nth layer given fixed values for the hidden states of others. The effective size of the state space collapses from n^n to n and the optimization with respect to the remaining hidden states can be performed in O(TKn^2) steps.
The chainwise Viterbi algorithm for approximately computing the full Viterbi path of the factorial HMM is obtained by piecing these subroutines together in the obvious way. First an initial guess is made for the Viterbi path of each component HMM by ignoring the intercomponent correlations and computing a separate Viterbi path for each chain. Then the chainwise Viterbi algorithm is applied, in turn, to each of the component HMMs. After the Viterbi algorithm has been applied n times, or once to each chain, the cycle repeats. Each iterations results in a sequence of hidden states that is more probable than the preceding one. Hence, this process is guaranteed to converge to a final though possibly suboptimal path. The chainwise Viterbi algorithm is not guaranteed to find the truly optimal sequence of hidden states for the factorial HMM. The approximation is premised on the assumption that the model describes a set of weakly coupled time series, in particular, that the auto-correlations within each time series are stronger that the cross correlations between them.
The Factorial HMM is an example of a nominal coupling of different data sources. Its separate streams evolve completely independent from each other, whether this is appropriate or not depends of course on the processes that are to be coupled. For example, in [29] it is shown that this model is not appropriate for integrating speech recognition and automatic lipreading, since there are too many causal dependencies between these signals. The layers of the Factorial Hidden Markov model do not have to contain the same number of states, but as the layers evolve in lockstep, they all have to contribute to the output signal at each time step the FHMM does not allow for different dynamic ranges. Furthermore the FMM has no way of modeling layer reliability.

5.5.3 The Linked Hidden Markov model

A possible way of coupling for dependent processes to address the sensor fusion problem is shown in figure 5.7.

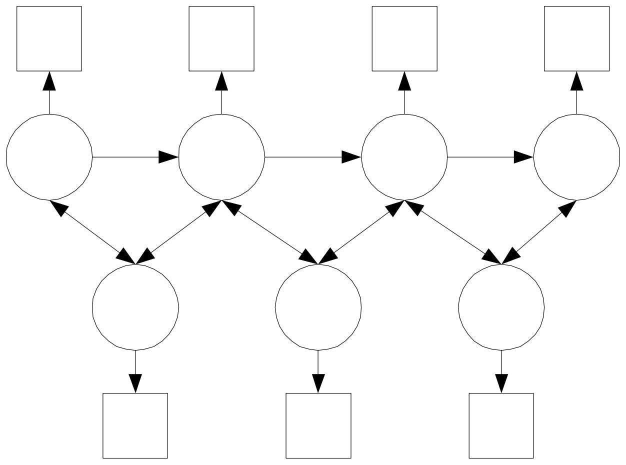


Figure 5.7 - The Linked HMM rolled out in time

This architecture is called the Linked Hidden Markov model, it has joint probabilities between synchronous states, encoding which state pairs co-occur more or less frequently. Its independence structure is suitable for expressing symmetrical synchronous constraints, like the fact that it is rare to hear a sound when someone's mouth is closed. The LHMM is equivalent to the Cartesian product HMM with a bias probability on each joint state as depicted in figure 5.8. The state of the first two-state chain are numbered A, B and the state of the second chain are numbered C,D in this figure. The states of the Product HMM are number 1 to 4, a_{ij} is the transition probability from state *I* to state *j* in the product HMM and C_{IJ} is the probability of the coupling between state *I* in chain one and state *J* in chain 2.

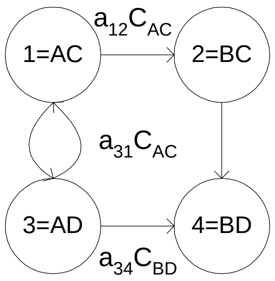


Figure 5.8 – the Linked HMM is equivalent to the Product HMM

Contrary to the Factorial HMM this model can produce different outputs for each modality. It allows for processes with different time scales but does not model the reliability of the modalities. As a consequence the training and inference algorithms for the Linked HMM are intractable but there exists an $O(TN^3)$ exact algorithm for training the two chain structure based on methods from statistical mechanics. The model can also be viewed as a simplification of Coupled HMMs that are described in section 5.2.5.

5.5.4 The Hierachical Hidden Markov model

In some situations one process clearly imposes constraints on another process. For example the baseline of a song constrains the melody and both constrain the harmony. This kind of hierarchical structure in a signal can be modeled by the Hierarchical Hidden Markov model shown in figure 5.9.

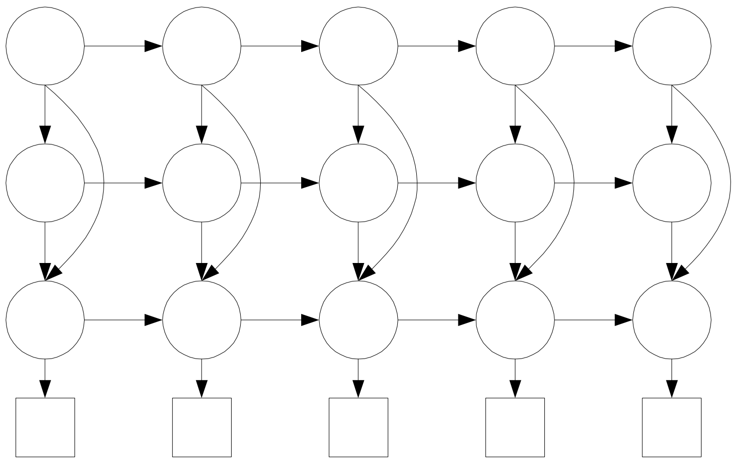


Figure 5.9 - HDTM rolled out in time

This structure which consists of a cascade of synchronous conditional probabilities down an ordered hierarchy of HMMs arised by giving probabilistic decision trees a markovian temporal structure. The model is intractable for exact calculations but there exist approximation algorithms that decompose the structure either in a number of HMMs or a number of probabilistic decision trees.

5.5.5 The Coupled Hidden Markov model

The previous two architectures have synchronous links and assume lockstep processes that do not have causal temporal influences on each other. To capture interprocess influences across time, the coupling must bridge time slices as shown by the crosswork of conditional probabilities in figure 5.10. This offers the strongest model of interprocess influences. It is appropriate for processes that influence each other assymetrically and possibly causally. The model that realizes this structure is called the Coupled Hidden Markov model.

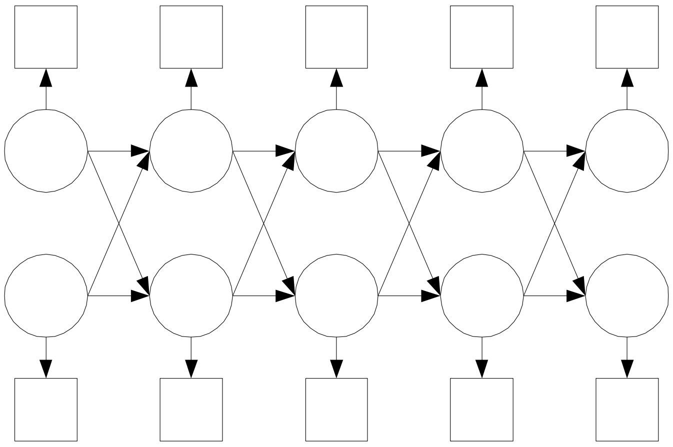


Figure 5.10 CHMM rolled out in time

Inference in Coupled HMMs has, once again, the same time complexity as the Cartesian product HMM, thus the exact algorithms are not attractive. Brand [23] describes an $O(T(KN)^2)$ N-heads dynamic programming algorithm obtained by relaxing the assumption that every transition in the trellis must be visited. This algorithm still visits every component state so that statistics may be collected for re-estimating transition and output probabilities.

The probability of a state sequence through a fully connected K chain given the multimodal observation sequence **O** is:

$$P(\mathbf{q}|\mathbf{O}) = \prod_{k=1}^{K}\left(\pi_{q_1^k} b_{q_1^k}(o_1^k)\prod_{t=2}^{T}\left(b_{q_t^k}(o_t)\prod_{l=1}^{K} P_{q_t^k|q_{t-1}^l}\right)\right)\Bigg/ P(\mathbf{O}) \tag{5.5}$$

where superscripting indexes a chain. $P_{q_t^k|q_{t-1}^l}$ is the probability of a state in chain k given a previous state in chain l, when $l = k$, this probability represents a transition otherwise it represents a coupling.
As was described in chapter one direct computation of these probabilities is intractable even for simple HMMs, therefore dynamic programming is used for the Viterbi and forward algorithms.
A coupled HMM of K chains has a joint state trellis that is in principle N^k states wide. The associated dynamic programming problem is $O(TN^{2k})$. The N-heads dynamic programming algorithm closely approximates the full combinatorial approach. It uses a subset of all state

sequences consisting of paths with the greatest probability mass applying the argument that low-probability sequences carry relatively little information for estimation problems. This is the same assumption as made by the beam search described in chapter 2.3.8.
Each state sequence through the trellis is double tracked, having a head in one HMM $h_{i,t}$ as in the normal trellis algorithms and an associated sidekick $k_{i,t}$ in the other HMMs. A sequence of {head, sidekick} pairs is called a path. Coupling two HMMS of N and M states now takes $N+M$ heads each with a sidekick. Three HMMs takes $N+M+L$ heads each with two sidekicks etc. Figure 5.11 shows a sub-path through a trellis with an associated sidekick sub-path through the trellis of the second chain. Arrows represent transitions, dashed arrows couplings.

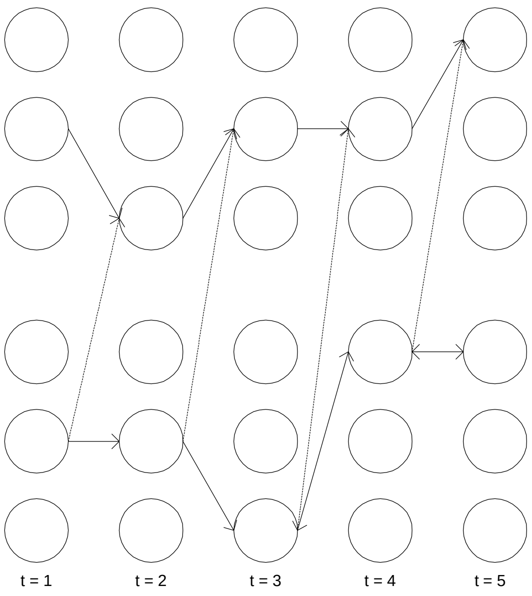


Figure 5.11 – a two level trellis showing a path and its sidekick

In the forward algorithm at each time step every head in the chain sums over the same set of paths and thus shares the same sidekick. This sidekick is simple the maximum marginal forward variable in the other HMM. New path probabilities are calculated given the paths sofar and the sidekicks. To calculate the forward variable associated with each head, the sidekicks are marginalized out.
For a CHMM with two chains the algorithm contains the steps below, where i and j denote states in the first chain and i' and j' denote states in the second chain. $k_{j,t}$ denotes the sidekick associated with state i in time step t and c denotes a coupling matrix. The transition matrix of the first chain is denoted by a and the transition matrix of the second chain by $a^{'}$.

