

Semester Thesis
Speech Synthesis with Hidden Markov Models

Stephan Weiss
Supervisor: Dipl.-Ing. Harald Romsdorfer

Computer Engineering and Networks Laboratory
Speech Processing Group
ETH Zurich

13.07.2007

Abstract

This work presents first steps in synthesising speech with statistical parametric models. Previous works served as base and reference. The present work provides in particular tools and knowledge for training Hidden Markov Models in the speech processing tool HTK as well as the extraction of the models to implement them in MATLAB. In the MATLAB environment the models can be used to estimate speech parameters in order to achieve a text to speech synthesis. A synthesis filter finally converts the parameters into audio. In the present report different approaches for estimating the parameters and training the models will be discussed. In particular, on the training side, the improvement of the synthesis when taking dynamic features into account is explained. Furthermore issues of context dependent versus context independent models and the necessity of clustering is discussed always keeping an eye on how to use and adapt the tools made for the specific task. On the parameter estimation side, challenges in calculation speed and state selection are described. The solutions are here as well accompanied with explanations of how to use and adapt the appropriate tools. In particular an iterative algorithm for the estimation and the mixture selection is presented.

As speech data base a German corpora with 186 high quality utterances of approximatively 27 minutes was used. The utterances are recorded from a professional speaker in a echo damped studio.

Keywords: speech synthesis, statistical parametric synthesis, HMM training in HTK

Contents

1	Introduction	4
1.1	Estimation of the Parameters	5
1.2	Training the Models	5
1.3	Structure	5
2	General Speech Synthesis with Hidden Markov Models	7
2.1	Speech Data Base	8
2.2	Training Phase	8
2.2.1	Segmentation and Labeling	8
2.2.2	Monophone Models	9
2.2.3	Triphone Models	10
2.3	Synthesis Phase	11
2.3.1	Parameter Estimation	11
2.3.2	Audio Synthesis	14
3	Feature Selection and Quality Measurement	15
3.1	Feature Selection	15
3.1.1	Mel Frequency Cepstral Coefficients: MFCCs	15
3.1.2	Dynamic Features	16
3.1.3	Other Features	17
3.2	Quality Measurement	18
3.2.1	Mel Cepstral Distortion: MCD	18
3.3	Used Speech Database	19
4	Monophone Models	20
4.1	General Model-Training in HTK	20
4.1.1	Setting the Parameters	20
4.1.2	Building the Prototype Models	21
4.1.3	Setting the Paths and Training the Models	22
4.2	Parameter estimation	22
4.2.1	Importing the HMMs to Matlab	22
4.2.2	Estimating the Parameters	24
4.2.3	Synthesising the Audio Signal	28
4.3	Results from Monophone Models	28
4.3.1	MFCCs Only	29
4.3.2	MFCCs with 1 st Derivatives	31
4.3.3	MFCCs with 1 st Derivatives and 2 nd Derivatives	32
4.3.4	Limitations	34

5	Triphone Models	36
5.1	Triphone-Model Training in HTK	36
5.1.1	Building Triphone Models	37
5.1.2	Data Based Clustering	39
5.1.3	Tree Based Clustering	40
5.2	Parameter Estimation	42
5.3	Results from Triphone Models	42
5.4	Future Considerations	43
6	Conclusion and Outlook	45
6.1	Conclusion	45
6.2	Outlook	45
7	Bibliography	47

1 Introduction

To synthesise speech a commonly used technique is the unit selection synthesis where pieces of speech are concatenated in the desired order. This implies that for each phone which should be synthesised an equivalent has to be recorded and labeled properly. To build such a database with sufficient well labeled phones not only needs time but also requires an appropriate size of a database with high quality utterances. As the synthesised parts are concatenated parts from the database the synthesis quality always is at most as good as the recordings and therefore directly related to the database quality. Moreover the synthesised speech naturally sounds equal to the recorded utterances with respect to intonation, emotion and style. To build synthesisers with more variations much larger databases are required. Nevertheless unit selection synthesis has shown itself to be capable of producing high quality natural sounding synthetic speech when constructed from large databases of well-ordered and well-labeled speech. Figure 1 shows the rough unit selection method and its direct dependency on the speech database. Note that the extraction and concatenation is made from middle to middle of one to the other phone and not from the beginning to the end of a phone. With this, context dependent structures can be taken into account and smooth concatenation boundaries are achieved, however larger databases are needed to cover all context dependencies.

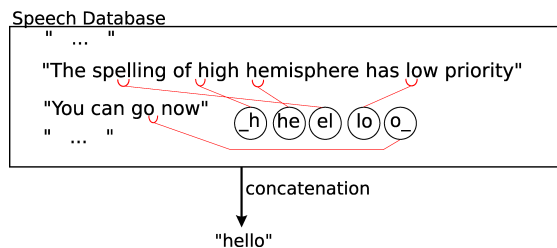


Figure 1: Rough schema of the unit selection process

If more variation and emotion is required from the synthesisers, a parametric model approach reduces the problem of large databases and database dependent quality of the synthesised speech. Moreover it is more robust to errors and unavailable utterances in the database. In particular in this work a speech synthesiser with Hidden Markov Models (HMMs) is described. It is, similar to the unit selection synthesis, trained with natural speech, however due to the parametric model approach, it allows a better modeling of variation.

The work can be divided into two main parts which represent the topology of the synthesiser itself. For the synthesiser parametric models have to be trained first in order to afterwards estimate parameters from the models. In the present work

the parameter estimation is discussed first because of less variation possibilities in the process compared to how training the models.

1.1 Estimation of the Parameters

The first part of the work shows how to estimate the parameters from the models to synthesise a given part of speech. The models are trained in the speech processing tool HTK [3]. Afterwards they have to be extracted and saved into a MATLAB compatible database in order to perform all estimations in MATLAB functions. The MATLAB environment was chosen for this task because of the necessity of calculations with extremely large (sparse) matrices. This environment allows fast calculations and provides good analysis tools for signals and matrices. Even though fast calculations are possible, the size of the matrices asks for a special algorithm to estimate the parameters in a reasonable time frame (i.e. seconds or minutes and not hours). Such an algorithm has been implemented following the ideas in [6].

1.2 Training the Models

The second part of the work was to find an accurate training for the models. This part consists of finding training parameters and in particular evaluating different base types of models such as models with and without dynamic features. Moreover differences between context dependent and context independent models are discussed in particular compared to the speech data base size and estimation robustness. Besides the evaluation of the trained models this part consists of providing the tools for setting up the trainings for the HTK speech processing tool.

1.3 Structure

In this report all tools for the model training with HTK and the parameter estimation from the models are described. Moreover different models are analysed and compared in order to provide a first overview of speech synthesis with Hidden Markov Model. In section 2 the general procedure of parametric speech synthesis is described. It shall give an overview of the topic in order to be able to follow easily the rest of the report. In section 3 the setup of the whole environment during the work is described. In particular the features used for the models are discussed and the method to compare different models (i.e. the error measurement) is explained. The section provides the information to understand the differences between the models trained later. Section 4 treats then the first group of models, the context independent models. First it describes shortly which tools are already provided.

Then it is explained how the parameter estimation is solved, and after the different model options are described together with improvements of the parameter estimation process. In the next section, in 5, the context dependent models are discussed and the necessary additional tools for HTK and the parameter estimation are described. Note that in this work no training parameter optimisation is made, however, problems and solutions with respect to context dependent models are presented. The report ends with the conclusion and the outlook for future work in section 6.

2 General Speech Synthesis with Hidden Markov Models

This section gives a general overview about speech synthesis with parametric probabilistic models and goes then further into details of the present work. In General the speech synthesis with Hidden Markov Models can be divided into two phases. First the training phase where the models are trained based on a speech data base and second the synthesis phase where the parameters are estimated from the models. Figure 2 shows the schematic procedure of the synthesis. For more details in particular about the training phase please refer as well to [2].

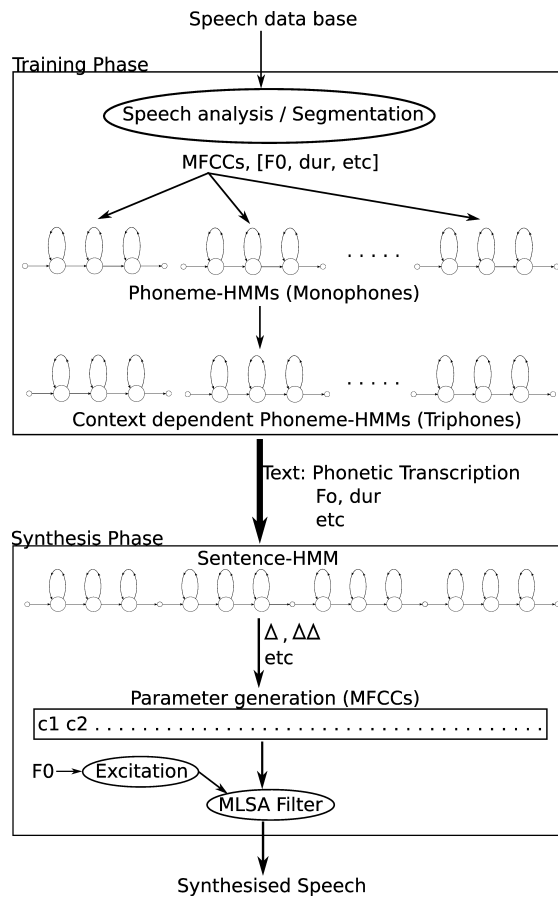


Figure 2: Schematic HMM synthesis

2.1 Speech Data Base

A speech data base serves for adequate training of the models. It is advantageous if the database consists of high quality phonetically balanced utterances from a professional speaker. The bigger the data base the better is the training because it contains more occurrences of each phone model and thus can be trained more accurately. Moreover small data bases are likely to have a relatively restricted selection of phones which can be trained. This means that some models of special phones needed during the synthesis do not even exist and workarounds or interpolations have to be found if possible. A critical and difficult decision is whether to use a smaller database of very high quality or a larger data base with low quality. In general the amount of data should receive more priority than the quality because of “perfect” trained models which can cause singularities during the calculations. This topic, however, is treated more in detail in section 4.

2.2 Training Phase

At this stage the user has to decide which features the models should be trained for. In general as well as in this work *Mel Frequency Cepstral Coefficients* (MFCCs) and their first and second derivatives (deltas and deltadeltas) are used. The MFCCs are explained in section 3.1.1. Other works also include the fundamental frequency and the duration together with other phonological properties such as syllable, word and phrase boundaries or different accentuation. In section 3 some features are described in more detail. The features are extracted from the original audio files and used for all future training with the models.

2.2.1 Segmentation and Labeling

The first alignment consists of dividing the signal length by the number of phones containing in the signal into equivalent segments. This results from a so called flat start where all models are initialised with the same values. The extracted features per frame of each utterance are put in a feature vector per frame. Usually a frame is extracted every 2 to 5ms. With the feature vectors and a Baum-Welch algorithm the models are then trained using the utterances in the speech data base. With the training the audio signals are segmented and labeled more exactly step by step converging to what manual segmentation would yield. Figure 3 shows a possible segmentation before the training (flat start) above the signal and after the training below the signal.

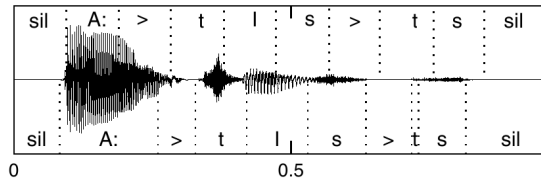


Figure 3: Signal labeling of “Artists”: Above: Initial segmentation. Below: Final segmentation

2.2.2 Monophone Models

With the above iteration by applying the Baum-Welch algorithm to train the models with the feature vectors models for a single context independent phone are obtained. One model usually consists of 3 emitting states which represent the beginning, the middle and the end of the phone respectively. This approach is also used in this work. An additional non emitting start and end node represent the boundaries of the model and can be omitted when concatenating models as they do not emit feature vectors. The model structure is chosen to be a left-to-right HMM in order to credit the chronological character of the inner phone parts. There might be an exception, however, and a transition from the first to the third state can be introduced to shorten the phone by omitting the middle state. This is often applied for silences in order to model short silences as well. Figure 4 shows a typical three state left-to-right model with the optional transition and the transition probabilities.