1. Calculate all partial forward variables $p_{t+1}(i)$:

$$p_t(j) = \left[\sum_i a_{ij} c_{k_{j,t-1}} \alpha^{*}_{t-1}(i) \right] b_j(o_t) \tag{5.6}$$

2. Choose one best sidekick from each chain:

$$k_{j,t} = \arg\max_{i'}(p_t(i')) \tag{5.7}$$

3. Calculate the full forward variables α^* for each path in the entire model, this variable is used for propagating probabilities:

$$\alpha_t^*(j) = b_j(o_t)\sum_i a_{ij} c_{ik_{j,t}} c_{k_{i,t-1}j} a'_{k_{i,t-1}k_{j,t}} \alpha_{t-1}^*(i) \tag{5.8}$$

4. Marginalize out each head over all possible sidekicks to obtain the forward variable in each chain. This variable is used for re-estimating parameters.

$$\alpha_t(j) = b_j(o_t)\sum_i a_{ij} c_{k_{i,t-1}j} \sum_g b_{k_{g,t}}(o_t) c_{jk_{g,t}} a'_{k_{j,t-1}k_{g,t}} \alpha_{t-1}^*(i) \tag{5.9}$$

The backward variables are calculated in a similar way using the sidekicks found in the forward algorithm. Viterbi analysis also works the same way but uses a maximization over previous paths instead of a summation. In this algorithm each head can have a different sidekick. If the dynamics within a chain dominate the interchain dynamics than the chainwise Viterbi algorithm described in section 5.3.3 can be used. In which a conventional Viterbi analysis is performed in one chain while holding state assignments constant in all others and is cycled through the chains until convergence to a local maximum is reached.

After collecting statistics using the *N*-heads method, transition matrices within chains are re-estimated according to the conventional HMM formulae. The formula for coupling matrices between HMMs is similar:

$$P(q'_t = i, q_{t-1} = j \,|\, \mathbf{O}) = \frac{\alpha_{j,t-1} c_{ji'} b_{q'_t=i}(o'_t) \beta_{i',t}}{P(\mathbf{O})} \tag{5.10}$$

$$\hat{c}_{i'j} = \frac{\sum_{t=2}^{T} P(q'_t = i, q_{t-1} = j \,|\, \mathbf{O})}{\sum_{t=2}^{T} \alpha_{j,t-1} \beta_{j,t-1}} \tag{5.11}$$

The approximation will of course weaken as more and more HMMs are coupled, since an exponentially small fraction of the paths are being sampled. The transition probabilities of a coupled HMM can be converted to and from those of a Cartesian product HMM. This forms the basis for an alternative training algorithm in which a Cartesian product HMM is factored and recoupled between each re-estimation. Compared to the *N*-heads method this algorithm has inferior complexity and accuracy, but it compares favorably to other algorithms.
The CHMM runs considerably faster than the Cartesian product HMM. But it does not have any complexity advantages over these methods. Its main advantage is that it explicitly represents interactions between HMMs and thus can capture system dynamics in the data that product HMMs cannot. Because the CHMM has less parameters than the Product HMM it also requires far less training data. For example a fully connected two chain CHMM with N states per chain has $2N$ state distributions each with a mean and a covariance matrix and $O(3N^2)$ transition and coupling parameters, the corresponding Product HMM has N^2 state distributions and $O(N^4)$ transition parameters.
Conventional HMMs are quite sensitive to the initial values of the parameters. Experiments have showed that LHMMs are generally more robust and CHMMs are far less sensitive to initial conditions.

The dynamic programming algorithms used in conventional HMMs automatically handle variations in tempo. In CHMMs this extends to variations in process synchronization.
All in all the Coupled Hidden Markov model is a pretty powerful model it can model a wide variety of couplings by making the coupling parameters stronger or weaker. By handling variations in process synchronization it allows for offsets between signals and it can handle different time scales. It also allows for different outputs from different modalities. Its only drawbacks are that it requires special algorithms that can only approximate the exact results instead of the regular HMM algorithms and it is not capable of modeling the reliability of the modalities.

5.5.6 The Multistream Hidden Markov model

The multistream approach processes several feature streams in parallel and independently of each other up to certain anchor points where they have to synchronize and recombine their partial segment-based likelihoods. Thus a certain level of asynchrony between the streams can be modeled. Furthermore the different streams are not restricted to the same frame rate and the underlying HMM models associated with each stream do necessarily have to have the same topology.
The multistream statistical model assumes that the observation sequence **O** is composed of *K* input streams $\mathbf{O}_k$ from different sources representing the utterance to be recognized. It further assumes that the a model *M* is composed of *J* sub-unit models M_j ($j = 1,\ldots,J$) associated with the sub-unit level at which to perform the recombination of the input streams. To process each stream independently up to the defined sub-unit level, each sub-unit model M_j is composed of parallel models M_{jk}, possibly with different topologies, that are forced to recombine their respective segmental scores at some temporal anchor points. The model is shown in figure 5.12.
To make things a little clearer imagine a combined speech, lipreading recognizer in which the models represent words and the sub-units represent phonemes. Now within the boundaries of a phoneme the signals are allowed to evolve independent of each other but at each phoneme boundary they have to evolve.

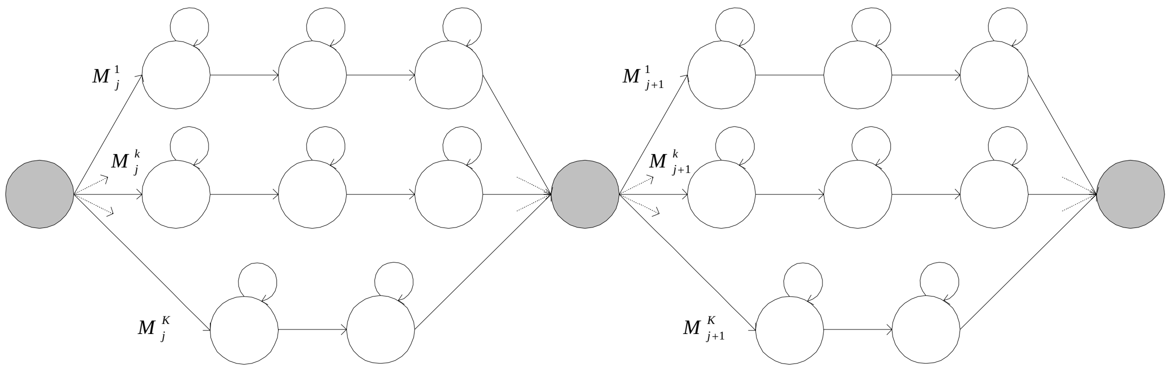


Figure 5.12 - The Multistream HMM

The gray recombination states in the figure are not a regular HMM state, since it will be responsible for combining probabilities accumulated over a same temporal segment for all the streams according to the rules discussed below.
The recognition problem for a likelihood based system can then be formulated in terms of finding the model *M* maximizing:

$$P(\mathbf{O}|M)=\prod_{j=1}^{J}P(\mathbf{O}_j|M_j) \qquad (5.12)$$

Where $\mathbf{O}_j$ represents the multiple stream subsequence associated with the sub-unit model M_j.
In the case in which the streams are assumed to be statistically independent the full likelihood can be decomposed into a product of stream likelihoods for each sub-unit. For this we can simply compute:

$$\log P(\mathbf{O}|M)=\sum_{j=1}^{J}\sum_{k=1}^{K}\log P(\mathbf{O}_j^k|M_j^k) \qquad (5.13)$$

This approach can be generalized to a weighted log likelihood approach to model the reliability of the different input streams. We have then:

$$\log P(\mathbf{O}|M)=\sum_{j=1}^{J}\sum_{k=1}^{K}w_j^k\log P(\mathbf{O}_{_j}^k|M_j^k) \qquad (5.14)$$

where w_j^k represents the reliability of input stream k. More generally, we may also use a non-linear system to recombine probabilities or log-likelihoods so as to relax the assumption of the independence of the streams. More generally, a nonlinear system to recombine probabilities or log-likelihoods so as to relax the assumption of independence of the streams.

$$\log P(O|M)=\sum_{j=1}^{J}f(W,\log P(O_j^k|M_j^k)) \qquad (5.15)$$

where W is a global set of recombination parameters and f is a nonlinear function that combines the probabilities of all chains.
The recombination of the input streams at a sub-unit level, e.g. word, syllable, phoneme, requires a significant adaptation of the recognizer algorithms, requiring a time synchronous Viterbi search that allows the decomposition of a single stream into independent components.
On the other hand recombination at the HMM state level is pretty simple to implement and amounts to performing a standard Viterbi decoding in which local log probabilities are obtained from a linear or non-linear combination of the local stream probabilities. Of course this approach does not allow for asynchrony beyond the state level and requires therefore identical topologies for the sub models. Yet, it has been shown to be very promising in experiments conducted by a number of researchers. Some authors actually do refer to this particular instance of the model as the multistream model. The remainder of this section is devoted to this model.

Training of the multistream HMM consists of two tasks. First, estimation of its stream component parameters, like mixture weights, means and variances, as well as, of the HMM state transition probabilities has to be performed. And second, appropriate exponents that sum to one have to be estimated. Maximum likelihood parameter estimation by means of the standard Baum-Welch algorithm can be used in a straightforward manner to train the first set of parameters. This can be done in two ways: Either train each stream component parameters set separately, based on single-stream observations, and subsequently combine the resulting single-stream HMM as in 5.12 or train the entire parameter set, excluding the exponents, at once using the multimodal observations.
In the first case the Baum-Welch algorithm is involved to separately train two single-modality single-stream HMMs as described in part I. Thus assuming known stream exponents, the two

resulting HMMs can be easily combined using emission probabilities given by 5.12 and a linear combination of their two transition matrices, weighted by stream exponents that sum to one . An obvious drawback of this approach is that the two single modality HMMs are trained asynchronously, i.e. they are trained using different forced alignments, whereas 5.12 assumes that the HMM stream components are state synchronous.
The alternative is to train the whole model at once, in order to enforce state synchrony. Due to the stream log-likelihood linear combination by means of 5.12 the EM-algorithm carries on in the Multistream HMM case with minor only minor changes. The only modification is that the state occupation probabilities are computed on basis of the joint observations, and the current set of Multistream HMM parameters.
The stream exponents cannot be obtained by maximum likelihood estimation. Instead, discriminative training techniques have to be used, such as the generalized probabilistic descent (GPD) algorithm or maximum mutual information (MMI) training. The simple technique of directly minimizing the word error rate on a held out data set can also be used. Clearly, a number of HMM stream parameter and stream exponent training iterations can be alternated to improve both parameter sets.
Finally, decoding using the Multistream HMM does not introduce additional complications, since 5.12 allows a frame-level likelihood computation, like any typical HMM decoder.
The great advantage of the Multistream HMM is that it is capable of modeling the reliability of the different streams. This becomes especially interesting if this reliability information is not global over the entire stream but is model dependent, it is for example well known that certain sounds are easier to distinguish using visual information while others are more reliably recognized using audio information.
Things become even more interesting when dynamic reliability information, for example based on the amount of noise in a room, is used. Furthermore the Multistream model in its general form allows for different sub-model topologies for its streams and as it allows for independence between streams up to a certain point it is capable of handling duration and offset differences between the streams. The latter is of course not true for the state synchronous Multistream HMM, which has the advantage that it does not requires any special algorithms, contrary to the general modal that requires algorithms not unlike the ones used in the Coupled Hidden Markov Model.

5.5.7 The Product Multistream Hidden Markov model

An extension of the state synchronous Multistream HMM allows the single stream HMMs to be in asynchrony within a model but forces them to be in synchrony at the model boundaries, as is the case in the general Multistream approach, thereby dissolving the disadvantage of state synchrony of the simple Multistream HMM. Single stream log-likelihoods are linearly combined at such boundaries using weights similar to 5.14. For large vocabulary speech recognition, HMMs are typically phones, therefore a reasonable choice for forcing synchrony constitutes the phone boundary.

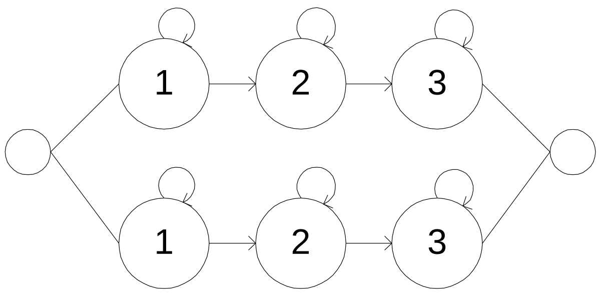


Figure 5.13 - Phone synchronous Multistream HMM

The model can be formulated as a composite or product HMM. Decoding under such a model requires calculating a single best path. The product Multistream HMM consists of composite states that have multimodal emission probabilities of certain single modality HMM states.