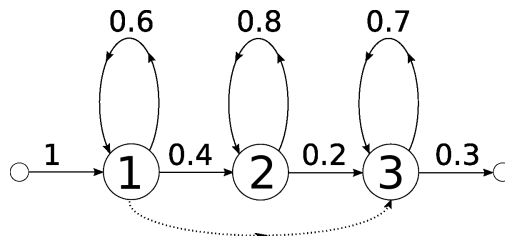


Figure 4: 3 state left-to-right HMM with optional transition (dotted)

These context independent models can already be used for the synthesis phase.

Continuous Models with Mixtures

The HMMs discussed here are continuous models with more than one Gaussian distribution. The number of the so called mixtures can vary. A state with more than one mixture (i.e. Gaussian distribution) can model the probability distribution of the feature vectors more adequate and is therefore preferred to single

Gaussian models. Each mixture is weighted with the weight factor w_i for the i -th mixture such that

$$\sum_{i=1}^N w_i = 1 \quad (1)$$

for all N mixtures. Such a state can then be seen as N states with the transition probabilities weighted with the mixture weight factor w_i . Figures 5 and 6 depict the structure.

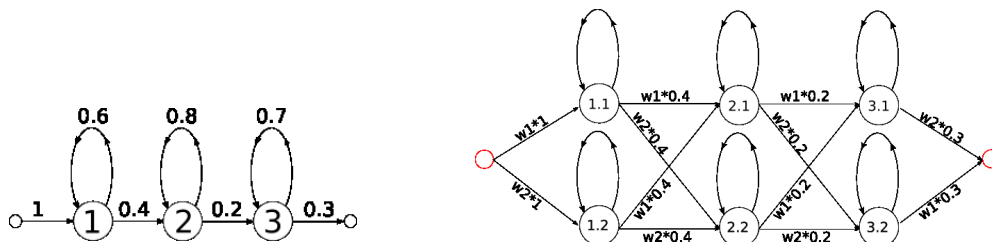


Figure 5: Mixture HMM: Compact

Figure 6: Mixture HMM: Fully drawn with 2 mixtures

Details on how to select the appropriate mixture and information about the number of mixtures per state are discussed in section 4.

2.2.3 Triphone Models

If the database is large enough it is advised to refine the context independent models to context dependent models. The latter models take the co-articulation into account and provide therefore more reliable parameters for the synthesis. For example the l sound is different between two o 's as in "Bartholomeus" than between two i 's as in "Illinois". Triphone models still model only one phone with a 3 state left-to-right model, however, there are several different models for the same phone but in other contexts. As *context* different variations can be implemented. In this work only the left and right phone neighbour (indicated with - and + respectively) are taken into account, therefore in this work context dependent models are equivalent to triphone models. However, other features as stress, word boundaries etc can be considered.

With this splitting the number of models increases dramatically and the speech data base sizes must be very large in order to cover all models with enough occurrences for a good training. To overcome this problem, the models and their states are clustered and tied. If a state of two models or the whole models do not differ more than a defined threshold they can share the specific states together such that they are both trained if either of the sequence occur in the training set. An example might be the middle state of the triphone models of an l between

two a 's or between two e 's. If the middle state of these models is in the same cluster it will be trained simultaneously whereas the first and third state only is trained with the corresponding word to the model. Figure 7 shows an example of clustering context dependent models of the l sound

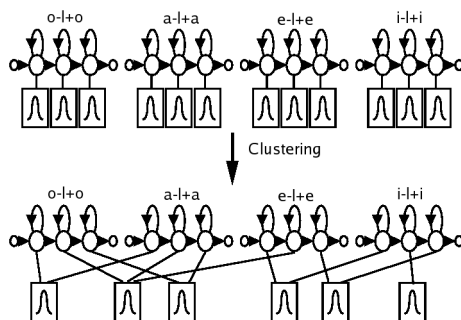


Figure 7: Clustering of triphone models representing different l sounds

Different kinds of clustering with different thresholds and their advantages and disadvantages are discussed in more detail in section 5. Note, however, that with triphone models (i.e. context dependent models) clustering is a must in order to adapt the number of models to the size of the speech database.

To train the triphone models the speech database has to be compatible to the models which means that the utterances have to exist in context dependent notation otherwise a rewriting of the data has to be done. Then the already trained monophone models can be refined by training them into triphone models. After this step the training phase is completed and speech synthesis can be performed as follows.

2.3 Synthesis Phase

The speech synthesis with the above trained models can be divided into two parts. First the feature vectors for a given phone sequence have to be estimated and second the estimated feature vectors have to be transformed into an audio signal. Note that the synthesised text has to be written in its phonetic transcription indicating the duration of each phone and its fundamental frequency. The phonetic transcription serves in the model selection and parameter estimation process while the fundamental frequency is necessary for synthesising the audio signal.

2.3.1 Parameter Estimation

The phonetic transcription together with the duration for each phone basically dictate the state sequence which is to chose for the synthesis. However, because

each phone is not only modeled with one but with three states and each state of those consists of usually more than one mixture a state sequence is to chose through this network. Remember that one state with multiple mixtures can be represented by multiple single mixture states with their transition probabilities weighted according to the mixture weights. The network from which the sequence has to be chosen consists therefor of concatenated phone models as depicted in figure 6. Neither the feature vectors are known to estimate the state sequence with a Viterbi algorithm nor the state sequence is known to estimate the feature vectors. Hence other estimations must be used.

Estimating the State Sequence

Two estimations for retrieving the state sequence are applied. The total length of each phone is known together with the phonetic transcription. First, the three states of the phone model which fits the phonetic transcription have to be spread over the total phone length. The number of beginning, middle and end states are calculated by normalising their transition probabilities and dividing the total phone length by the normalised probabilities. If for example the total duration of the phone represented by the model in figure 5 is 40 states (calculated from the given phone length in ms and the frame shift after each a feature vector is extracted), the first state is repeated $0.6 * n * 40 = 11$ times, the second state $0.8 * n * 40 = 15$ times and the third state $0.7 * n * 40 = 14$ times with $n = 1/(0.6 + 0.8 + 0.7)$ being the normalisation factor.

Second, the best mixture for each state in the sequence has to be chosen to estimate the feature parameters. In general the sharpest dominant mixture already yields a good estimation, however, in section 4 a more sophisticated iterative algorithm from [6] is presented. The selection of one particular mixture is done to reduce computational complexity while at the same time - being an adaptive algorithm - keep a detailed distribution curve of the features. Now the sate sequence is definitely defined and the feature vectors can be estimated

Estimating the Feature Vectors [4]

First the feature vectors have to be described in more detail. Assumed that Mel Frequency Cepstral Coefficients (MFCCs) and their derivatives are used as features as it is done in the present work, the feature vectors for frame t can be written as $\mathbf{o}_t = [\mathbf{c}'_t, \Delta \mathbf{c}'_t, \Delta^2 \mathbf{c}'_t]'$ where $\mathbf{c}_t = [c_t(1), c_t(2), \dots, c_t(M)]$ is the vector with the first M MFCCs and $\Delta^n \mathbf{c}_t$ are the n-th derivatives according to

$$\Delta^n \mathbf{c}_t = \sum_{i=-L_n}^{L_n} w_n(i) \mathbf{c}_{t+i} \quad n = 0, 1, 2 \quad (2)$$

where $L_0 = 0, w_0(0) = 1$. Note that the formula also holds for the MFCCs themselves in other words for the 0th derivative. Derivatives with neighborhood one (i.e. $L_1 = 1, L_2 = 2$ because of neighborhood one relative to first derivatives) would for example be calculated as follows

$$\Delta \mathbf{c}_t = \frac{1}{2}(\mathbf{c}_{t+1} - \mathbf{c}_{t-1}) \quad (3)$$

$$\begin{aligned} \Delta^2 \mathbf{c}_t &= \frac{1}{2}(\Delta \mathbf{c}_{t+1} - \Delta \mathbf{c}_{t-1}) \\ &= \frac{1}{4}(\mathbf{c}_{t+2} - 2\mathbf{c}_t + \mathbf{c}_{t-2}) \end{aligned} \quad (4)$$

where for the first derivative

$$w_1(-1) = -0.5, w_1(0) = 0 \text{ and } w_1(1) = 0.5$$

and for the second derivative

$$w_2(-2) = 0.25, w_2(-1) = 0, w_2(0) = -0.5, w_2(+1) = 0 \text{ and } w_2(+2) = 0.5$$

The calculation of the w-parameter is explained in detail in section 4. As for now, with equation 2 the feature vector \mathbf{o}_t can be rewritten in linear dependency with the MFCC vectors. All feature vectors can be taken together to one large vector $\mathbf{O} = [\mathbf{o}'_1, \mathbf{o}'_2, \dots, \mathbf{o}'_T]'$, similarly all MFCC vectors can be expressed in one large vector $\mathbf{C} = [\mathbf{c}'_1, \mathbf{c}'_2, \dots, \mathbf{c}'_T]'$ where T denotes the number of states for the whole synthesis part. Using the linear dependency derived above \mathbf{O} can be expressed by

$$\mathbf{O} = \mathbf{W} * \mathbf{C} \quad (5)$$

The structure and calculation is left to the section 4 where it is described in full detail for the present used settings.

To estimate the best feature vectors $P(\mathbf{O}|\mathbf{q}^*, \lambda, T)$ has to be maximised. This is the probability of a feature vector sequence \mathbf{O} given the estimated state sequence \mathbf{q}^* , the Hidden Markov Models defined by λ and total length T (i.e. the total number of states to synthesise). To this end

$$\mathbf{O}^* = \arg \max_{\mathbf{O}} P(\mathbf{O}|\mathbf{q}^*, \lambda, T) \quad (6)$$

has to be calculated. With equation 6 the following term has to be maximised with respect to \mathbf{O}

$$\begin{aligned} P(\mathbf{O}|\mathbf{q}^*, \lambda, T) &= P(\mathbf{WC}|\mathbf{q}^*, \lambda, T) \\ &= \frac{1}{\sqrt{(2\pi)^{fMT} |\mathbf{U}|}} \exp\left(-\frac{1}{2} (\mathbf{WC} - \mu)' \mathbf{U}^{-1} (\mathbf{WC} - \mu)\right) \end{aligned} \quad (7)$$

Where $\mu = [\mu'_{\mathbf{q}_1^*}, \mu'_{\mathbf{q}_2^*}, \dots, \mu'_{\mathbf{q}_T^*}]'$ and $\mathbf{U} = \text{diag}[\mathbf{U}'_{\mathbf{q}_1^*}, \mathbf{U}'_{\mathbf{q}_2^*}, \dots, \mathbf{U}'_{\mathbf{q}_T^*}]'$ and $\mu'_{\mathbf{q}_t^*}$ and $\mathbf{U}'_{\mathbf{q}_t^*}$ are the mean vector and the diagonal covariance matrix of the state q_t^* of the estimated state sequence q^* . The variable f denotes the number of used derivations where $f = 1$ if \mathbf{O} only consists of the MFCCs, $f = 2$ if the first derivatives are implemented and $f = 3$ for including also the second derivatives of the MFCCs. By setting

$$\frac{\partial P(\mathbf{O}|\mathbf{q}^*, \lambda, T)}{\partial \mathbf{C}} = \mathbf{0}_{\mathbf{T}M \times 1} \quad (8)$$

the equation

$$\mathbf{R}\mathbf{C} = \mathbf{r} \quad (9)$$

with

$$\begin{aligned} \mathbf{R} &= \mathbf{W}'\mathbf{U}^{-1}\mathbf{W} \\ \mathbf{r} &= \mathbf{W}'\mathbf{U}^{-1}\mu \end{aligned}$$

is obtained and thus a MFCC sequence in \mathbf{C} can be estimated. Considering that $\mathbf{W} = [fTMxTM]$ and $\mathbf{U} = [fTMxfTM]$ a fast algorithm has to be derived to compute \mathbf{C} efficiently. As an example a short sentence might be 5 sec long and each 2ms a feature vector is extracted. To synthesise the sentence $T = 5/0.002 = 2500$ states have to be concatenated. Using 26 MFCCs with their first and second derivatives yields $M = 26$ and $f = 3$. The \mathbf{W} matrix will therefore be a $[195'000 \times 65'000]$ matrix! Even though sparse (not diagonal like \mathbf{U}), the required computational power is huge. An iterative solution from [6] is described in detail in section 4.