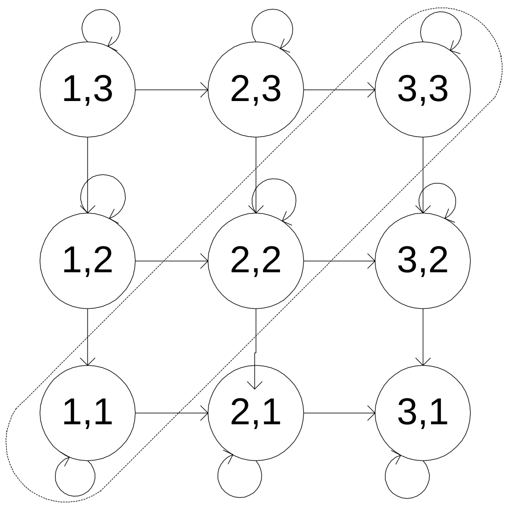


Figure 5.14 – The Product Multistream HMM

These single stream emission probabilities are tied for states along the same row, or column depending on the modality, therefore the original number of mixture weights, mean and variance parameters is kept in the new model. The transition probabilities of the single modality HMMs are now shared by several transition probabilities in the composite model. The product HMM allows restricting the degree of asynchrony between the two streams by excluding certain composite states in the model topology.
As the number of states in the composite HMM is the product of the number of states of all its individual streams, such restrictions can reduce this number considerably and speed up decoding. Figure 5.13 shows a phone synchronous Product Multistream HMM, that allows for limited asynchrony between its separate streams within phone boundaries.

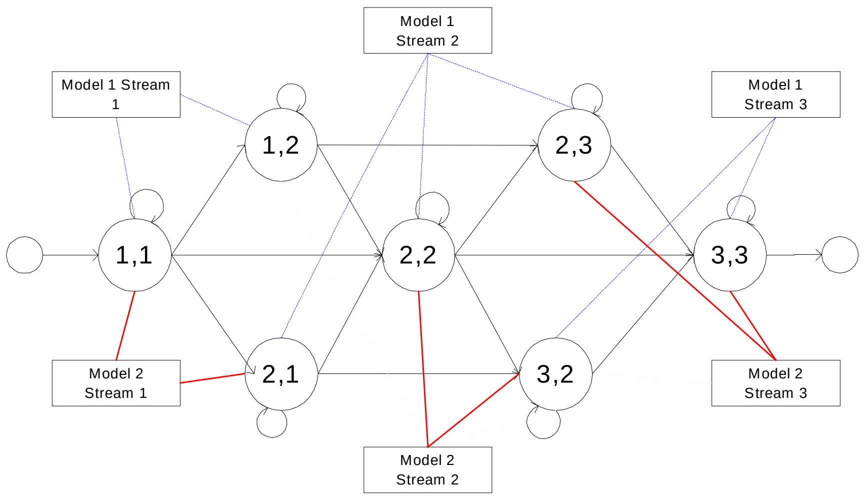


Figure 5.15 – Stream tying in a Product Multistream HMM with limited state asynchrony

In the extreme case, when only the states that lie in its diagonal are kept, the model becomes equivalent to the state synchronous multistream HMM, as is illustrated by the dashed ellipse in figure 5.14. Training of the product HMM can be done, similarly to the multistream HMM, either separately or jointly. This modal has all the properties of the synchronous Multistream HMM and it also allows for asynchrony between signals like the general Multistream approach.

5.6 LATE INTEGRATION

Late integration incorporates separate recognizers for the different modalities and then combines the outputs of these recognizers to get the final result. This approach can easily handle different classifications in different channels as the recognizers for them are separate and the combination is at the output level. In general, the model scores from the different modalities can be combined along with a language model score as independent sources of information. If $P_i(\mathbf{W}\,|\,\mathbf{O}_i)$ denotes the probability provided by the model of modality i for a hypothesis $\mathbf{W}=w_1w_2\ldots w_w$ given the evidence $\mathbf{O}_i$ from the modalities. And $P(\mathbf{W})$ denotes the language model probability, then the linear model that combines all the available information **I** can be defined as:

$$P(\mathbf{W}|\mathbf{I}) = \frac{1}{Z_I} P(\mathbf{W})^{\lambda_{LM}} \prod_i P_i(\mathbf{W}|\mathbf{O}_i)^{\lambda_i} \qquad (5.16)$$

Where Z_I is a normalization factor such that the probabilities for all possible lattice hypotheses $\mathbf{W} \in (V)$ add to one. The weights in this formulation are constant for every model although they could be dynamic, weighting each of the scores with different exponents for different segments of a hypothesis.
Different recognizers may use a different number of distinguishable classes, for example audio recognizers typically use phonemes while video recognizers use visemes. A viseme can be seen as a set of phonemes with similar visual properties. Below the general method is adapted in a number of ways to this specific case, similar methods can be used to integrate other modalities.

1. Phone based classification for both audio and video. Combined likelihood for a given phone hypothesis is computed in the following way:

$$\log P_i(\mathbf{W} \mid \mathbf{O}) = w_a \log P_{a_i}(\mathbf{W} \mid \mathbf{O}) + w_V \log P_{v_i}(\mathbf{W} \mid \mathbf{O}) \qquad i = 1, 2, \ldots N_{phones} \tag{5.17}$$

Where P_{a_i}, P_{v_i} and P_i are the audio, video and combined likelihoods for phone *i* respectively. w_a and w_v are weights given to audio and video hypothesis with $w_a + w_v = 1$.

2. This method uses phone based classes for audio data and viseme based classes for video data. The equation for likelihood computation in this method is as follows:

$$\log P_i(W \mid \mathbf{O}) = w_a \log P_{a_i}(W \mid \mathbf{O}) + w_v M_{ij} \log P_{v_j}(W \mid \mathbf{O}) \tag{5.18}$$

   Where P_{v_j} is the likelihood for viseme *j* given by video vector and M_{ij} is the conditional probability of phone *i* given viseme *j*. M_{ij} can be computed over part of the training set. Again $w_a + w_v = 1$.

3. This method computes the combined likelihood for a phone in two phases. In the first phase only viseme based classes are used for both audio and video. At the end of phase one, the most likely viseme based class is found. In the second phase, phone based models are used for both audio and video to get the most likely phone inside the viseme given by the first phase.

$$\text{phase 1: } P_i(\mathbf{W} \mid \mathbf{O}) = w_{a_1} \log P_{a_i}(\mathbf{W} \mid \mathbf{O}) + w_{v_1} \log P_{v_i}(\mathbf{W} \mid \mathbf{O}) \tag{5.19}$$

$$\text{phase 2: } P_j(\mathbf{W} \mid \mathbf{O}) = w_{a_2} P_{a_j}(\mathbf{W} \mid \mathbf{O}) + w_{v_1} P_{v_j}(\mathbf{W} \mid \mathbf{O}) \tag{5.20}$$

   Where *k* is determined as the most likely viseme in the first phase *j* is a phoneme in the set of phonemes formed by this viseme. Here $w_{a_L} + w_{v_L} = 1$; different weights are used in the first and the second phase. Experiments by IBM [21] have shown that method three outperforms the other two methods and method one performs slightly better that method 2. Of course this was tested on the English language. Our goal was to use similar methods for the Dutch language.

Another simple multiphase approach is to use the video recognizer to generate a lattice containing the most likely hypothesis and then use the audio recognizer to find one single hypothesis within this lattice. Disadvantage of this approach is of course that it is hard to implement in real time.
In general this methods has a number of advantages. It is a relatively simple method, that allows for different classifications for different modalities, also different model topologies and different time scales are possible. The method uses normal Baum-Welch and Viterbi algorithms. Further it models the reliability of the separate streams. On the other hand it does not model any interaction among processes, information on process interaction is completely lost in the recognition phase. All processes have to generate the same hypothesis, thus *N*-best lists are needed. In continuous speech this method is a bit problematic since recombination is done at the end of a spoken utterance.

6 INTEGRATION EXPERIMENTS

To test the multimodal models described in the last chapter a number of experiments was conducted to integrate additional modalities in the speech recognizer of chapter four. This chapter describes these experiments and their results.

6.1 INTRODUCTION

As was described in chapter 5 the use of multiple modalities in speech recognition may result in more robust systems. The problems that have to be solved in the integration of multiple modalities include asynchrony between signals and different levels of reliability. A number of models for multimodal recognition was described in the last chapter.
To test multimodal recognition using these models in practice a number of experiments were conducted. The results of these experiments are described in this chapter. Goal of these experiments was to integrate a second data stream in the speech recognizer described in chapter four. Two techniques described in the last chapter were evaluated. Feature fusion because of its conceptual simplicity and the Multistream HMM which is one of the more powerful integration techniques. The Multistream HMM allows for a certain level of asynchrony between signals and is capable of modeling the reliability of the separate streams. It also has the advantage that it can be realized using the conventional dynamic programming algorithms implemented for example in the Hidden Markov Toolkit that was used during these experiments.
In our experiments, two kinds of additional data streams were used: feature vectors generated by an automatic lip-tracker that was build in the Knowledge Based Systems group at TU Delft and speech feature vectors generated by a Linear Predictive Coding preprocessor. The latter contain approximately the same information as the vectors of the unimodal speech recognizer so we do not expect them to give a substantial rise in performance levels. But on the other hand, exactly because they contain the same information as the original data stream the performance should not decrease. Therefore these vectors were used to validate the models. The integration of lipreading features is an actual example of multimodal speech recognition and shows the difficulties of multimodal integration.
In the next section a short description of the lipreading vectors is given. Next a description of the data set used in these experiments is given followed by the description of the base-line recognition system that was used to compare the multi-modal systems to. Subsequently, the experiments will be described and finally, conclusions will be drawn.

6.2 LIPREADING

The visual part of the speech signal is processed so that it generates two distinct sets of features: geometric and the aerial ones. The geometric feature extraction starts from filtering the image using the *lip-selective* filter. The filter must map the given pixel color to the intensity value from [0,1] interval in such a way that it highlights only the lips in the image. Such filters are possible thanks to the fact that lips have usually more reddish color than the rest of the face. The actual form of the filter is not crucial for the way the data is processed further. For all our experiments we used the filter that incorporates a simple Artificial Neural Network (for details see [33]).

As soon as the image filtering is done, the image is transformed into polar coordinates around the center of the mouth. This center point $\langle X_{center}, Y_{center} \rangle$ can be found by computing the center of gravity of the distribution obtained from filtering the image. The resulting intensity function

$J(\alpha, r)$ is processed further. There are two interesting properties of this function; its conditional mean and variance for specific angle:

$$M(\alpha) = \frac{\sum_r rJ(\alpha, r)}{\sum_r J(\alpha, r)} \tag{6.1}$$

$$\sigma^2(\alpha) = \frac{\sum_r (r - M(\alpha))^2 J(\alpha, r)}{\sum_r J(\alpha, r)dr} \tag{6.2}$$

Those two values relate directly to the thickness of the lips and their distance from the center of the mouth in a given direction. They describe therefore the shape of the lips in a given video frame. Both of those values can be easily estimated from the filtered image.
The shape of the lips is not the only determinant of the spoken utterance. There are some other important factors such as position of the tongue, teeth etc. Some of them can be directly observed in the video sequence, the others not. It is essential in case of lip-reading to extract from the visual channel as much information about the utterance being spoken as possible.

It would probably be possible to track the actual positions of the teeth and tongue to some limited extent. Such a task would be however too complex and therefore infeasible for a lip-reading application. There are however some easily tractable occurrences that can be measured in the image and which relate to the positions and movements of the crucial parts of the mouth. The teeth for example are much brighter than the rest of the face and can therefore be located using a simple filtering of the image intensity. We use here a filter with the steep-wise linear shape with a specified threshold for pixel brightness. It results in the image with only the teeth highlighted.

The visibility and the position of the tongue cannot be assessed as easily as in case of the teeth, especially that the color of tongue is pretty much indistinguishable from the color of the lips. We can however easily assess the amount of the mouth cavity that is not obscured by the tongue. While teeth are distinctly bright, the whole area of the mouth behind the tongue is usually darker that the rest of the face. We can use therefore a filter analogue to the teeth filter but with the high response for the dark pixels.

In order to use the information presented in the filtered images, we need to extract some quantitative descriptions of them. We chose to use the total area of the highlighted region and the position of its center of gravity relative to the center of the mouth as the main features. In total this amounts for a 6 dimensional feature vector. The earlier experiments prove that using those aerial features in addition to the geometric description of the lips results in better recognition results [34].

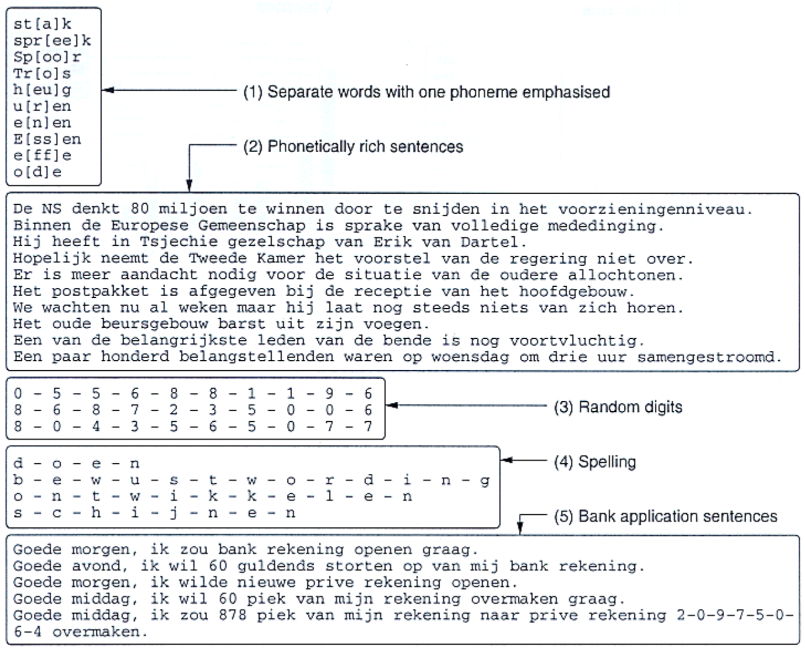
6.3 DATA COLLECTION

To conduct these experiments training and testing data was needed, in particular to experiment with the integration of lip-reading and speech recognition audio-visual data sets were needed. As such a data set was not available we gathered our own data set.

To collect data a number of respondents were asked to read prompts showed on the screen of a laptop in front of a digital video camera. The respondent was seated in front of the laptop computer on which the prompts were displayed to her/him. The prompts were sequentially read from the prompt set and displayed using a simple *PromptShow* application. It allows for choosing the appropriate prompt set file, selecting the section from which it will start etc. The progress of the prompts was controlled by the experimentator, so that the subject's task was only to read the contents of the prompts. We did the recordings using a SONY TRV20E digital camcorder on standard DV tapes equipped with Cassette Memory chips. In this way we could digitally store the code of the recording session for further reference. In order to record an audio signal with a satisfactory quality, we had to use an external microphone, which was hung on the respondents' neck. We used a standard low-cost computer microphone because of its availability, light construction and satisfactory quality.

The camera was placed on a tripod behind the laptop. During each recording, we used the camera's LCD screen to monitor the position of the subject's mouth in the field of view. It proved that the setup was comfortable enough so that most of the participants didn'*t* move substantially during the recordings. The camera's direction was adjusted usually only in the beginning of each new section.
Each of the recorded sessions was edited using a video editing software and cut into smaller sequences. The video sequences were then converted from a standard DV format to MPEG1 stream. Moreover, from all of the scenes audio data was extracted and saved externally. Further, the proper transcriptions of the utterances were added.
The audio was recorded at using a sampling of 44 kHz with 16 bit resolution. For use in these experiments the audio files were converted to 8 bit A-law format, the format used by the speech recognizer described in chapter 4. The video was sampled at 25 Hz.
The set of prompts that is used in our recordings is derived from the prompts recorded for the Polyphone database and from the telebanking application grammar showed in chapter 2.2.
Our prompts collection is divided in 24 sections, each of them with the same structure described later. Recording all of the 24 sections with each of the participants would not be really feasible as it would require almost a two-hour session. All of the subjects agreed that one hour of recordings is already a hard experience. We constrained ourselves to one-hour sessions, which resulted in recording between 10 and 14 sections of the prompt set. Because of the organizational issues such as introducing the subject, resetting the setup etc. during a single hour of recordings we gathered between 25 and 45 minutes of actual material.
Each section of the prompt set contains a fixed number of different utterances. The example section can be seen in figure 6.1. The utterances in a section are:

- 1 sentence containing 10 separate small words.
- 10 phonetically rich sentences randomly chosen from the phonetically rich sentences from Polyphone.
- 3 ten-digit sequences. These digits were randomly generated and have uniform distribution in the whole prompt set.
- 4 spelled words.
- 5 utterances from the telebanking application.


(1) Separate words with one phoneme emphasised
(2) Phonetically rich sentences
(3) Random digits
(4) Spelling
(5) Bank application sentences

```
st[a]k
spr[ee]k
Sp[oo]r
Tr[o]s
h[eu]g
u[r]en
e[n]en
E[ss]en
e[ff]e
o[d]e
```

```
De NS denkt 80 miljoen te winnen door te snijden in het voorzieningenniveau.
Binnen de Europese Gemeenschap is sprake van volledige mededinging.
Hij heeft in Tsjechie gezelschap van Erik van Dartel.
Hopelijk neemt de Tweede Kamer het voorstel van de regering niet over.
Er is meer aandacht nodig voor de situatie van de oudere allochtonen.
Het postpakket is afgegeven bij de receptie van het hoofdgebouw.
We wachten nu al weken maar hij laat nog steeds niets van zich horen.
Het oude beursgebouw barst uit zijn voegen.
Een van de belangrijkste leden van de bende is nog voortvluchtig.
Een paar honderd belangstellenden waren op woensdag om drie uur samengestroomd.
```

```
0 - 5 - 5 - 6 - 8 - 8 - 1 - 1 - 9 - 6
8 - 6 - 8 - 7 - 2 - 3 - 5 - 0 - 0 - 6
8 - 0 - 4 - 3 - 5 - 6 - 5 - 0 - 7 - 7
```

```
d - o - e - n
b - e - w - u - s - t - w - o - r - d - i - n - g
o - n - t - w - i - k - k - e - l - e - n
s - c - h - i - j - n - e - n
```

```
Goede morgen, ik zou bank rekening openen graag.
Goede avond, ik wil 60 guldends storten op van mij bank rekening.
Goede morgen, ik wilde nieuwe prive rekening openen.
Goede middag, ik wil 60 piek van mijn rekening overmaken graag.
Goede middag, ik zou 878 piek van mijn rekening naar prive rekening 2-0-9-7-5-0-
6-4 overmaken.
```

Figure 6.1 – example section

At the current stage, we have recorded 8 sessions with 8 different subjects. This gives in total over 4 hours of constant recordings. The recorded subjects are all native Dutch speakers. There are 7 male subjects and one female. Data from five of these subjects was transcribed and used in these experiments. From each subject 4 or 5 sessions were used. This data set was split in a training set of approximately 500 utterances from all speakers and a test set containing 30 utterances from all speakers.

6.4 THE BASE-LINE SYSTEM

As starting point in these experiments the monophone system from section 4.4 was used. This particular systems was chosen because it does not contain any complicating structures like multiple mixtures or context dependent models, making model integration easier. Further the modest results achieved by this model leave room for gains in performance. On the other hand its simple structure also limits the performance that can be reached by this model, therefore great leaps in performance are not to be expected, it is the relative performance that is of interest here. To provide a baseline system to which the multimodal systems could be compared the monophone system was re-estimated twice using the training set. This resulted in an adapted monophone recognizer showing the following results on the test set:

Table 6.1

Percentage of words correct:	45.14%
Word accuracy percentage	2.70%
Percentage of sentences correct	0.00%

To perform feature fusion a program called *cvf*, which stands for *concatenate vector files*, was written. This program concatenates the vectors in a file from one set of HTK vector files to the corresponding vectors in a file from another set. The concatenated vectors are saved in a new file. The sample rate of this new file can be specified at the command line of the program. If the input files' sample rates differ or they differ from the output sample rate, then interframe

interpolation or copying of the current vector is used to obtain the right frame rate. If the input files have different file lengths the last vector of the shortest file is repeated until all vector from the other file are processed.
The program can also add an offset to one of the input files. The first vector of this file is than repeated for the duration of the offset, as a result all other vectors are shifted in time.

6.5 EXPERIMENT 1:

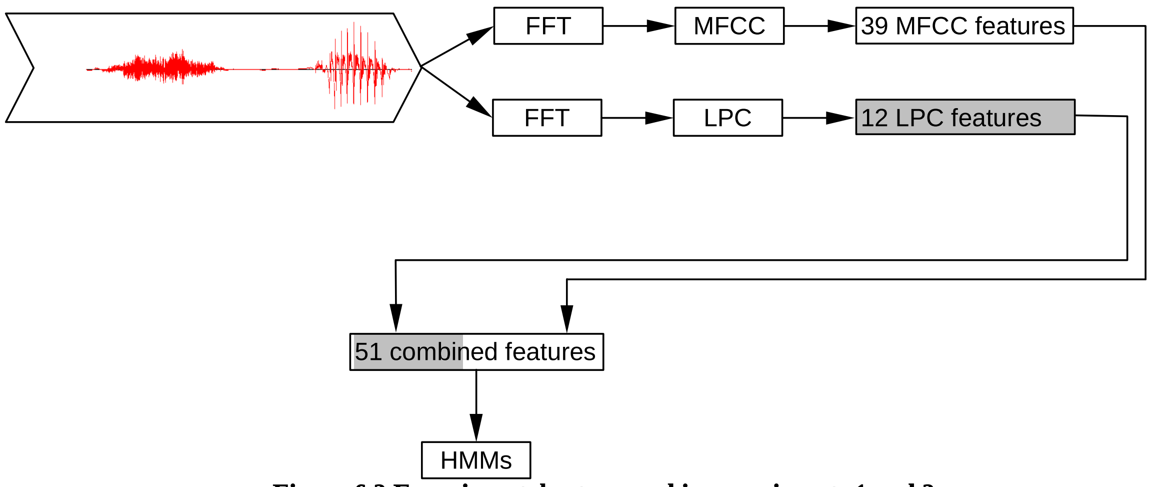


Figure 6.2 Experimental setup used in experiments 1 and 2

6.5.1 Design

To validate a fusion technique, data streams should be used that are known to have features that adequately reflect the data, to make sure that the results are not influenced by problems in the preprocessing phase. Linear Predictive Coding produces features that are known to represent the speech signal quite well. LPC is based on different aspects of human speech production than Mel scale filterbank analysis and therefore may provide some complementary information. And even if it does not, it should at least not have a negative influence on the overall recognition results. Therefore these features were used to validate our models before tests involving lipreading data were done.
In this experiment the 12-dimensional LPC feature vectors were extracted from the utterances in the training set at a frame rate of 10 ms with a overlapping window of 25 ms. A simple recognizer was build using these feature vectors to see whether these relatively small vectors contain adequate information. Initial three state left-to-right LPC based model were created by setting the means and variances of all models to the global mean and variance calculated over these vectors.
Subsequently a multimodal system was build using feature fusion. To do so, the LPC vectors were coupled to the MFCC audio vectors using *cvf*. As the two vector types were sampled at the same rate using the same window size there was no need for interpolation of frames. The Hidden Markov models from the unadapted audio-only system were altered to incorporate 51 instead of 39 dimensional distribution functions and the means and variances of all models were extended by concatenating the global means and variances calculated over all LPC vectors to them.