2.3.2 Audio Synthesis

The estimated MFCC vectors allow now the synthesis of the audio signal. To this end a synthesis filter is implemented. The most frequently used filter is a Mel Log Spectral Approximation. Together with the excitation information from the fundamental frequency and the flag if a phone is voiced or unvoiced, the MFCCs are reversely calculated to the spectrum they represent. Then the audio signal is generated based on the synthesised spectrum. For detailed information about the calculation of the MFCCs refer to section 3. However, as this is not part of the current work the exact transformation of the estimated MFCCs to the corresponding audio signal is not discussed here.

3 Feature Selection and Quality Measurement

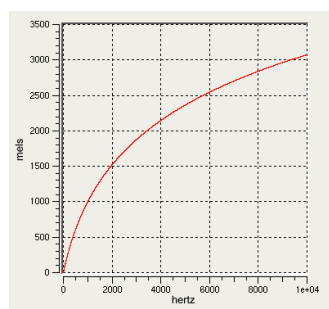
Prior to discuss the parameter estimation and the model training it is important to define a quality measurement of the estimated parameters as well as of the synthesised signal. In this section not only the error measurement is discussed but also the general quality measurement and the selection of features.

3.1 Feature Selection

In order to synthesise the audio signal, representative features of the speech signal are needed. The most commonly used features in speech processing are Mel Frequency Cepstral Coefficients (in short MFCCs). Other features as the fundamental frequency, accentuation, word boundaries etc can be added to refine the model training. In the following some of these features are discussed.

3.1.1 Mel Frequency Cepstral Coefficients: MFCCs

The MFCCs contain the information for the synthesised signal. A feature for speech processing should at the same time minimize the inner class variance while maximising the outer class variances. Fullfilling this requirement fairly well the MFCCs are similar for the same phone and different from one phone to another. They are calculated from the spectrum of the audio signal by filtering it with a mel filterbank and taking the logarithm. A filterbank converts the frequency to the *mel* scale which represents the human perception sensitivity. Thus higher frequencies are mapped closer together than lower frequencies. Equation 10 shows the relation of the two frequency scales and figure 8 depicts the plot of the formula.



$$m = 1127 * \ln(1 + f/700) \quad (10)$$

Figure 8: Mel-Hz plot

Where m is the frequency in *mel* and f the frequency in *Hz*. In order to receive mel coefficients, the filterbank bins a range of frequencies. Figure 9 shows a typical filterbank with 24 filters. Note that the bin sizes increase

with the frequency in Hz according to the mel transformation. In the mel scale they are equally distributed.

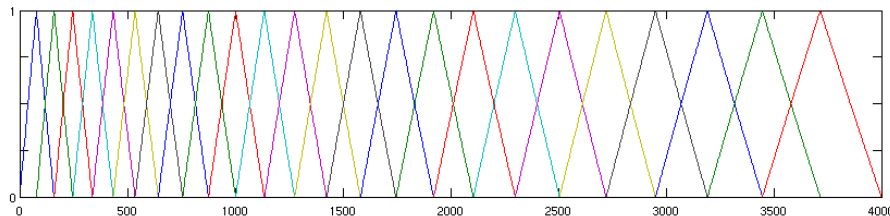


Figure 9: Mel filterbank in Hz with 24 filters

The MFCC extraction can be summarised in figure 10.

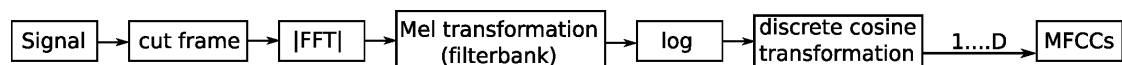


Figure 10: Schematic MFCC extraction from audio signal

The discrete cosine transformation (DCT) simply represents the real $|FFT|^{-1}$ transformation. This can be expressed with equation 11

$$c(m) = \frac{1}{J} \sum_{j=1}^J (\log S_j) \cos \left[m \left(j - \frac{1}{2} \right) \frac{\pi}{J} \right] \quad 0 \leq m \leq D \quad (11)$$

where D are the number of coefficients to be extracted, J the number of filters in the mel filterbank and S_j is the Energy of the j -th filter.

3.1.2 Dynamic Features

Additionally to the MFCCs their derivatives can be added to the models as dynamic features. In the synthesis these features provide a certain smoothness and thus better performance as it is discussed in section 4. Equation 12 shows the general calculation of the derivatives of the MFCCs.

$$\Delta^n \mathbf{c}_t = \frac{\sum_{i=-\hat{L}_n}^{\hat{L}_n} i \mathbf{c}_{t+i}^{(n-1)}}{\sum_{i=-\hat{L}_n}^{\hat{L}_n} i^2} = \sum_{i=-L_n}^{L_n} w_n(i) \mathbf{c}_{t+i} \quad n = 0, 1, 2 \quad (12)$$

where $L_0 = 0$, $w_0(0) = 1$ and \hat{L}_n is the neighborhood of $(n-1)$ th derivatives to build the n -th derivative. Note that when $\hat{L}_1 = 2$ and $\hat{L}_2 = 2$ then $L_1 = 2$ but $L_2 = 4$.

3.1.3 Other Features

Generally two kinds of features can be distinguished. On one hand there are the real features which contain the audio information to be synthesised from a given model. These are namely the MFCCs with their derivatives in this work. On the other hand auxiliary features can be defined which help to define whether another model should be trained or synthesised. While the real features optimise the choice of parameters within the the model, auxiliary features split the models into more specific ones. For example considering the context in order to build triphone models splits the “l” model into an “ $o-l+o$ ” model, representing the “l” in “olo” and an “ $i-l+i$ ” model representing the “l” in “ili” and so on. Thus auxiliary models always carry the problem of adapting the speech database size upwards or need better clustering.

Note that the real features mostly are independent of each other because of diagonal matrices. Thus the derivatives of the n^{th} MFCC only optimise the choice of the n^{th} MFCC. If the speech data base is large enough more auxiliary features help to diversify the phone characteristics. The database must contain enough occurrences of the distinct phone models to train each of them accurately. A training with a small speech database according to the amount of auxiliary features will result in either poorly trained models or, if tying is applied and big clusters are allowed, the additional auxiliary features will vanish in the tying and clustering process.

Context Dependency

If context dependent models are trained this mostly includes considering some phones previous and after the current phone, however the model itself is still for only the present phone. In the present work one phone at each side was taken into account. Considering this the amount of models increases dramatically and the speech database must be of a certain size. In section 5 first steps with different clustering methods are shown with a relatively small database. Tying and clustering is a must.

Fundamental Frequency and Duration

Before adding the fundamental frequency its effect should be analysed. It occurs in a continuous form in the database and has thus to be quantised accordingly. For each bin a model can then be trained. However, as the MFCCs are usually not supposed to be very sensitive to changes of the fundamental frequency the small difference between the models might vanish in the tying or clustering process unless a large speech database is used. Note that adding the fundamental frequency as real feature in the models will not bring any quality improvement as it is estimated independently from the MFCCs, for the dependency is not known

and diagonal matrices are used. Estimating the fundamental frequency only serves if the synthesis is wanted to have the same fundamental frequency as the original speaker and no extra information is provided.

The same holds for applying the duration as real or auxiliary feature. However the differences of the MFCCs using the duration as auxiliary feature is more significant than using the fundamental frequency.

Other Features

Given the database is large enough, adding more auxiliary features as syllable and word boundaries, accentuation, intonation, phrase types etc can improve the synthesis quality. For each of the features a specific quantisation has to be applied in order to train the specific models.

3.2 Quality Measurement

For the quality measurement of the synthesised parameters and signal one objective equation based and one subjective audio based measurement is used. The audio measurement simply consists of synthesising the estimated parameters with the synthesiser provided by H. Romsdorfer.

3.2.1 Mel Cepstral Distortion: MCD

This error measurement is an internationally used formula and applied in particular in paper [11]. Equation 13 shows the calculation for the MCD of frame t .

$$E_{MCD_t} = 10/\ln(10) * \sqrt{2 \sum_{i=1}^{dim} (mc_{i,t}^o - mc_{i,t}^e)^2} \quad (13)$$

where dim is the dimension of the feature vector, $mc_{i,t}^o$ and $mc_{i,t}^e$ are the i^{th} element of the original and estimated vector of frame t respectively. Note that this formula is not normalised and depends not only on the feature vector dimension but also on the normalisation applied to the vectors. The higher coefficients being normally smaller than the lower ones this formula also is more sensitive to minimise the error of the lower coefficients. Even though internationally used comparison with other works is restricted due to the mentioned weaknesses. However a lower error usually implies a better synthesis. Thus equation 13 serves well for measuring the own progress provided that the settings for the vector dimension and normalisation are left unchanged. For this purpose equation 13 can be simplified to the root square error as in equation 14 by eliminating the scaling factors.

$$RSE = \sqrt{\sum_{i=1}^{dim} (mc_{i,t}^o - mc_{i,t}^e)^2} \quad (14)$$

Normalisation

Basically two different kinds of normalisations can be applied. First the normalisation over the actual sentence to be synthesised is fast and does not need other information as the sentence itself. However in this work a normalisation over the whole speech database is applied in order to have better comparison between the synthesised sentences using equation 13. The normalisation type is a simple mean and variance normalisation where all data of the database are considered as a Gaussian distribution where mean and variance are estimated. This Gaussian is then mapped to a normalised Gaussian with zero mean and variance one. With the mapping information any synthesised values are mapped in the same way and thus all synthesised data are normalised in the same manner.

3.3 Used Speech Database

The used database contains German utterances from a professional speaker. The sentences are recorded in an echo damped studio and are thus of very high quality. There are 186 utterances in approximately 27 minutes which qualifies the database as a relatively small data base. The recordings are sampled in 16KHz mono and stored as wave files. Additionally the database contains manually segmented files which contain the duration and fundamental frequency for each phone per sentence, moreover the sentences are written in their phonetical transcription as whole sentences and as syllables.

4 Monophone Models

Once the feature selection is made and the quality measurement defined different models are to be trained to optimise the synthesis. In this section in particular monophone models with different dynamic features are discussed. However prior to the model discussion a short introduction is given on how to use the HTK scripts to train in general the models. Moreover, the feature estimation does not let as much of free space in the process as the model training with its different parameters and thus the general feature estimation routine written in this work will be presented first together with the HTK scripts. In section 5 where context dependent models are discussed additional scripts will be highlighted.

4.1 General Model-Training in HTK

HTK is a public accessible speech processing tool based on Hidden Markov Models. It implies several functions which facilitate the work with HMMs. Therefor it is used here to train the different models. All details about the vast functions of HTK can be read in the HTK manual [3]. Some preexisting scripts shall be described in the following. Note that the paths and filename are related to the present scripts and can be changed given the change is also made in the scripts. The training philosophy of HTK is an iterative one meaning that first models are trained with the basic settings and then refining steps are applied. This way manual refining steps such as building triphone models from monophone models are also possible.