6.5.2 Results

The LPC-only models were re-estimated three times after which Viterbi alignment and two more re-estimation cycles were performed to obtain a simple monophone LPC based recognizer for the data set. Viterbi recognition of the test set on these models gave the following results:

Table 6.2

Percentage of words correct:	30.15%
Word accuracy percentage	-2.43%
Percentage of sentences correct	0.00%

The concatenated models the last section were re-estimated by performing three cycles of embedded re-estimation using the training set, the following results were obtained:

Table 6.3

Percentage of words correct:	47.84%
Word accuracy percentage	3.51%
Percentage of sentences correct	0.00%

Further training cycles did not significantly change these values.

6.5.3 Discussion

The experiment with the LPC only vectors showed that these vectors do contain some information representing the speech signal, so they can be used to validate fusion techniques. Coupling of LPC and MFCC features does indeed improve the performance of the system. The feature fusion technique works for these two data streams. This was of course to be expected as they were sampled at the same frame rate and represent the same signal, so they actually are synchronous.

6.6 EXPERIMENT 2

6.6.1 Design

Incorporating two preprocessors that sample the same signal at the same rate may not be very useful in practice. But things get interesting if the second stream is sampled at a different frame rate. This way contextual information can be incorporated in a vector and so the vectors may give a better representation of the continuous speech signal. This may be especially useful in case of a lossy channel or a noisy environment. The vectors in the second data stream can compensate for missing or distorted vectors in neighboring positions.
In this experiment LPC vectors calculated using a larger window were computed and fused with the normal MFCC vectors to see whether vectors calculated over a longer time span do indeed exhibit the desired behavior.
LPC cepstral coefficients, which are a bit more stable than normal LPC coefficients, were extracted from the audio files using a sampling rate of 10 ms, a 50 ms wide overlapping window. A second set of LPC cepstral coefficient vectors was calculated using a sampling rate of 10 ms and a 75 ms wide window. For both vector sets a LPC only system was build and trained in the same as in the previous experiment. Fused systems were also build in both cases again by

concatenating the global means and variances calculated over a LPC vector set to the means and variances of all states from each model in the unadapted audio system.

6.6.2 Results

The LPC-only system with 50 ms windows was re-estimated 3 times. Viterbi recognition produced the following results:

Table 6.4

Percentage of words correct:	28.00%
word accuracy percentage	1.08%
Percentage of sentences correct	0.00%

The fused system using these vectors was re-estimated three times.

Table 6.5

Percentage of words correct:	46.22%
word accuracy percentage	2.70%
Percentage of sentences correct	0.00%

The LPC-only system with 75 ms windows was trained by 3 cycles of re-estimation, giving the following results:

Table 6.6

Percentage of words correct:	30.27%
word accuracy percentage	13.24%
Percentage of sentences correct	0.00%

Its fused version was also re-estimated three times, giving:

Table 6.7

Percentage of words correct:	46.49%
word accuracy percentage	3.55%
Percentage of sentences correct	0.00%

6.6.3 Discussion

So, incorporating features sampled over longer time spans does indeed have a positive result on the performance. It is remarkable to see that the LPC feature vectors that are calculated over a window of 75 ms perform better than the ones calculated using a 50 ms window. The inclusion of more context information can apparently improve a recognizer.

6.7 Experiment 3

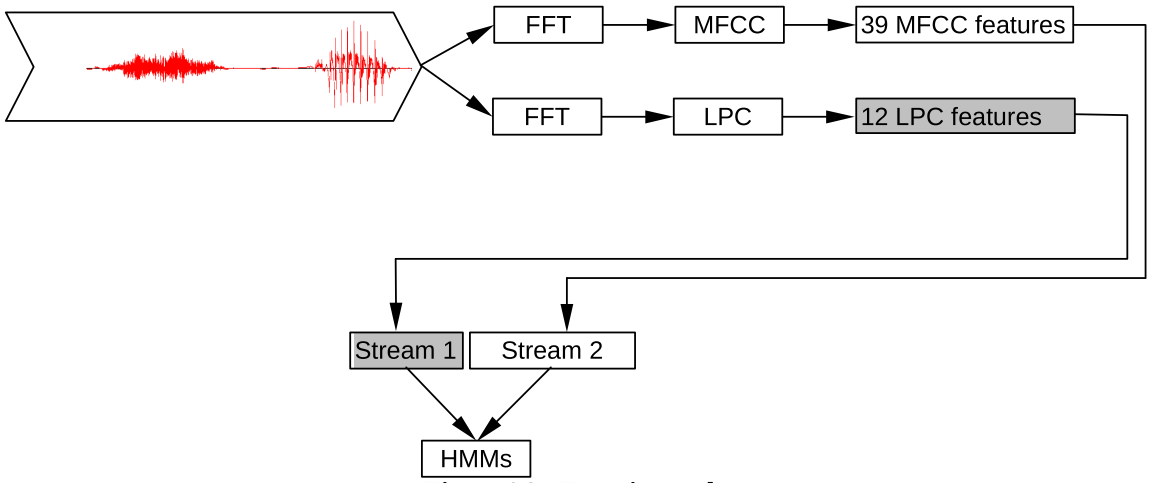


Figure 6.3 - Experimental setup

6.7.1 Design

Of all fusion techniques described in the last chapter the Multistream approach is one of the most attractive approaches. Depending on the model topology used it can cope with a certain amount of asynchrony between signals and it can model the reliability of the separate streams.
In this experiment the state synchronous Multistream HMM was validated. Once again the LPC cepstral coeffient vectors with 75 ms window were used as second data stream. Multistream models were created by copying the means and variances of the models of the unadapted audio-only system to the means and variances of the first stream of the corresponding Multistream model.
The means and variances of each state in the second stream of every model were set to the global mean and variance calculated over the LPC feature vectors.

6.7.2 Results

The models were trained 3 times using embedded re-estimation. Both streams had the same global weights (1.0, 1.0).

Table 6.8

Percentage of words correct:	46.44%
word accuracy percentage	3.51%
Percentage of sentences correct	0.00%

A second system was trained 3 times this time using a global weight of 1.2 for the first stream and a global weight 0.8 of for the second stream:

Table 6.9

Percentage of words correct:	48.58%
word accuracy percentage	3.61%
Percentage of sentences correct	0.00%

A third system was trained also 3 times this time using a global weight of 0.8 for the first stream and a global weight of 1.2 for the second stream:

Table 6.10

Percentage of words correct:	46.23%
word accuracy percentage	3.30%
Percentage of sentences correct	0.00%

6.7.3 Discussion

The first system that does not weight its streams has practically the same results as the 75 ms window system from experiment 2, not only the percentages are similar, but also the same mistakes are made during recognition. This shows that the two datastreams are indeed synchronous and their combination can be adequately modeled by the feature fusion technique. No modeling power is added by the Multistream technique in this case.
The second and third system show that the original audio stream is more reliable that the LPC stream. This is not a surprise as the original system uses 39 dimensional feature vectors that are well trained on a huge training set and subsequently adapted to the training set used here. The LPC vectors on the other hand are only 12-dimensional and do not include any delta or acceleration coefficients. They are trained only a few number of times on a limited data set. This suffices to capture the most important aspects of the underlying signal but not to give a reliable model of this data.

6.8 EXPERIMENT 4

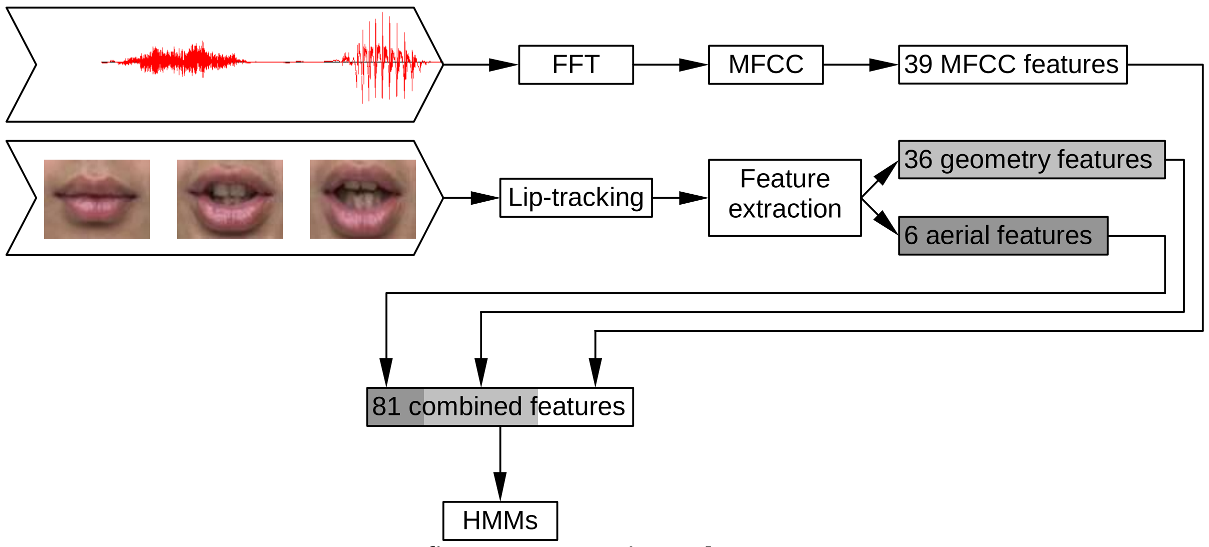


figure 6.4 – Experimental setup

6.8.1 Design

The previous experiments showed that the feature fusion and Multistream models can be used for multimodal integration. But the question remains how well they perform when a data stream composed of two dependent but different data streams is used. In this and the following experiments lipreading features were integrated in the speech recognizer. As was already explained in the introduction of chapter one, lipreading provides some complementary information to the audio recognition process and is therefore a good candidate for improving speech recognition by multimodal fushion.

In this experiment 42 dimensional lipreading feature vectors were concatenated to the 39 dimensional MFCC speech vectors. The lipreading vectors were sampled every 40 ms, so the frame rate of the video vectors is 4 times lower than the audio vector frame rate. The vectors were concatenated using *cvf*, interframe interpolation was used to obtain output vectors at a framerate of 10 ms per frame.
To create an initial set of HMMs first the gobal mean and variance were computed over the set of interpolated video vectors. These mean and variance vectors were then added to the means respectively the variances of ever state off all the models in the set of audio models and the definitions of the models were altered to incorporate these 81 dimensional feature vectors.

6.8.2 Results

The initial models were trained by two cycles of embedded training with the tool HERest using the concatenated vectors. Then recognition was performed on the test set using these multimodal models. The following results were obtained:

Table 6.11

Percentage of words correct:	6.22%
word accuracy percentage	-38.38%
Percentage of sentences correct	0.00%

Additional training cycles reduced the recognition percentages even more.

6.8.3 Discussion

These results show that the system was not capable of finding any pattern in the training data, the models randomly produce vectors. This is very likely due to the large dimensionality of the vectors and the relatively small training set.