4.1.1 Setting the Parameters

All details about the settings can e found in the manual however the most important ones are listed here. The settings on how to train the models are stored in *HCopy.wav.cfg* in the *htkscripts* directory. To train the models with MFCCs use

```
TARGETKIND = MFCC_D_A_E
TARGETFORMAT = HTK
```

where the optional parameters *_D* stands for adding the 1st derivatives and *_A* for the 2nd derivatives. The parameter *_E* optionally adds as well the zeroth MFCC - which is the energy - to the feature vector. *TARGETFORMAT* should be *HTK* for future work with the models. To define the neighbourhood window for the derivatives use

```
DELTAWINDOW =  $\hat{L}_1$                 for the first derivatives and
ACCWINDOW =  $\hat{L}_2$ 
```

for the second derivatives. Where \hat{L}_n defines the number of neighbours of each side. Note that the window sizes are the \hat{L}_1 and \hat{L}_2 respectively in equation 12 in section 3. This has in particular to be taken into account for building the values

of $w_n(i)$ relative to the MFCCs in the same equation.

To calculate the number of elements in a feature vector the number of MFCCs has to be set with

$$\text{NUMCEPS} = N$$

and the dynamic features have to be added. If for example $TARGETKIND = MFCC_D_A_E$ and $NUMCEPS = 13$ then the feature vector has a length of 42 elements with the following structure:

$$\begin{aligned} \mathbf{c}_t = & [c_t(1), c_t(2), \dots, c_t(13), c_t(0), \\ & \Delta c_t(1), \Delta c_t(2), \dots, \Delta c_t(13), \Delta c_t(0), \\ & \Delta^2 c_t(1), \Delta^2 c_t(2), \dots, \Delta^2 c_t(13), \Delta^2 c_t(0)] \end{aligned} \quad (15)$$

Note that the energy term is always added at the end and counts additional to the 13 MFCCs. The window size and in particular the frameshift after each a feature vector is extracted is defined here with

$$\text{WINDOWSIZE} = N_{Win}$$

$$\text{TARGETRATE} = N_{Frm}$$

and are represented in steps of 100ns, thus a frameshift of 2ms is achieved by setting $TARGETRATE=20000.0$. The other parameters are self explaining or explained in the *HCopy.wav.cfg* and in the HTK manual.

4.1.2 Building the Prototype Models

HTK uses prototype models to train the model for the speech data base. Thus these prototypes have to be initialized properly. Also in the *htkscripts* directory the file *proto_begin* has to be changed as follows

```
~ o
<STREAMINFO> 1 N_tot
<VECSIZE> N_tot <NULLD> <TARGETKIND>
```

where N_{tot} is the total number of elements calculated in one feature vector and $TARGETKIND$ has to be replaced by the expression used in *HCopy.wav.cfg*, for example *MFCC_D_A_E*. Similarly the header of the file *prototype* has to be adapted. Moreover in that file at each state the number of mean values and variances has to be adapted. This has not only to be done for the number itself but also for the amount of numbers below the lines $\langle MEAN \rangle N_{tot}$ and $\langle VARIANCE \rangle N_{tot}$. However the values of these buffer numbers are irrelevant. If the number of states change it has to be specified here as well by simply copying more states and adapting the transition matrix below the line $\langle TRANSP \rangle N_{states}$. Where N_{states} is counted with the start and end node included and possible transitions are marked with a non-zero value in the matrix, the value then, however, is irrelevant and will be trained.

4.1.3 Setting the Paths and Training the Models

At this stage all parameters for the HTK toolbox are set. To train the models automatically a script *segment* in the *segtools* directory exists. However first it has to be specified where to train and store the models with the corresponding file. For this purpose the variables in the *config.ger_schu* file have to be adapted to the desired paths. The variables are self explaining and described in the file itself. Note however that for the number of iterations the variables *MIXTURESTEPS* and *ERESTSTEPS* have to be set as strings with increasing numbers. For example for 3 iterations after each mixture duplication set *ERESTSTEPS*='1 2 3' because the variables are read later on in other scripts step by step.

Once all paths are set all variables can be exported and the script *segment* in the *segtools* directory can be started to start the automatic training. Depending on the settings and parameters to train it can take a few hours to more than half a day. The last built *hmms.mmf* file in the specified directory contains then all the trained models. The *segment* script trains the models first without silences in between the utterances to train the silence model and in a second training silence models between the utterances are introduced where appropriate. First models with only one mixture are trained and after completion of each training phase with the defined loops in *ERESTSTEPS* the mixtures are doubled until the end of *MIXTURESTEPS* is reached.

4.2 Parameter estimation

The basic schema for the parameter estimation from the HMMs is described in section 2 under 2.3.1. The cost efficient routine described in [6] is implemented here and explained in more details. The routine is written in MATLAB and thus the models from HTK have to be imported first to a MATLAB format.

4.2.1 Importing the HMMs to Matlab

To import the HMMs from HTK to a mat file for MATLAB the function *readHMMs* was written. The function not only saves a mat file calls *cdhmms.mat* but also creates a lookup table in *hmms.list* for the values of the different phones and their states. This lookup table is particularly useful if the amount of models is so large that matlab cannot load the mat file because of memory issues. The lookup table provides then a relatively fast and memory efficient way to read the needed values. The function takes the following parameters in the given order

mmfile This is the master model file (.mmf) created by HTK containing the trained models. The number of mixtures, states per model and the dimension of

the feature vectors are read automatically, however it is assumed that all models have the same structure.

phonelist The phonelist with the entries if a model is tied is only necessary to store the name of the file in the lookup table so it is not necessary to mention it later in the synthesis process.

statesperphone, dim, mix They are also arguments which are written to the lookup table and represent the number of states per model without the initial and end state, the number of MFCCs used including the energy term and the number of mixtures respectively.

resume With this feature, a broken readout can be continued at a specific model. The name of the model has to be in HTK notation and without quotation marks.

If only one argument is given it is taken as *mmfile*, only two arguments are taken as *mmfile* and *resume*. The stored *cdhmms.mat* file contains the struct *cdhmm* with the fields

- **.name:** Name of the model
- **.mu:** [S x M x D] matrix containing the mean values of the Gaussians of the model. Where S is the number of states in the model without initial and ending state, M the number of mixtures and D the total dimension of a feature vector.
- **.sigma:** [S x M x D] matrix containing the variance values corresponding to the mean values.
- **.c:** [S x M] matrix containing the mixture wights for each mixture M in state S.
- **.a:** [\hat{S} x \hat{S}] matrix containing the transition matrix, this time with the initial and end states included.

The *hmm.list* file with the lookup table stores in the first line the master model file name with the other arguments discussed above. Then each line starts with the name of the model followed by the number for mean and variances for the first state and all mixtures, then for the second state and all mixtures and so on until the last two numbers declare beginning and end line of the transition matrix. A model with 3 emitting states and 2 mixtures may have the following entry

“sample” $M_{1,1}$ $V_{1,1}$ $M_{1,2}$ $V_{1,2}$ $M_{2,1}$ $V_{2,1}$ $M_{2,2}$ $V_{2,2}$ $M_{3,1}$ $V_{3,1}$ $M_{3,2}$ $V_{3,2}$ P_S P_E

where the model name is “sample” and $M_{s,m}$ and $V_{s,m}$ is the line number in the .mmf file for the mean and variance values of the s^{th} state and m^{th} mixture respectively. P_S and P_E give then the star and end line for the transition matrix. Each value is separated with a *tab* character.

4.2.2 Estimating the Parameters

Once the models are imported to the MATLAB notation the parameters can be estimated. Remember that phonetic transcriptions as well as the fundamental frequency and the duration of each phone to be synthesised is provided from the high quality speech database. Thus to estimate the parameters in order to synthesise the desired text, a sample from the speech database is taken and the parameters read into MATLAB. First the state sequence has to be estimated as described in 2.3.1 which is to normalise the transition probabilities and divide the total length of the phone by these normalised probabilities. The exact formula to chose the appropriate mixture is given in equation 16

$$\hat{i}_t = \arg \max_{i_t} \left(\ln(c_{q_t, i_t}) - \frac{1}{2} \ln(|\mathbf{U}_{\mathbf{q}_t, \mathbf{i}_t}|) \right) \quad (16)$$

where c_{q_t, i_t} and $\mathbf{U}_{\mathbf{q}_t, \mathbf{i}_t}$ is the mixture weight and diagonal covariance matrix of the state q and the mixture i at time t . The equation therefor selects the feature vector with the sharpest mixtures as an initial choice. Once having the state sequence means and variances can be extracted and the maximization of equation 7 can be performed.

The Function *estimparam*

The Matlab function *estimparam* automates these steps. It takes the following arguments

- file: The label file .lab with the phonetic transcription and durations
- hmms: The models as .mat file
- tiefile: A list with all models and tied models in a separate column
- winshift: The frameshift after each a feature vector is extracted
- w: A [1xf] vector where f is the order of the last implemented derivative. The elements of the vector are the neighbourhood for the $(n - 1)^{th}$ derivative to calculate the n^{th} derivatives. In other notation the \hat{L}_n in 12.

It calculates first the normalisation parameters over the whole speech data base and reads the lengths for each phone and the total utterance to be synthesised. The state sequence is then selected according to the discussion above in a loop for each phone and the means, variances and mixture weights are concatenated accordingly in vectors. After this step the matrices \mathbf{U} and the vector μ used for equation 9 are built. Then the W matrix is calculated and all information is passed to the optimisation function *opt.c*. This returns the estimated MFCCs which then are normalised by using the above calculated normalisation parameters. Moreover a smoothing filter is applied to eliminate disturbing fluctuations. The filter order is given in the variable *order*. The MCD is calculated as described in 3.2.1 and the estimated MFCCs are compared to the original MFCCs extracted from the feature file *.fea* generated from HTK. The plots with the comparisons as well as the MCD give then a rough image about the quality of the performed estimation.

Calculating W

As the W matrix reflects the equations of calculating the derivatives in 12 its structure and entries depend on the neighbourhoods \hat{L}_n . With the parameter w these neighbourhoods are passed to a function *makeW* which generates a template matrix *wt* for calculating the n^{th} derivative of frame t . To this end the first equation in 12 is applied to each derivative recursively in order to calculate the $w_n(i)$ of the second equation in 12 and thus having the direct relation between the n^{th} derivatives and the MFCCs. Note that only the coefficients in the mentioned equations are calculated. The code for calculating the coefficients for the first derivatives is therefor

```
% Build divisors of deltas after HTK definition (p.68 in HTK book)
dDiv = sum((1:delta).^2)*2;
% Calculate deltas
wt(1,n+1)=1;
for i=1:delta
    wt(2,n+1+i)=i/dDiv;
    wt(2,n+1-i)=-i/dDiv;
end
```

where *delta* is the \hat{L}_1 neighbourhood. For the argument $w=[1]$ this yields a template matrix *wt*

$$\begin{array}{ccc} 0 & 1 & 0 \\ -0.5 & 0 & 0.5 \end{array}$$

which leaves the current MFCCs unchanged (first row) and calculates the first derivative with one MFCC neighbour at each side. For $w=[1 \ 1]$ the coefficients for the second derivatives will be present accordingly in a third row to *wt*.

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & -0.5 & 0 & 0.5 & 0 \\ 0.25 & 0 & -0.5 & 0 & 0.25 \end{bmatrix}$$

note here that $\hat{L}_2 = 1$ but the neighbourhood which is directly dependent from the MFCCs is $L_2 = 2$ each side and thus the template matrix is a $[3 \times 5]$ matrix. The size always is $[f+1 \times 2L_f+1]$ where f is the order of the highest derivative and L_f is the neighbourhood for the f^{th} derivative with respect to the MFCCs. The template is now concatenated such that

$$\mathbf{O} = \mathbf{WC}$$

\mathbf{O} being the feature vector and \mathbf{C} being the MFCC vector as described in 2.3.1.

To explain the W matrix it is split in different blocks according to the structure of \mathbf{O} . The first M elements of \mathbf{O} is the MFCC vector extracted from the first frame. Remembering that \mathbf{C} is a $[M \times 1]$ vector where M is the number of MFCCs per frame and T is the number of frames in the audio signal to be synthesised, the first $M \times M T$ rows of W are

$$\begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & 0 & \dots & 1 & \dots & 0 \end{bmatrix}$$

then in the \mathbf{O} vector the next M elements are the first derivatives of the first MFCC vector. With a neighbourhood of $\hat{L}_1 = 1$ the following $M \times M T$ rows in W are

$$\begin{bmatrix} 0 & 0 & 0 & \dots & 0.5 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 & 0.5 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 & 0.5 & 0 & \dots & 0 \\ \vdots & & \ddots & & & & & \ddots & & \vdots \\ 0 & 0 & & \dots & & & & & 0.5 & 0 \end{bmatrix}$$

Note that the -0.5 term vanishes because of boundary conditions. The m^{th} block will therefor look like

$$\begin{bmatrix} 0 & \dots & -0.5 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0.5 & 0 & 0 & \dots & 0 \\ 0 & \dots & 0 & -0.5 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0.5 & 0 & \dots & 0 \\ \vdots & & \vdots & & \ddots & & \vdots & \ddots & & \ddots & & \ddots & & \ddots & \vdots \\ 0 & \dots & 0 & \dots & & -0.5 & \dots & 0 & 0 & 0 & \dots & & & 0.5 & \dots & 0 \end{bmatrix}$$

if second derivatives are applied another similar $M \times M T$ block will follow and the first feature vector in \mathbf{O} for the first frame is built. This is now repeated for all T frames by shifting the $M \times M T$ blocks by M columns each frame. One realises that the blocks can be built by multiplying the elements in the template matrix w_t with an identity matrix of the size $M \times M$. These matrices can be concatenated

along the columns in the order as the elements in wt appear while leaving the rest of the $M \times MT$ block at zero. Thus each row in the template matrix builds a $M \times MT$ block. Using all f rows for the T frames will yield a W matrix of size $[fMT \times MT]$. If for example second derivatives are used (this yields 3 rows in the wt template) with 24 MFCCs each frame with a shift of 2ms in a sentence of 5 seconds length, then $f=3$, $M=24$ and $T=2500$ and thus $W=[180'000 \times 60'000]$! Eventhough W is sparse, it is not diagonal.

Iterative Estimation in *opt_c*

Once all needed matrices are built to estimate C in equation 9 there are two implementations which can be evaluated. Either the direct way to build one whole sentence HMM and estimate all parameters at once solving 9 or the iterative way where only a few parameters are estimated at each loop. Considering the above calculation of the size of W the first way only works for very small sentences, else MATALAB does not have the necessary calculation power. The iterative way is described in [6]. Function *opt_c* estimates the MFCCs given the following arguments

- W : The above discussed W matrix
- μ : The concatenated mean vector for the whole utterance
- σ : The concatenated variance vector for the whole utterance
- $feats$: The number of derivatives applied +1 (i.e. the number of rows in the template matrix wt)
- $pruning$: a pruning factor to handle very small variances and their inverse values

At the initialisation state a first choice of mixtures in the state sequence is made according to equation 16 then the mean values are read from the selected mixtures as initial MFCC solution. Note that if no dynamic features are applied this would already be the final solution.

Then the initialisation loop starts as described in [6]. The function does not perform the calculations with the whole variance and mean vector but only includes a given neighbourhood specified in the variable *Neigh*. It was shown that the influence of neighbour states more than 20 states away from the actual considered one do not influence the actual iteration step in a significant manner, thus by setting $Neigh=30$ or more calculation time is significantly reduced while the result is basically the same.

Optionally the optimisation loop can be uncommented to reestimate the mixtures

for the calculations. This loop follows the algorithm described in [6] and thus compares $\ln(P[\mathbf{O}, \mathbf{q}|\lambda])$ which is the probability of the best feature vectors and state sequence given the model. For each iteration the algorithm changes the actual mixture to the new mixture in the state where the best improvement was achieved. This is repeated until the improvement drops below a threshold defined in the variable *thres* or the maximal number of loops defined in the variable *loops* is reached.

However the initial mixture selection is fairly well so that nearly no changes were observed while running the mixture optimisation loops. Compared to the consumed calculation time it is therefore recommended to disable the mixture optimisation loop. Tests have shown that only at the beginning and end of the utterance some adaptations are made. This may also be due to boundary conditions. An effort has been made to include the boundary conditions as HTK does when calculating the feature vectors and in particular the derivatives. However, the utterances begin and end with a silence and only the first few frames are touched by the boundary conditions. Thus this is ignored. The other boundary conditions are present by selecting only a neighbourhood around the actual iteration step. Handle these conditions properly also has only a small effect as the neighbourhood should be chosen in a way that the last few neighbours anyway have only a very small influence on the actual estimated frame. By selecting *Neigh=40* boundary conditions of this kind can be ignored as well.

The function returns at the end the estimated MFCCs in a vector *c* with size [MTx1].

4.2.3 Synthesising the Audio Signal

The function *estimparam* is embedded in a function *test_mfcc_synthesis* written by H. Romsdorfer. The function is self explaining, however verify that all parameters such as the frameshift, window size, number of MFCCs per feature vector and so on are set correctly in both the *test_mfcc_synthesis* as well as on the line where *estimparam* is called! Note that this is a first step in building a synthesiser and thus the quality still can be improved. It is suggested to synthesise first the audio signal by vocoding the feature vectors directly from HTK and only then compare this result to the estimated feature vectors. By comparing the vocoded signal with the original audio file the error of the synthesiser is shown.

4.3 Results from Monophone Models

With the above training and estimation tools first steps in monophone models are made. In the following three different types of models are presented, first with only the MFCCs as feature vectors then adding the 1st derivatives and finally adding

the 2^{nd} derivatives to the vectors. For all models the MCD error measurement is applied. However, as mentioned in 3.2.1 this error measurement is sensitive to the training parameters. Thus a more general view of the different model types is given here rather than sophisticated statistics. The examples are taken from *sent_002.wav* which is the German sentence “Beteiligung von Passanten”. It contains 25 phones including the silence at the beginning and excluding the silence at the end. The sentence is 1.622 seconds long and has 811 states, for each 2ms a feature vector of 12 MFCCs plus the energy term is extracted. Models with a 5ms frameshift were trained but did not perform better. The time measurement are made on a Pentium 4 with 2GHz. The results shall not represent professional statistics, however, they give a good overview about the calculation power needed and the achieved synthesis quality. The neighbourhood in the models with the derivatives included is always set to 2 neighbours each side with reference to the $(n - 1)^{th}$ derivative. In other notation $\hat{L}_1 = \hat{L}_2 = 2$.

4.3.1 MFCCs Only

If no dynamic features such as the derivatives of the MFCCs are taken into account the estimation of the MFCCs yields the mean values of the mixtures of the selected state sequence. The estimation process is very fast, however the achieved quality is low. Figures 11 to 14 show the curve of the estimated MFCCs in red and the original MFCCs from HTK in blue. Both normalised as discussed.

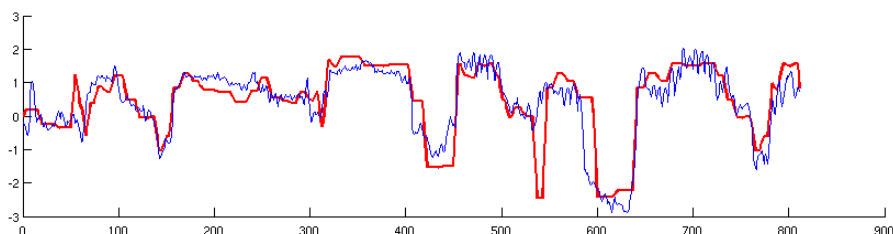


Figure 11: 1^{st} MFCC

Note that the 3 mean values of each state per phone are good visible due to the lack of dynamic features. The smoothing only results from applying the mentioned filter after estimating the MFCCs. The stair like form with the significant steps is also audible in the synthesis. However, as the quality of even the vocoded signal is relatively poor, it is difficult to hear the impurities caused by the estimation. The MCD in this example is 15.54 and is taken as reference for the other models. The estimation times for this example are the following:

- Overall: 13 seconds

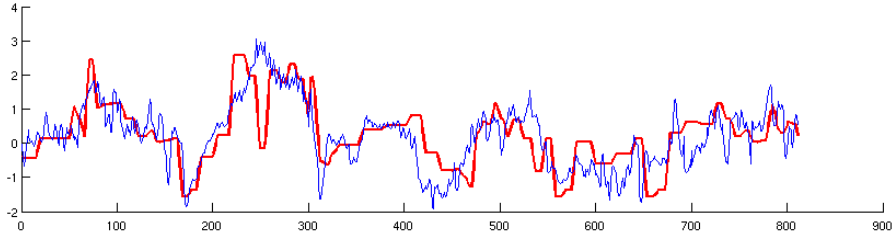


Figure 12: 3rd MFCC

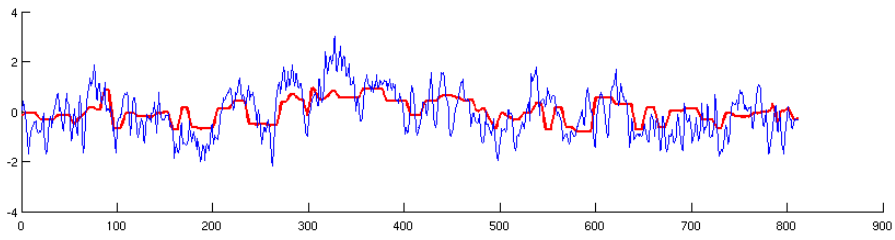


Figure 13: 11th MFCC

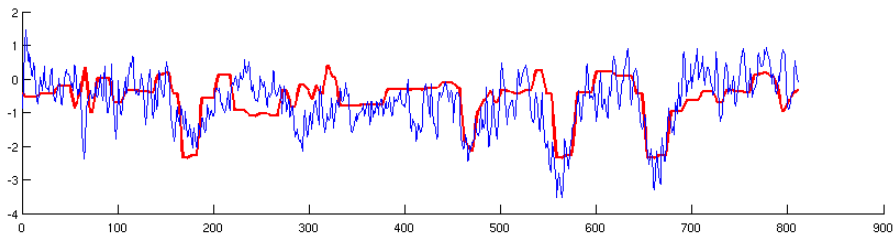


Figure 14: 12th MFCC

- Reading the sates: 8 seconds
- Building the W matrix: 3 seconds
- Estimating the parameters: 2 seconds

Thus the real estimating time is neglectable compared to the reading time.

4.3.2 MFCCs with 1st Derivatives

Including the first dynamic feature makes the steps caused by the previous model disappear. Figures 15 to 18 show the estimated and original MFCC curves for the same sentence. The models are again trained with 12 MFCCs plus the energy term per vector this time with the 1st derivatives included. All other training parameters are left unchanged.