6.9 EXPERIMENT 5

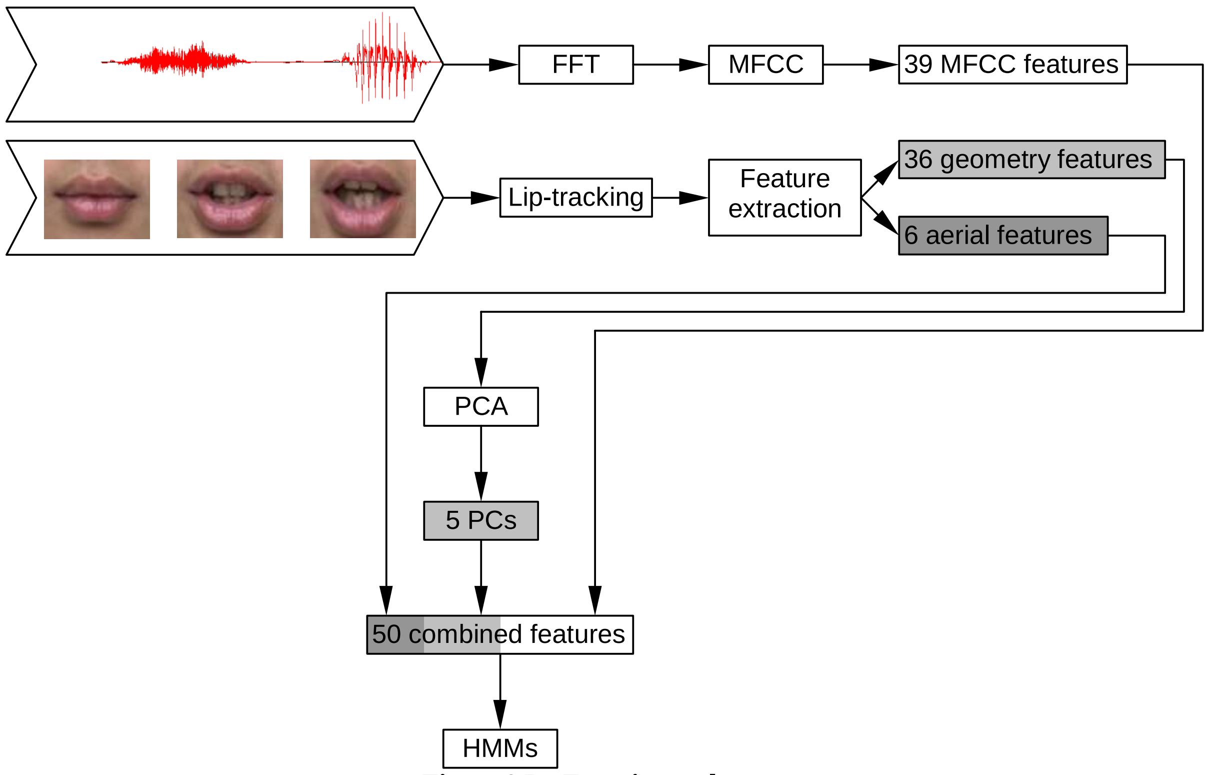


Figure 6.5 – Experimental setup

6.9.1 Design

Based on the conclusions of the first experiment the dimensionality of the lipreading vectors was reduced considerably by means of Principal Component Analysis (PCA). This is a common signal processing technique that can be used for dimension reduction and visualization of a highly dimensional data. Although performing PCA on large and complex datasets is no trivial task due to the fact that it involves inverting large matrices, its basic idea is very simple. The PCA performs in fact a projection on the principal components of the data so that the most data variation is concentrated in the first couple of the components. This is done by at first finding the direction in which data variance is the highest (first principal component). The second principal component (PC) is the direction that maximizes the data variation being also perpendicular to the first one. The third and all following components must be perpendicular to all of the previous ones. In this way, the PCs construct a coordinate system in the data-space that maximizes the variation of the data in the first couple of dimensions. Using only the first few of the components allows for reduction of data dimensionality without loosing too much of the information contained in the data. Moreover, in most cases, only the first few PCs relate to the real information contained in the data, the rest of them contains only noise-related variations. In this way PCA allows also for noise reduction.

In our case we decided to perform the PCA on the geometrical part of the feature vector. Firstly, the data vectors G(*i*) have to be moved to the origin of the coordinate system:

$$G'(i) = G(i) - \sum_{k=1}^{N} \frac{G(k)}{N} \qquad i = 1..N \tag{6.3}$$

Then the data vectors *G'(i)* are collected in a large matrix **G** in such a way that each row of this matrix contains just one vector *G'(i)*. The PCA is further performed by doing a singular value decomposition of the **G**:

$$\mathbf{U}\,\mathbf{S}\,\mathbf{V}' = \mathbf{G} \tag{6.4}$$

Where the **U** and **V** matrices are unitary and **S** is a diagonal matrix with the nonnegative diagonal elements sorted in decreasing order. As the **U S** part of the decomposition contains the columns with the decreasing variance, the rotation **V** (inv(**V**) = **V'**) is the transformation we were looking for. For each vector *G*'(*i*) we can now obtain its representation in the coordinates described by PCs:

$$g(i) = G'(i)\mathbf{V} \tag{6.5}$$

A quick look at the variances in the transformed data shows that the variation of the data beyond 5th PC is negligible (see Figure 6.6). We used therefore only the first 5 PCs as a feature vector. Using even smaller number of PCs we performed some visualization of the data (see Figure 6.7 and 6.8)

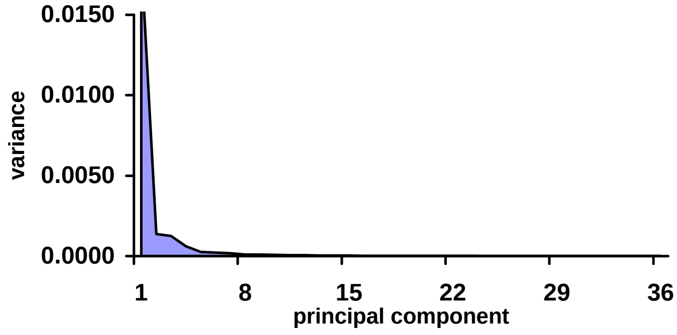


Figure 6.6 - The variance in the data after projection into the principal components.

The five PCA features together with the 6 aerial features resulted in vectors with 11 features each. These vectors were concatenated to the audio vectors in the same way as in the first experiment. So now the concatenated vectors and the state distribution functions were of dimension 50.

6.9.2 Results

Re-estimation was performed with these models and these concatenated vectors twice, giving the following results.

Table 6.12

Percentage of words correct:	42.70%
word accuracy percentage	0.81%
Percentage of sentences correct	0.00%

6.9.3 Discussion

This result approaches the results of the audio only system, showing that the problems encountered in the first experiment were indeed caused by the high dimensionality of feature

space. However the performance of this multimodal system is still not better than the performance of the audio only system.
To determine how these results came about the data was closely inspected. In order to facilitate the visualization of the high dimensional data-vectors we used the first two principal components and plotted them on the 2D plane. The full dataset can be seen as the background cloud of points in Figures 6.7 and 6.8.

In the first attempt to visualize the dependencies between the feature vectors and the uttered speech, we used the speech recognizer to obtain the segmentation of the data using Viterbi alignment. After obtaining the segmented speech, we could map any video-frame to the appropriate phoneme being spoken at that time. Having this, we could plot the visual feature vectors for each specific phoneme separately. To our big surprise, there seemed to be absolutely no correlation between the phoneme and the values of the first two PCs. When plotting the feature vectors for a specific phoneme, the resulting points would be scattered all over the full dataset region. This clearly suggests that there is no trivial equivalency between the audio and video frames.

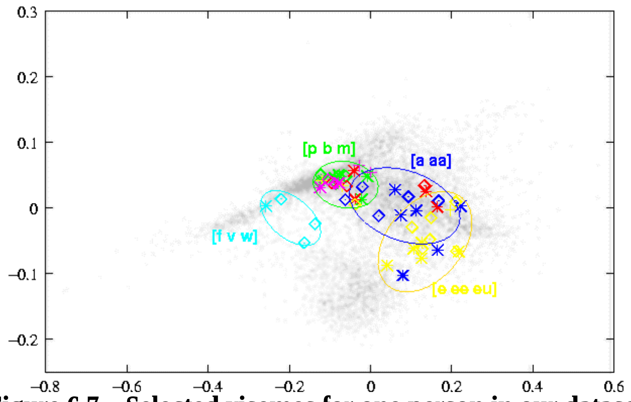


Figure 6.7 – Selected visemes for one person in our dataset

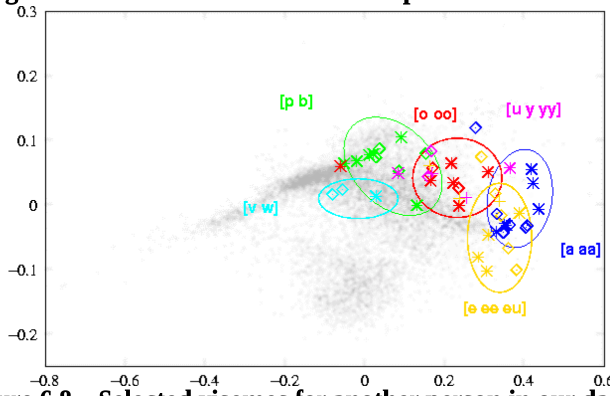


Figure 6.8 – Selected visemes for another person in our dataset

In order to make the investigation into the visual features independent to the audio signal, we performed some manual labeling of the dataset. In a selected number of recorded video-sequences we labeled the phonemes that had a clear point of occurrence in the video. For example, most of the vowels could be labeled by finding the video frame where the mouth is opened to the most extent for given phoneme. Some of the consonants, such as {p b m v w f}, are also easy to label because their point of articulation is clearly visible. The resulting vectors were once again plotted using the first two PCs. As it can be seen in Figures 6.6 and 6.7, the different phonemes gather in clusters corresponding to their visemes. Of course, the structure is not absolutely clean and some person-dependency and noise remains, but clearly there is a correlation between the uttered sound and the extracted representation of the visual data.

The reason for the completely different results when using automatically segmented labels and data labeled manually is that there is an inherent lack of synchronization between the audio and video signal. We decided to investigate in which part of the audio segment the viseme occurs. In order to do so, we used the automatically segmented phone ranges for the phonemes that were manually labeled. The phone ranges were normalized and the histogram of the positioning of manual label within the range was made (see Figure 6.9).

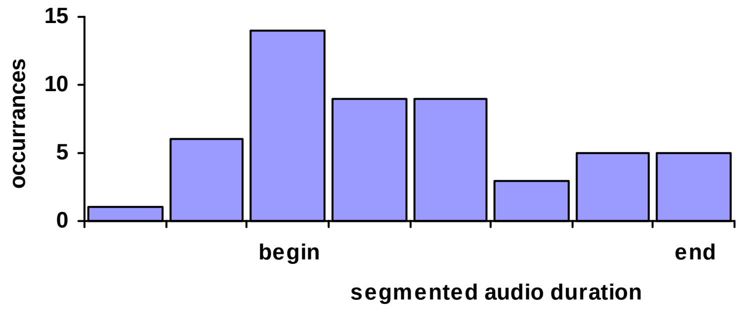


Figure 6.9 - The visibly distinguishable phoneme occurrence with respect to the audibly segmented phoneme.

As can be seen, although there is a clear preference for the viseme to appear at the beginning of the utterance, there is a substantial amount of the visemes that occur in the whole phoneme range and a small number of visemes preceding the utterance. As it was mentioned earlier, the HMMs could handle to some extent the variability of the viseme placement within the modeled time range. It is however absolutely impossible for a model to incorporate the observations that are not presented to it at all. The problem can be solved by either synchronizing the separate models for visual and auditory observations at some higher level speech blocks (such as words or sentences) or by a simple time-shifting of the datastreams so that the respective occurrences in both streams at least overlap each other.