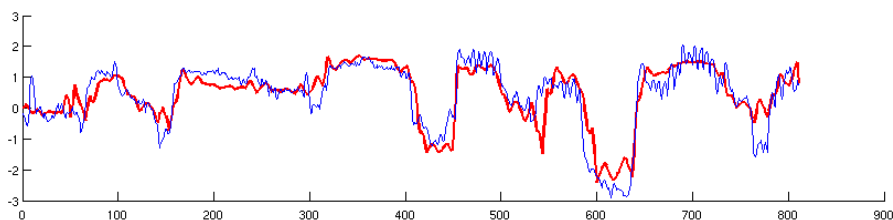


Figure 15: 1st MFCC

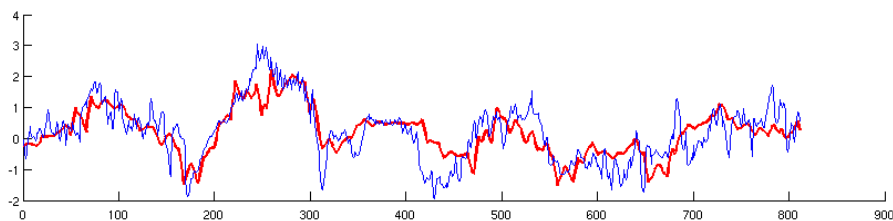


Figure 16: 3rd MFCC

The improvement is also audible in the synthesised audio file. An MCD of 14.84 is calculated which is an improvement of 4.5% compared to the above models without dynamic features. The calculation times are the following:

- Overall: 44 seconds
- Reading the sates: 9 seconds
- Building the W matrix: 11 seconds

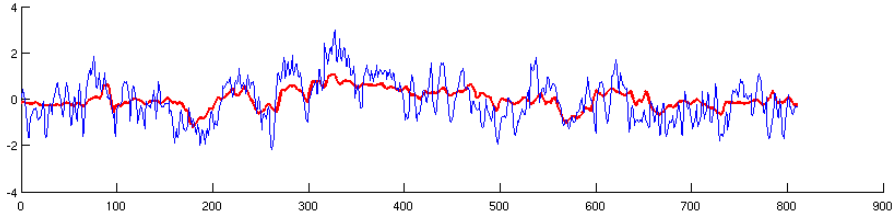


Figure 17: 11th MFCC

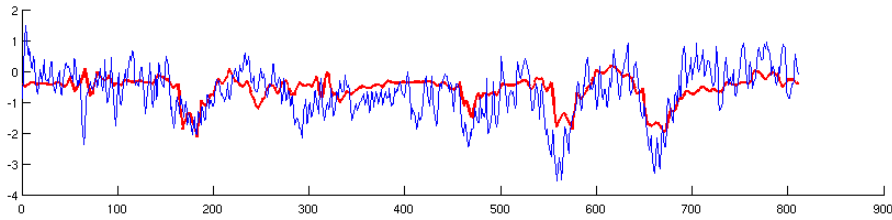


Figure 18: 12th MFCC

- Estimating the parameters: 24 seconds

It is obvious that the building of the W matrix increases significantly because of the adding of the dynamic features. W being more complex and less sparse than in the first models without derivatives, the estimation time also increases significantly. However, the improvement of quality legitimises the calculation power.

4.3.3 MFCCs with 1st Derivatives and 2nd Derivatives

Again here an additional dynamic feature is included in the models. Besides adding the second derivatives to the MFCC vector with the first derivatives no parameters have been changed. Figures 19 to 22 show the results.

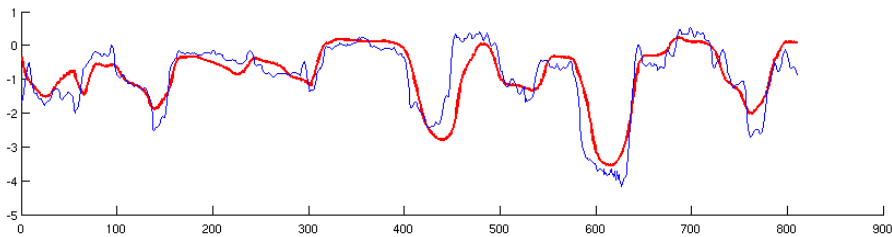


Figure 19: 1st MFCC

Observe the smoothing which is introduced by adding the second derivative. Even though the MCD is 14.65 which is a further improvement of 1.5% the quality

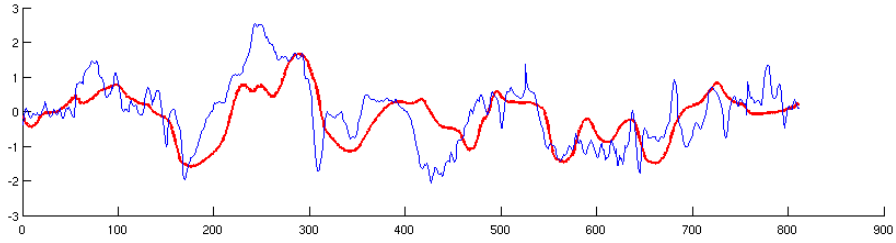


Figure 20: 3rd MFCC

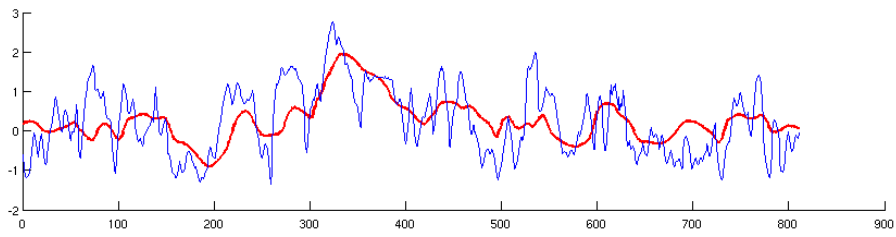


Figure 21: 11th MFCC

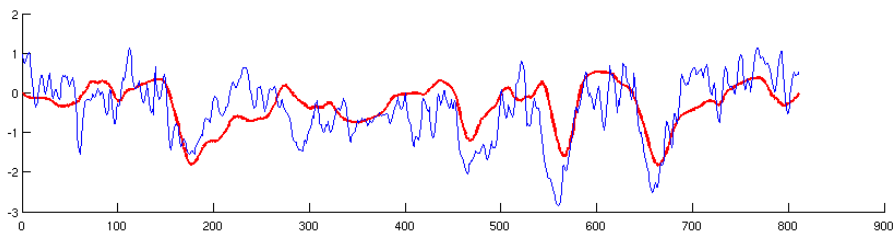


Figure 22: 12th MFCC

of the synthesised utterance is lower compared to the models with only the first derivatives included. This might be the result of the oversmoothing caused by the second derivatives. Having a look at the calculation time it is questionable if the inclusion of second derivatives leads to the desired goal:

- Overall: 115 seconds
- Reading the sates: 10 seconds
- Building the W matrix: 28 seconds
- Estimating the parameters: 77 seconds

As the W matrix gains on complexity the matrix building and parameter estimating time increase significantly.

4.3.4 Limitations

From the plots above one sees that the higher MFCCs are as expected estimated like strongly smoothed compared to the original curve. This is caused from the averaging during the training process. While different versions of the same phone have similar curves on the lower MFCCs the higher ones fluctuate more and more representing the small differences of each version. Thus lower MFCCs are trained more constant than higher ones and the averaging effect changes less the appearance. This can be taken into account by diversifying the models more and more. One solution is using triphone models. Other solutions are to implement other auxiliary features like syllable and word boundaries, intonation, accentuation and so on.

The advantage of monophone models is that the number of models is very low and handy. Thus each phone model receives an accurate training because enough occurrences of each phone are available in even small databases as the used one.

Calculation Problems

An issue which occurred from time to time are small variances in the mixtures. Because the speech database is of high quality, mixtures tend to be peaky, even more if a lot of mixtures are used. In the above estimations 8 mixtures per state are used. If a phone is recorded very constant without much of variation sharp mixtures are trained with low variance. They are highly probable to be selected according to the selection criteria discussed. Because the inverse of these values is built during the estimation process divisions by nearly zero occur which cause high peaks in the estimated MFCCs. One possibility to avoid these peaks is to clip these values to zero if a variance is too small, this is done with the *pruning*

parameter passed to the *opt_c* function. Another solution would be to reduce the number of mixtures to 4 or even 2. In general, however, this is also a hint that high quality data bases not necessarily are better for the synthesis. More important is the amount of occurrences to train the phone model accurately. This appears even more in the triphone models discussed in the following section.

5 Triphone Models

In order to have more detailed models of the different variations of the phones the context is taken into account. The most common step is to build triphone models out of the already trained monophone models. Only the left and the right phone of the actual one are considered here as context. Otherwise with a larger neighbourhood the number of models increase to the infinite and no database would be large enough to train all variations accurately. As in section 4 first the handling of the HTK tools is described before explaining the MATLAB tools to estimate the parameters and finally discussing the results.

5.1 Triphone-Model Training in HTK

At this stage it is assumed that monophone models are already trained. They will be retrained then to triphone models. If each monophone model is split in to its various triphone representations a method has to be applied to handle the number of models compared to the speech database size. Otherwise the data base does not contain sufficient occurrences to train each model which results in poorly trained models and thus low quality synthesis. Mainly two different methods are used to cluster the triphone models. The data based clustering is a method which bins all “similar” models. Where “similar” is defined by a threshold value and a distance measurement described in the HTK manual [3]. This method is very simple and straight forward. However it has the big disadvantage that only trained triphones can be synthesised. If during a synthesis a new triphone occurs no hint is given which of the trained models are closest to the new triphone and thus cause the least error. The scheme of data based clustering is shown in figure 23.

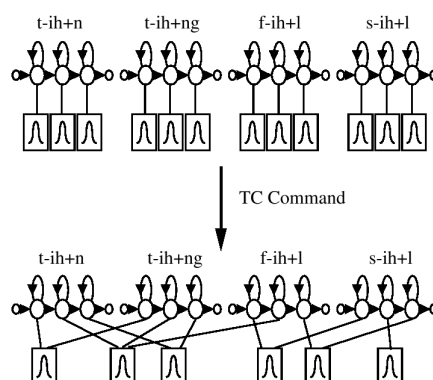


Figure 23: Data based clustering using the TC command in HTK: [3] p. 153

Conversely the tree based clustering method uses a decision tree to bin the tri-

phones. If a new triphone occurs during the synthesis stage, it can proceed the decision tree until it reaches a leaf, all models in that leaf are usable to synthesise most closely the new triphone. The disadvantage is the tree construction which has to be performed for each language. Figure 24 shows such a decision tree.

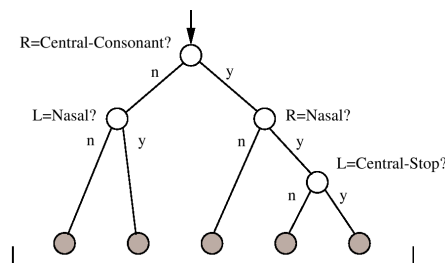


Figure 24: Decision tree:R=right side, L=left side of actual phone [3] p. 155

The training for each method is different and is explained in the following. First, however, the general steps to transcribe the monophone models to triphone models is explained

5.1.1 Building Triphone Models

The following steps have to be applied to gain the unclustered triphone models

- Untie tied monophone models in the master model file (.mmf)
- Build a triphone list by transcribing all monophones to triphones
- Tie all transition matrices of each phone in a triphone set
- Clone the monophone models to triphone models
- First training of the triphone models

Untying the Monophone Models

Up to now this step has to be done manually. It consists in untying the models which are tied according to the HMM list file which is per default *hmm.list*. Untying means simply copying the model as many times as the model is tied and renaming it according to the different tied models written in the HMM list file. An entry in *hmm.list* of

```
ph2: ph2_tied
ph9 ph2_tied
```

means thus that the model named *ph2_tied* in the .mmf file has to be copied once by renaming the original model to *ph2:* and the copied one *ph9*.

Triphone Transcription

In this step a list of all triphones occurred in the speech database is generated in the file *triphones*, moreover the list is written to the master label file *wintri.mlf* for future use in HTK. The transcription of the .tr files (i.e. the HTK label files) is performed from the \$SEGDIR/trans to the \$SEGDIR/tritrans directory by maintaining the filenames. Note that the HTK command file *mktri.led* contains

```
WB sp
WB sil
TC
```

which keep all silences in the utterances and thus builds biphone models accurately. With the *maketritrans* script

```
maketritrans traintrifiles.scp triphones wintri.mlf mktri.led $SEGDIR/trans
$SEGDIR/tritrans
```

the files are generated. The file *traintrifiles.scp* contains a list of all .tr files to be transcribed with path and filename per row.

Tie Transition Matrices

Because the transitions do not differ much from one variation to another within the same phone the transition matrices are tied together. This also allows a better training of the transition probabilities. In order to tie the matrices later with the *HHed* command of HTK a command file has to be written with the *maketrihed* script

```
maketrihed $SEGDIR/hmms_orig.list triphones
```

the HTK command file *mktri.hed* is generated. The file *hmms_orig.list* contains all untied (original) HMMs, one name per row and the file *triphones* is the file generated in the previous step being the analogue to *hmms_orig.list* but for the triphones. The file *mktri.hed* contains the tying information for the transition matrices as follows

```
CL triphones1
TI T_ph2: {(*-ph2:+*,ph2:+*,*-ph2:).transP}
TI T_ph9 {(*-ph9+*,ph9+*,*-ph9).transP}
:
```

where the CL command specifies the triphone list and the TI commands ties all triphones **-[ModelName]+**, and all biphones *[ModelName]+** and **-[ModelName]* for the phone *[ModelName]* being in the middle, on the left or right side of another phone respectively. This is repeated for all existing (mono-)phones. In the *maketrihed* script further tying such as tying all middle states etc. can be included

by simply adding the corresponding lines in the script. The following sample ties all middle states of the triphones corresponding to the phone *ph2*:

```
TI T_ph2:.2 {(*-ph2:+*,ph2:+*,*-ph2:).state[3]}
```

where the model state name *T_ph2:.2* can be arbitrarily chosen.

Cloning to Triphone Models

Once the HTK command file for the tying is written, the cloning from monophone models to triphone models can be made with

```
HHed -B -H $SEGDIR/HMM/hmmN1/hmms.mmf  
-M $SEGDIR/HMM/hmmN2 mktri.hed $SEGDIR/hmms_orig.list
```

where the monophone models are stored in *\$SEGDIR/HMM/hmmN₁/hmms.mmf* and the triphone models saved in binary format to *\$SEGDIR/HMM/hmmN₂*.

First Triphone Training

The freshly transcribed triphone models are now trained for the first time unclustered to obtain a statistic file *stats* which contains among other information the occurrences in the speech database. This will be useful for the future clustering.

```
HERest -B -t 250.0 150.0 1000.0 -s stats  
-H $SEGDIR/HMM/hmmN2/hmms.mmf -I wintri.mlf -X tr  
-L $SEGDIR/tritrans -M $SEGDIR/HMM/hmmN3  
-S trainfiles.scp triphones1
```

the models are taken from *\$SEGDIR/HMM/hmmN₂/hmms.mmf* and saved in binary format to *\$SEGDIR/HMM/hmmN₃*. Note that the file *trainfiles.scp* contains all training data, namely the feature files *.fea* extracted from the speech data base. The binary format is advisable here to maintain the master model file *.mmf* as small as possible.

5.1.2 Data Based Clustering

For the data based clustering only a few steps are needed. First a command list with the clustering commands has to be generated, then the clustering has to be applied and a re-training with the clustered models has to be performed. The command list is best generated with the *maketiehed* script

```
maketiehed $SEGDIR/hmms_orig.list stats TRO TTC trilst
```

where the *stats* file contain the statistics received from the first training of the triphone models. *T_{RO}* and *T_{TC}* are the outlier threshold for the RO command and the distance threshold for the cluster command TC. These parameters decide the size of the built clusters and must be chosen carefully. Too small clusters yield a high number of models which most probably will be poorly trained because of

missing occurrences in the speech database. Too large clusters will destroy the advantage of triphone models because differences are simply clustered together. More information about these threshold can be found in the HTK manual [3]. The output of the script is on one side the HTK command list file *mktie.hed* and the new model list *trilist*. This list has to be provided later to the MATLAB functions, for it not only contains all models but also the information about which models are tied and clustered together. The beginning of *mktie.hed* may look like

```
RO 20 "stats"
TC 300 "ph2:S2" {(*-ph2:+*,ph2:+*,*-ph2:).state[2]}
TC 300 "ph2:S3" {(*-ph2:+*,ph2:+*,*-ph2:).state[3]}
TC 300 "ph2:S4" {(*-ph2:+*,ph2:+*,*-ph2:).state[4]}
:
```

where each state of the phone *ph2:* is allowed to be clustered if its distance is less than 300.

Once having the command list file, clustering can be applied

```
HHed -B -H $SEGDIR/HMM/hmmN3/hmms.mmf
-M $SEGDIR/HMM/hmmN4 mktie.hed triphones1
```

where the models are read from *\$SEGDIR/HMM/hmmN₃/hmms.mmf* and written in binary format to *\$SEGDIR/HMM/hmmN₄*. Then re-training the models yields to the desired result. This step can also be repeated several times in order to train the models more accurately:

```
HERest -A -t 250.0 150.0 1000.0 -s stats.tri
-H $SEGDIR/HMM/hmmN4/hmms.mmf
-I wintri.mlf -X tr -L ../seg/tritrans
-M $SEGDIR/HMM/hmmN5 -S trainfiles.scp triphones1CO
```

the models are again taken from *\$SEGDIR/HMM/hmmN₄/hmms.mmf* and stored in ASCII format in *\$SEGDIR/HMM/hmmN₅*. This format is mandatory in order to apply the MATLAB function *readHMMs* to translate the models into MATLAB notation as described in 4.2.1. The new statistics about the training are stored in the *stats.tri* file. At this stage more mixtures can be included with the HTK command *HHed*.

5.1.3 Tree Based Clustering

Tree based clustering is not as straight forward as data based clustering already because this kind of clustering is only allowed to 1-mixture models in HTK. Because of this, the process described in 5.1.1 has to be applied to the corresponding models.

In this work general questions for the decision tree have been used which are applicable language independently. In [13] some suggestions are presented from which the most general were taken. Similar to the data based clustering a command list file for HTK has to be generated. This list first contains the questions and then the clustering commands similar to the *mktie.hed* file.

To build the questions the MATLAB function *buildquestions* is used. It takes as arguments

- ROThres: The threshold for the RO command similar to the data based clustering
- Statsfile: The path and name of the statistic file provided from the first training of the triphone models with one mixture

It then generates the questionnaire in *mktree.hed* following the structure

```
QS "L_Name" {affirmative models}
QS "R_Name" {affirmative models}
```

where R_ and L_ name stands for the question considering the right and left side respectively of the actual phone. The name can be chosen arbitrarily. The affirmative models are those models which can answer the question with "yes". For example the question if the left phone is an "i" or an "e" would be

```
QS "L.ie" { phi-*, phe-* }
```

the script automatically generates an left and right question for each question entered in the code.

The next step is to add the cluster commands to the file. The script *maketreehed* appends these commands to the *mktree.hed* file built previously with MATLAB. The script is invoked as follows

```
maketreehed $SEGDIR/hmms_orig.list  $T_{TB}$  trilist
```

where the arguments are basically the same as in *maketiehed* in the data based clustering except that the statistic file and the threshold value for the RO command already were given to the MATLAB function *buildquestions*. Thus only the distance threshold for the tree clustering command TB together with the original monophone list *hmms_orig.list* and the new clustered list *trilist* as output have to be passed as arguments. Here again experience and several tests reveal the best cluster size thresholds. After running the *maketreehed* script the HTK command list file *mktree.hed* is complete and clustering can be applied.

Applying the clustering is done similar to the databased clustering with

```
HHEd -B -H $SEGDIR/HMM/hmm $N_3$ /hmms.mmf
-M $SEGDIR/HMM/hmm $N_4$  mktree.hed triphones
```

where the models are taken from \$SEGDIR/HMM/hmm N_3 /hmms.mmf and stored in binary format in \$SEGDIR/HMM/hmm N_4 . Remember that the file *triphones* is the triphone model list generated in 5.1.1.

The training is then performed with

```
HERest -A -t 250.0 150.0 1000.0 -s stats.tri
-H $SEGDIR/HMM/hmmN4/hmms.mmf -I wintri.mlf
-X tr -L $SEGDIR/tritrans -M $SEGDIR/HMM/hmmN5
-S trainfiles.scp trilst
```

By saving the models in ASCII format they can be extracted to MATLAB notation using the MATLAB function *readHMMs* described in 4.2.1. The new statistics about the training are again stored in the *stats.tri* file.

5.2 Parameter Estimation

For the parameter estimation of triphones the MATLAB function *estimparamTri* is used. It is basically identical to the *estimparam* function. However, it builds internally the triphones from the monophones read in the label file *.lab*. This feature takes also silences into account and builds biphones where necessary. The rest is equivalent to the *estimparam* function described in 4.2.2

5.3 Results from Triphone Models

During the training it was realized that the database used in this work is too small to apply accurate triphone models. Both data based clustering as well as tree based clustering performed similarly. This is also because the full advantage of tree based clustering was not needed, for training sentences were taken for the synthesis. These sentences obviously contain no new triphones and thus the data clustered models always had a model at hand to be synthesised. The lowest Mel Cepstral Distortion MCD achieved was 17.2 which still is higher than with only MFCC feature vectors in monophone models. Given this it is assumed that with the perfect cluster size a value below the MCD with monophone models and second derivatives applied can be achieved. However the search of that perfect cluster size is a time consuming procedure and is recommended to do with a larger database where triphone models are more appropriate. This kind of search is done for example in [11]. In figure 25 the dependence of occurrences for training the model and accuracy in estimating the parameter is depicted.

The plot shows the initial frames of the sentence *sent_002.wav* “Beteiligung von Passanten”. The green square represents the triphone model *ph_j-pht+pha* in “Beteiligung” which occurs 107 times in the data base and thus estimates the parameter fairly well. The model *phE-ph_j+pht* right before only occurs 38 times and already shows weaknesses. However the most important ones are those which only occur under 10 times and thus cannot be trained accurately at all. Such an example is the model *phi-phl+phI* with 4 occurrences. Figures 26 to 29 show

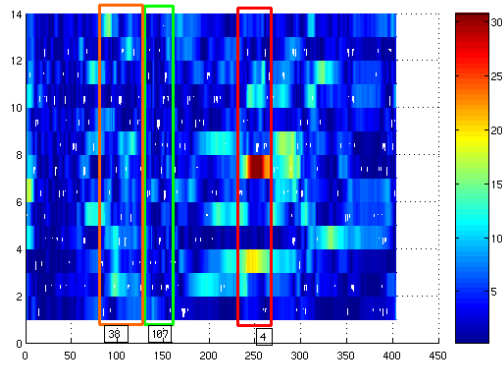


Figure 25: MCD per MFCC (y) and time frame (x)

entire MFCC estimation over the sentence also used in 4.3. The TC and TB command threshold was set to 500 and the RO command threshold to 30. All other parameter are equal to the ones in 4.3.

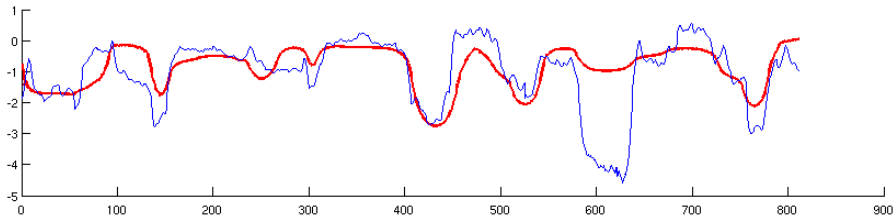


Figure 26: 1st MFCC

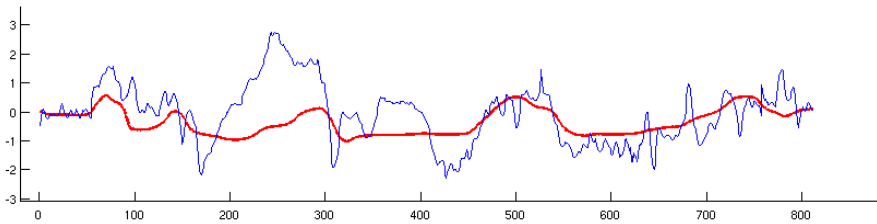


Figure 27: 3rd MFCC

5.4 Future Considerations

In order to provide a good modelling for the middle part of the phone which is probably not very context dependent the middle states could be tied together.