6.10 EXPERIMENT 6

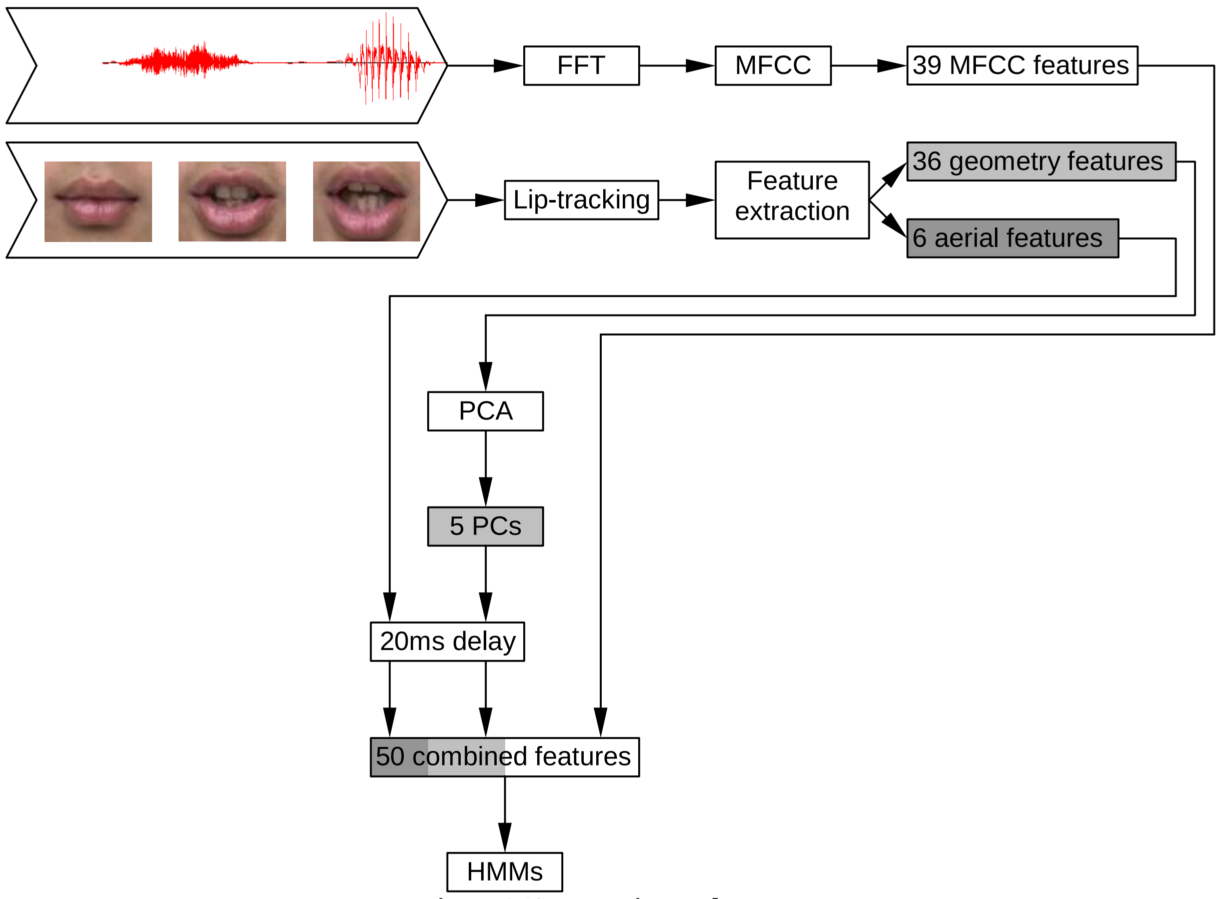


Figure 6.10 – experimental setup

6.10.1 Design

The conclusions of the last section suggest that if the video vectors are offset by 20 ms than most of the video vectors should now be approximately in the same time frame as the corresponding audio vector, so recognition results should improve compared to the earlier experiments. To test this hypothesis an experiment similar to the first two experiments was performed, but this time the video vectors were offset by 20 ms.

6.10.2 Results

After three re-estimation cycles the following results were obtained:

Table 6.13

Percentage of words correct:	45.95%
word accuracy percentage	3.24%
Percentage of sentences correct	0.00%

6.10.3 Discussion

The performance of this system was similar to that of the audio-only system and better than the performance of the other audio-visual systems. The fact that offsetting the signal gave better recognition shows that the audio and video signals are indeed not synchronized. This implies that the feature fusion technique without a synchronization module is not the best fusion technique for combining lipreading and speech features, since this approach expects the signals to be synchronized. Using a fixed offset is of course not a very adequate solution as the interaction between the signals is a dynamic process.

6.11 Experiment 7

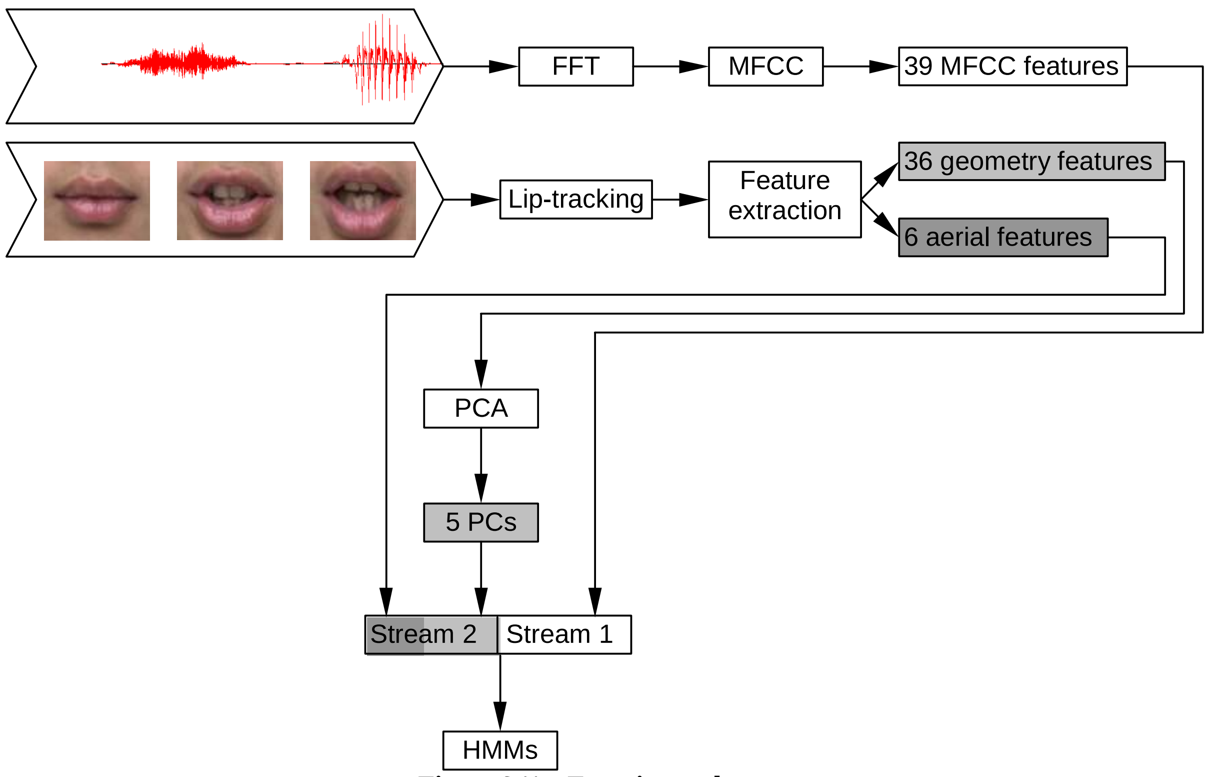


Figure 6.11 – Experimental setup

6.11.1 Design

The results of the last test and the analysis performed on the data showed that there is a certain level of asynchrony between audio and visual vectors. Therefore an experiment with the multistream HMM that allows for a certain kind of asynchrony was conducted. This model is also capable of modeling reliability of the modalities. In the case of audio-visual speech recognition the audio data contains much more information than the video stream therefore the reliability of the audio stream should be set higher than the reliability of the video stream.
In this experiment a state synchronous Multistream HMM was used, the means and variances in the first stream of the model were copied from the audio-only system. The means and variances of each state in the second stream of every model were set to the global mean and variance calculated over the video feature vectors. The same 11-dimensional video vectors as in the previous experiment were used.

6.11.2 Results

Two training iterations with relative weight 1.2 for the audio stream and 0.8 for the video stream were performed, this resulted in:

Table 6.12

Percentage of words correct:	47.30%
word accuracy percentage	6.76%
Percentage of sentences correct	0.00%

6.11.3 Discussion

The limited asynchrony in this model resulted in a better performance than in the audio only system.
Further inspections of the lipreading data revealed that the lipreading features used sofar were partially person dependent as can be seen in figure 6.7 and figure 6.8, were for example the viseme [a, aa] lies in to complete different regions in the projective plain. Although HMMs are capable of capturing such effects our dataset proved to be too small to build a robust lipreading recognizer that could handle the variations between lip movements of different persons and the variations in these signals introduced by visual coarticulation. As a consequence it was also impossible to build a robust audio-visual recognizer or to try more advanced couplings like the Coupled Hidden Markov model which requires a large data set that contains examples of all possible variations in audio and visual data and of the interaction between these modalities.

6.12 OVERVIEW OF RESULTS

Table 6.13 shows an overview of the results of all experiments described in this chapter.

Table 6.13

Experiment	Percentage of words correct.	Word accuracy percentage
Base-line system	45.14%	2.70%
LPC	47.84%	3.51%
LPC 50ms	46.22%	2.70%
LPC 75ms	46.49%	3.55%
LPC 75 ms Multistream	48.58%	3.61%
42 visual features	6.22%	-38.38%
11 visual PCA features	42.70%	0.81%
11 visual PCA features 20 ms offset	45.95%	3.24%
11 visual PCA features Multistream	47.30%	6.76%

7 CONCLUSIONS AND RECOMMENDATIONS

Part I of this report focuses on the general theory of automatic speech recognition and the development of an automatic large vocabulary speech recognizer in particular. The theory of speech recognition, presented in chapter 2, is a theory of mathematical nature; most textbooks on this subject concentrate mainly on the Hidden Markov model and its dynamic programming algorithms. More practical aspects like the advantages or disadvantages of certain model topologies or the steps involved in data preparation and training are seldom covered. Many questions concerning these issues came up during the development of the speech recognizer described in chapter four.

The first issue that has to be addressed is data preparation. This part of the process took much more time than initially expected. Data had to be selected that met the criteria posed for the recognizer. As huge data collections are needed for training of speaker independent speech recognizers a tool had to be created that could perform automatic data selection.

One of the decisions that has to be made during data preparation is the topology of the acoustic models. In experiments with the English TIMIT speech corpus, preceding this project, we found that three-state left-to-right phoneme models had a similar performance as five state left-to-right models. Since the three-state model needs less parameters, it was chosen for this project. However it would be interesting to see how the five-state model performs when it is used to model advanced triphone models for the Polyphone data set. During training it was found that it is important to create good initial models, HMM are, due to their Markov property, very sensitive to initial values. Errors in the initial model set a limit to the performance of the final system. In case of the Polyphone data no segmented transcriptions were available, so initial isolated training of the phoneme models could not be done. This problem was solved by using global means and variances for all models based on the assumption that this are neutral values where the parameter values from all models are distributed around and which do not give an error bias. As segmented transcriptions can be created using Viterbi alignment, the recognizer build in this project can be used to segment the data in the Polyphone database. Training of a model set on this segmented data may result in a recognizer that performs better while using for example fewer mixtures. With respect to the initial models it is important that the silence models are not used in the first few training phases, they may absorb data belonging to other phones.

The monophone model built here booked limited results, we found that reasonable levels of performance could only be reached by using multiple mixture distributions and context dependent models. In case of segmented data fewer mixtures could have been used. The system really started to improve when triphones were introduced. They provided the system with the capability of modeling the variations introduced by coarticulation. Five different triphone clusterings were built and evaluated to find a good set of generalized triphones. One of these systems was chosen for further development. More research on this topic could be done to find an optimal set of triphones for this system. One of the problems that occurred when triphones were introduced was the problem of unseen triphones: there are many triphones in the Dutch language that are not in the training set used. This problem was solved by extending the set of triphone models with a set of multiple mixture triphone models that were used to represent all corresponding unseen triphones. It would be interesting to compare the results of this data-driven approach to the results of a tree-based clustered triphone system that uses linguistic knowledge to assign unseen triphones to the right cluster.

The language models used in this system are simple backed-off bigrams, they have limited modeling power and put only weak constraints on the syntax of the utterances. This is confirmed

by the fact that the speech recognizer relies mainly on its acoustic models as was found during fine tuning of the system. State of the art systems use at least trigram language models, the performance of the recognizer might be improved by using this or even better language models, this is especially important to ensure good performance levels in case of larger vocabularies.
The overall results of the final system are quite good, more than ninety percent of all words are recognized correctly. The errors that occur are usually small and typically involve wrong conjugations of a verb or hesitations by the speaker. This kind of errors could be reduced by a more powerful language model or a preprocessing module that check the syntax of a sentence. The system can cope with mouth noises like smacking or loud breathing and this system is speaker independent. It has been tested with vocabularies of more that 5000 words, the performance decreased only slightly in this case. The system can be adapted to other environments and performance can be further improved, especially on sentence recognition, by making it person dependent as was shown in 4.5.5. Because of its open source the system can be easily incorporated as a speech engine in applications or it can be used to build more advanced recognizers like multimodal recognizers.

The emphasis in part II of the report is on the development of models for multimodal integration. The models presented have different properties, it depends on the processes that are to be integrated which method is most suitable. For dependent processes that evolve synchronously the feature fusion technique offers a simple way to integrate the signals. On the other extreme lie processes that evolve completely independent and possibly asynchronous, but all influence the outcome of the recognizer. This type of coupling can be realized using the Factorial Hidden Markov Model or late integration. In the case of multiple modalities that give information on what is spoken by a person, the signals will typically depend on each other and a limited amount of asynchrony can occur. For example, there is a clear dependence relationship between the shape of the mouth and the sound uttered. These processes and their relations can be modeled using the Coupled Hidden Markov model or the Multistream Hidden Markov model. The Multistream model requires more knowledge about the processes on the side of the model designer because model and sub-model units have to be chosen, but the advantage of the Multistream model is that if its topology is not too complicated, simple variations of the dynamic programming algorithms for speech recognition can be used and furthermore it models the reliability of the datastreams.
The experiments in this chapter show that the feature fusion technique and the Multistream HMM can indeed be used to incorporate multiple modalities in a speech recognizer.
We tried to incorporate context information in the feature vectors by using LPC vectors calculated over relatively long time spans. This showed some improvement in recognition performance, thereby confirming the correctness of our models, but the real power of this approach is likely to show in noisy environments, this would be an interesting subject for further research.
The integration of lipreading features in the speech process may result in a recognizer that is much more robust to noise and ultimately may be capable of operation in a multispeaker environment. Our experiments showed that the feature fusion technique is not very well suited for integration of audio and visual features because of asynchrony in the speech signal. The Multistream approach seems to be able to cope with this problem and to improve on the performance of audio-only recognition. Once again the improvements may be much larger than shown in the above experiments if noisy audio is used. In a noisy environment performance of the unimodal ASR is less or equal to the performance of the lipreading component. From fusion we might expect an improvement of ASR. On the other hand in a relatively silent environment the audio part of the recognizers is able to capture most of the dynamics of the speech itself,

when both recognition subsystems perform well they will come to the same conclusion most of the time, so fusion will not result in spectacular results in this case.
The systems built in our experiments were only trained a small number of times and used only three states per model with single Gaussian distribution functions. This of course limits the performance of the systems. As was shown in chapter four, to create more advanced models that better model the dynamics within the audio and video signals and the dependencies between them, further training and the introduction of multiple mixtures and context dependent models is necessary. However these techniques require large amounts of training data to prevent overtraining. In our experiments we found that convergence of the models was already reached after a few training iterations, increasing the number of parameters in these models would certainly have resulted in overtrained models. This is no surprise as we used a training set containing only 500 utterances from 5 different persons while the database needed to build an audio-only recognizer already contained over 20000 utterances.
Recording and preparing an audio-visual database is a time and effort-consuming task, creating a useful database with a scale comparable to Polyphone may take years. But it is certainly worth the cost as even the small dataset used in our experiments allows for improvements in recognition results.
Furthermore we found that there are still many problems to be solved in the area of lipreading. The lipreading systems used in our experiments did not compensate (enough) for person dependent features and once again it was the lack of training data that limited the ability of the HMMs to capture the highly dynamic video signal. This problems should be solved first before the problem of integrating the speech recognition and lipreading systems is attacked. When robust lipreading features a large representative data collection are available couplings using more advanced models like the Multistream Product HMM and the Coupled Hidden Markov model could also be applied to integrate audio and video recognizers.

REFERENCES

[1] Rabiner, L.R., Juang, B. H., *Fundamentals of Speech Recognition*, Prentice Hall, Englewood Cliffs, N.J., 1993

[2] Jelinek, F., *Statistical Methods for Speech Recognition* (Language, Speech, and Communication) MIT Press, January 1999

[3] Alphen, P. van, *Hidden Markov Models in Speech Recognition*, Academic Press

[4] Huang, X.D., Ariki, Y., Jack, M.A., Hidden Markov Models for Speech Recognition, Edinburgh University Press, 1990

[5] Furui, S., *Digital Speech Processing, Synthesis and Recognition,* Second Edition Revised and Expanded, Marcel Dekker, February 2001

[6] Jurafsky, D., Martin, J. H., Van der Linden, K., *Speech and Language Processing: An Introduction to Natural Language Processing,* Computational Linguistics and Speech Recognition, Prentice Hall, January 2000

[7] Junqua, J., *Robust Speech Recognition in Embedded Systems and PC Applications*, Kluwer International Series in Engineering and Computer Science, Secs 563, Kluwer Academic Publishers, May 2000

[8] Barksdale, K., Rutter, M., *IBM ViaVoice for the Office Professional*, Speech Recognition Series South-Western Educational Publishing, November 2000

[9] Young, S., Kersaw, D., Odell, J., Ollason, D., Valtchev, V., Woodland, P. C., *The HTK Book (for HTK Version 3.0),* Cambridge University Engineering Department

[10] Woodland, P.C., Odell, J., Young, S.J., *Large Vocabulary Continuous Speech Recognition Using HTK*, in Proc. ICASSP 1994

[11] Woodland, P.C., Leggetter, C.J., Odell, J., Valchev, V., Young, S.J., *The Development of the 1994 HTK large vocabulary speech recognition system,* Cambridge Univerity Engineering Department, in Proc. ICASSP 1995

[12] Woodland, P.C., Odell, J., Young, S.J., *Tree-Based Tying for High Accuracy Acoustic Modelling,* Cambridge Univerity Engineering Department, in Proc. ARPA Human Language Technology Workshop, March 1994

[13] Woodland, P.C., Young, S.J., *The HTK Tied-state Continuous Speech Recognizer,* Cambridge Univerity Engineering Department, Proc. Eurospeech '93

[14] Woodland, P.C., Young, S.J., *Broadcast News Transcription Using,* Cambridge Univerity Engineering Department, in Proc. ICASSP '97

[15] Grocholewski, S., *Acoustic Modeling for Polish*, International Workshop Speech and Computers, SPECOM'2000, St. Petersburg, September 2000

[16] Dupont, S., Luettin J., *Using the Multi-Stream Approach for Continuous Audio Visual Speech Recognition*, IDIAP Research Report 97-14

[17] Dupont, S., Bourlard H., Ris, C., *Robust Speech Recognition Based on Multi-Stream Features,* IDIAP.

[18] Bourlard, H., Hagen A., *Using Multiple Time Scales In The Framework of Multi-Stream Speech Recognition,* Int. Conf. on Spoken Language Processing, Beijing 2000

[19]Luettin, J., Dupont, S., *Continuous Audio-Visual Speech Recognition*, IDIAP, Dalle Molle Institute for Perceptual Artificial Intelligence

[20] Basu, S., Neti, N., Rajput, A., Senior, L., Subramaniam, Verma, A., *Audio-Visual Large Vocabulary Continuous Speech Recognition In The Broadcast Domain*, IBM T.J. Watson Research Center.
[21] Neti, C., Potamianos, G., Luettin, J., Mattews, I., Glotin, H., Vergyri, D., Sison, J., Mashari., A., Zhou, J., *Audio-Visual Speech Recognition*, IBM T.J. Watson Research Center, Workshop 2000 Final Report
[22] Neti, C., Basu., S., Verma, A., Faruquie., T., Late Integration in Audio-Visual Continuous Speech Recognition, IBM T. J. Watson Research Center, Yorktown Heights, New York.
[23] Brand, M., *Coupled Hidden Markov Models for Modeling Interacting Processes*, MIT Media Lab Perceptual Computing / Learning and Common Sense Technical Report 405,
[24] Brand, M., Oliver, N., Pentland, A., *Coupled Hidden Markov models for complex action recognition*, MIT Media Lab Perceptual Computing / Learning and Common Sense Technical Report 407
[25] Oliver, H., Rosario., B., Pentland, A., *A Baysian Computer Vision System for Modeling Human Interactions*, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol 2. no. 8., August 2000
[26] Saul, L., Jordan, M., *Mixed Memory Markov Models: decomposing complex stochatic processes as mixtures of simpler ones*, Machine Learning, 1-11-1998, Kluwer Academic Publishers, Boston.
[27] Saul, L., Jordan, M., *Boltzmann Chains and Hidden Markov Models*, Advances in Neural Information Processing Systems 7. MIT Press, Cambridge, MA.
[28] Ghahramani, Z., Jordan, M., *Factorial Hidden Markov Models*, Machine Learning 29, 245-275 1997, Kluwer Academic Publishers, Boston.
[29] Logan, B., Moreno., P., J., *Factorial Hidden Markov Models for Speech Recognition*, Cambrige Research Laboratory, Technical Report Series, September 1997
[30] Kristjansson, T., Frey, B., Huang, T.S., *Event-coupled hidden Markov Models*, IEEE International Conference on Multimedia & Expo 2000, New York, Electronic Proceedings 2000
[31] Pan, H., Liang, Z., Huang, T.S., *A New Approach to Integrate Audio and Visual Features of Speech*, Department of Electrical and Computer Engineering, Beckman Institute for Advanced Science and Technology, University of Illinois, Urbana-Campaign

[32] Wojdel, J.C., Rothkrantz, L.J.M., *Obtaining Person-independent Feature Space for Lip-reading*, Knowledge Based Systems, Delft Univerity of Technology
[33] Wojdel, J.C., Rothkrantz, L.J.M., *Robust Video Processing for Lipreading applications*, in EUROMEDIA '2001, pages 195-199, Valencia, Spain, 2001.
[34] Wojdel, J.C., Rothkrantz, L.J.M., *Using Aerial and Geometric Features in Automatic Lipreading*, in Proc. Eurospeech 2001, Scandinavia, September 2001.

[35] Damhuis M., Boogaart T., in 't Veld, C., Versteijlen, M.,W. Schelvis, W., Bos, L., Boves L., *Creation and Analysis of the Dutch Polyphone Corpus*, Proceedings ICSLP '94, pp. 1803-1806, 18-22 September 1994,Yokohama, Japan
[36] Boogaart, T.I., Bos L., Boves, L., *Use of the Dutch Polyphone Corpus for Application Development*, Proc. IVTTA'94, pp. 145-148, 26-27 september 1994, Kyoto, Japan