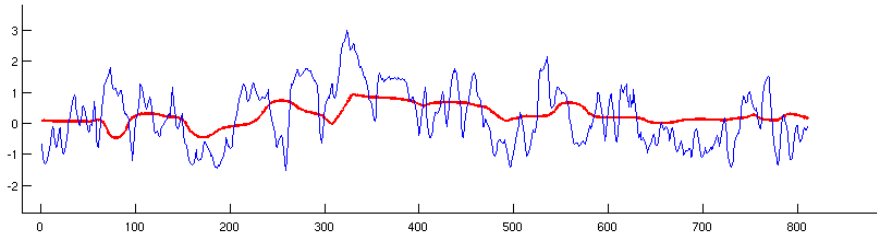


Figure 28: 11th MFCC

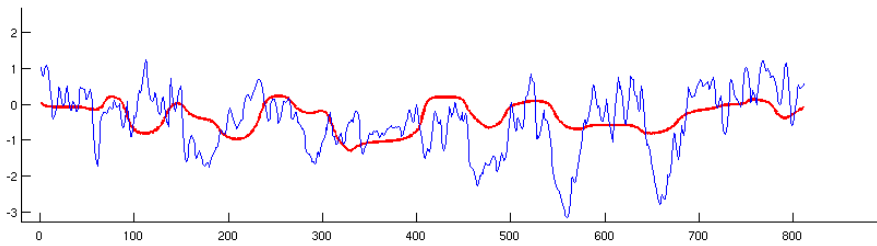


Figure 29: 12th MFCC

Leaving the outer states free results in diversifying the transitions between two phones.

Moreover to verify the difference between the data based and tree based clustering utterances with new triphones should be tried to synthesise. In this case, however, the MATLAB function *estimparamTri* has to be adapted to proceed the decision tree upon new triphones. Currently an error message is printed if a triphone is not found in the HMM list.

6 Conclusion and Outlook

6.1 Conclusion

With this work a solid base for speech synthesis with Hidden Markov Models is constructed. First steps are made and fairly well results are achieved. The results show that dynamic features are a must to obtain good synthesis quality. However doubts are shown in using as well the second derivatives as dynamic features. The calculation power needed is increased significantly compared to the gained improvement. Thus only using the first derivatives as dynamic features might be sufficient for the synthesis.

A calculation problem of divisions by zero which occur with very narrow mixtures (i.e. small variances) shows that more mixtures not necessarily increase the synthesis quality. Using less mixtures yield to a broader distribution and thus could eliminate the problem of small variances. In the same context falls the database size and its quality. If the database has a lower quality the models are trained less constantly and thus the mixtures will be broader. Moreover this together with the result from the triphone models lets assume that the size of the data base is much more important than its quality.

On the audio synthesis side it is difficult to measure the quality because of additional errors are introduced by the synthesiser itself. However, the estimated MFCCs are found to reproduce very similar audio signals compared to the original MFCCs. Thus future work should be concentrated on the synthesiser to be able to hear more accurately the quality differences between the trained models.

6.2 Outlook

The next step to this work is undoubtedly the improvement of triphone models by using a larger database. As mentioned in the conclusion the database could be of lower quality if only it provides enough data to train the models accurately. To find the best cluster size it is recommended that an automated setup is written which looks for the performance curve demonstrated in [11]. Beginning with a small clustersize a reinforcement learning algorithm can be applied. Each time the MCD on the test set is lower than before the step by which the clustersize is increased, is augmented as well until the MCD is higher, then the stepsize is adjusted accordingly. Figure 30 shows a possible search path.

Another step, given the data base is large enough, is to introduce even more features to refine the models. In particular phonological features can help to model different variations of a phone. The features are discussed in 3.1. Note that features trained with the MFCCs do generally not improve the quality because the dependency between the MFCCs and the feature is normally not known. Thus the

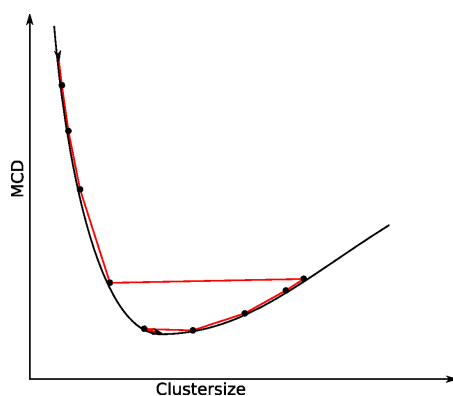


Figure 30: MCD compared to cluster size: from [11]

feature is trained and estimated separately from the MFCCs. Only the derivatives of the MFCCs which stand in direct relation with the coefficients are usefull to train with the models. The phonological features are thus to be added as the context dependency by splitting existing models into new and more detailed models. Examples of phonological features can be sillable or word boundaries, accentuation, in general all prosodic attributes. Including the phrase type.

Once having achieved a fairly good synthesis, speaker adaption can be attacked. The idea is to train HMMs with a database of one speaker. Then by only having a few utterance from another speaker the models are re-trained to sound like the new speaker. First steps in this direction were made in [4].

7 Bibliography

References

- [1] Beat Pfister, René Beutler, “Sprachverarbeitung I, Skript zur Vorlesung”, ETH Zurich, Switzerland
- [2] Beat Pfister, René Beutler, “Sprachverarbeitung II, Skript zur Vorlesung”, ETH Zurich, Switzerland
- [3] Steve Young, Gunnar Evermann, Mark Gales, Thomas Hain, Dan Kershaw, Xunying (Andrew) Liu, Gareth Moore, Julian Odell, Dave Ollason, Dan Povey, Valtcho Valtchev, Phil Woodland, “The HTK Book (for HTK Version 3.4)”, 2006
- [4] Junichi Yamagishi, “An Introduction to HMM-Based Speech Synthesis”, 2006
- [5] Keiichi Tokuda, Takayoshi Yoshimura, Takashi Masuko, Takao Kobayashi, Tadashi Kitamura, “Speech Parameter Generation Algorithm For HMM-Based Speech Synthesis”, Nagoya Institute of Technology, Tokyo Institute of Technology, Japan
- [6] Keiichi Tokuda, Takashi Masuko, Tetsuya Yamada, Takao Kobayashi, Satoshi Imai, “An Algorithm For Speech Parameter Generation From Continuous Mixture HMMs With Dynamic Features”, Tokyo Institute of Technology, Japan
- [7] Keiichi Tokuda, Takashi Masuko, Jun Hiroi, Takao Kobayashi, Tadashi Kitamura, “A Very Low Bit Rate Speech Coder Using HMM-Based Speech Recognition/Synthesis Techniques”, Nagoya Institute of Technology, Tokyo Institute of Technology, Japan
- [8] Toshiaki Fukada, Keiichi Tokuda, Satoshi Imai, Takao Kobayashi, Tadashi Kitamura, “An Adaptive Algorithm For Mel-Cepstral Analysis Of Speech”, Tokyo Institute of Technology, Japan
- [9] Takashi Masuko, Keiichi Tokuda, Takao Kobayashi, Satoshi Imai, “Speech Synthesis Using HMMs With Dynamic Features”, Tokyo Institute of Technology, Japan
- [10] Keiichi Tokuda, Takao Kobayashi, Satoshi Imai, “Speech Parameter Generation From HMM Using Dynamic Features”, Tokyo Institute of Technology, Japan

- [11] Alan W. Black, “CLUSTERGEN: A Statistical Parametric Synthesiser Using Trajectory Modeling”, Carnegie Mellon University, USA
- [12] Alan W. Black, Tanja Schulz, “Speaker Clustering For Multilingual Synthesis”, Carnegie Mellon University, USA, 2006
- [13] Julian James Odell, “The Use Of Context In Large Vocabulary Speech Recognition”, Queen’s College, UK, 1995
- [14] Harald Romsdorfer, Beat Pster, “Phonetic Labeling and Segmentation of Mixed-Lingual Prosody Databases”, ETH Zurich, Switzerland, 2005

Sommersemester 2007
(SA-2007-46)

Semesterarbeitsaufgabenstellung

für

Herr Stephan Weiss

Betreuer: H. Romsdorfer ETZ D97.5
Stellvertreter: Dr. B. Pfister ETZ D97.6

Ausgabe: 18. April 2007
Abgabe: 13. Juli 2007

Sprachsynthese mit Hidden-Markov-Modellen

Einleitung

Eine neuere Methode zur Synthese von Sprachsignalen basiert auf kontextabhängigen Laut-HMM (Hidden-Markov-Modelle). Diese Methode ist attraktiv, weil sie grundsätzlich die Möglichkeit bietet, die stimmlichen Eigenschaften der Sprachsynthese auf eine neue Stimme zu adaptieren, also eine Art der Stimmtransformation durchzuführen.

Aufgabenstellung

In dieser Arbeit soll unter Verwendung eines bestehenden Audiokorpus von phonologisch annotierten, deutschen Sätzen ein Prototyp einer HMM-basierten Sprachsynthese entwickelt werden. Dazu sind im wesentlichen folgende Schritte notwendig:

- Training der HMM-Laut-Modelle anhand des deutschen Audiokorpus. Ein Tool zum Training dieser Modelle wird zur Verfügung gestellt.
- Implementierung eines Algorithmus zur HMM-Parametergeneration in Matlab. Ein Algorithmus ist in [1, 2, 3] beschrieben.

- Synthese des Sprachsignals unter Verwendung der ermittelten HMM-Modelle ([4]). Eine Funktion zur Synthese des Sprachsignals aus einer Sequenz von HMM-Parametern und entsprechenden HMM-Laut-Modellen wird zur Verfügung gestellt.
- Implementierung eines Verfahrens zur Qualitätsbeurteilung der erzeugten HMM-Parametersequenz. Ein mögliches Fehlermass wird in [5] präsentiert.
- Experimente zur Qualitätsverbesserung der Sprachsynthese durch Optimierung der HMM-Laut-Modelle und der HMM-Parametergeneration. Dazu müssen Experimente mit verschiedenen Konfigurationen von HMM-Eingangsgrößen, wie verschiedene MFCCs (Mel Frequency Cepstral Coefficients), phonologische Merkmale und phonetischer Lautkontext (siehe z.B. [6, 5]), durchgeführt werden.

Die ausgeführten Arbeiten und die erhaltenen Resultate sind in einem Bericht zu dokumentieren (siehe dazu [7]), der in gedruckter und in elektronischer Form abzugeben ist. Zusätzlich sind im Rahmen eines Kolloquiums zwei Präsentationen vorgesehen: etwa drei Wochen nach Beginn soll der Arbeitsplan und am Ende der Arbeit die Resultate vorgestellt werden. Die Termine werden später bekannt gegeben.

Literaturverzeichnis

- [1] K. Tokuda, H. Matsumura, T. Kobayashi, and S. Imai. Speech coding based on adaptive mel-cepstral analysis. In *Proceedings of the ICASSP 1994*, pages 197–200, Adelaide, Australia, April 1994.
- [2] K. Tokuda, T. Yoshimura, T. Masuko, T. Kobayashi, and T. Kitamura. Speech parameter generation algorithms for HMM-based speech synthesis. In *Proceedings of the ICASSP 2000*, pages 1315–1318, Istanbul, Turkey, June 2000.
- [3] J. Yamagishi. An introduction to HMM-based speech synthesis. Technical report, Tokyo Institute of Technology, October 2006.
- [4] S. Imai. Cepstral analysis synthesis on the mel frequency scale. In *Proceedings of ICASSP 83*, pages 93–96, Boston, USA, April 1983.
- [5] Alan W. Black. CLUSTERGEN: A statistical parametric synthesizer using trajectory modeling. In *Proceedings of Interspeech 2006*, pages 1762–1765, Pittsburgh, Pennsylvania, September 2006.
- [6] Alan W. Black and Tanja Schultz. Speaker clustering for multilingual synthesis. In *ISCA Tutorial and Research Workshop on Multilingual Speech and Language Processing (MultiLing 2006)*, Stellenbosch, South Africa, April 2006.
- [7] B. Pfister. *Richtlinien für das Verfassen des Berichtes zu einer Semester- oder Diplomarbeit*. Institut TIK, ETH Zürich, März 2004. (http://www.tik.ee.ethz.ch/~spr/SADA/richtlinien_bericht.pdf).

- [8] B. Pfister. *Hinweise für die Präsentation der Semester- oder Diplomarbeit*. Institut TIK, ETH Zürich, März 2004.
(http://www.tik.ee.ethz.ch/~spr/SADA/hinweise_praesentation.pdf).

Zürich, den 18. April 2007