

Ein Hidden-Markov-Modell (HMM) basiertes Operngesangssynthesystem für Deutsch

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur/in

im Rahmen des Studiums

Computational Intelligence

eingereicht von

Mag.phil. Dr.techn. Michael Pucher

Matrikelnummer 9209069

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer/in: Univ.Prof. Dipl.-Inf. Dr.rer.nat. Jens Knoop

Wien, 25.03.2015

(Unterschrift Verfasser/in)

(Unterschrift Betreuer/in)

A Hidden-Markov-Model (HMM) based Opera Singing Synthesis System for German

MASTER THESIS

for obtaining the academic degree

Master of Science

within the study program

Computational Intelligence

submitted by

Mag.phil. Dr.techn. Michael Pucher

Matriculation number 9209069

at the
Faculty of Informatics of the Vienna University of Technology

Supervision
Supervisor: Univ.Prof. Dipl.-Inf. Dr.rer.nat. Jens Knoop

Vienna, 25.03.2015

(Signature author)

(Signature supervisor)

Michael Pucher
1030 Wien, Schrottgasse 6/17

„Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.“

Wien, 25.03.2015

Acknowledgments

Initial work for this thesis was done at a stay at National Institute of Informatics (NII) in Japan in 2014, which was funded by NII and the Austrian Science Fund (FWF) project P23821-N23. The recording of opera songs was funded by NII.

Kurzfassung

In dieser Diplomarbeit wird ein Hidden-Markov-Modell (HMM) basiertes Operngesangssynthesystem für Deutsch entwickelt, das auf einem japanischen Gesangssynthesystem für Popsongs basiert. Die Entwicklung besteht aus der Integration einer deutschen Textanalyse, eines Lexikons mit Graphem-zu-Phonem Übersetzung, und eines Silbenvervielfältigungsalgorithmus. Außerdem werden synthetische Opernstimmen der vier wichtigsten Sängerkategorien Mezzo, Sopran, Tenor, und Bass entwickelt und die Methode mit der der Korpus erstellt wurde wird beschrieben. Darüber hinaus wird eine Methode entwickelt um die vorhandenen Daten (Waveforms und MusicXML Dateien) in ein für das Training der Modelle geeignetes Format umzuwandeln. Für das Training wird eine SängerInnenabhängige Methode für das Deutsche adaptiert. In einer objektiven und subjektiven Evaluation werden verschiedene Parameterkonfigurationen für das Training und die Synthese evaluiert. Mit der subjektiven Evaluation wird gezeigt dass Operngesangssynthese von moderater Qualität mit diesem System und den begrenzten vorhandenen Trainingsdaten möglich ist, und dass die Dauermodellierung der wichtigste Qualitätsparameter der Modelle ist. Für ein Synthesystem von hoher Qualität sind mehr Trainingsdaten notwendig, da bekannt ist dass die verwendeten Lernalgorithmen bessere Ergebnisse mit mehr Daten liefern. Das derzeitige System bildet die Basis für so ein zukünftiges System und kann auch für ein allgemeines Gesangssynthesystem verwendet werden. Vor dieser Arbeit war ein derartiges Gesangssynthesystem basierend auf HMMs nur für Japanisch und Englisch verfügbar.

Abstract

In this thesis we develop a Hidden Markov Model (HMM) based opera singing synthesis system for German that is based on a Japanese singing synthesis system for popular songs. The implementation of this system consists of an integration of German text analysis, lexicon and Letter To Sound (LTS) conversion, and syllable duplication. We also develop opera singing voices for the main four singer categories mezzo, soprano, tenor, and bass and describe the recording method that was used to record opera singers to acquire the data that is used for modeling. These voices can be used for opera singing synthesis and automatic alignment of singing. Furthermore we develop an alignment method that is used to transform the available data (waveforms, Music Extended Markup Language (MusicXML) files) into a format suitable for training the voices. For the training itself we adapt a singer-dependent training procedure to German. Finally we present an objective and subjective evaluation of the mezzo voice where effects of different parameter configurations during training and synthesis are evaluated. With the subjective evaluation we can show that moderate quality opera singing synthesis is feasible with the limited amount of training data at hand and that correct duration modeling is the most influential quality parameter at this stage. For a high quality opera singing synthesis system we would need more training data as it is known that the quality of the models increases with larger amounts of data. The current system provides the basis for such a future high quality system, and can also be used as a front-end for a general German singing synthesis system. Before our work such an HMM-based singing synthesis system was only available for Japanese and English.

Contents

1	Introduction	1
2	State-of-the-art	5
2.1	Hidden Markov Model (HMM)	5
2.1.1	Discrete Density Hidden Markov Models (DDHMM)	5
2.1.2	Continuous Density Hidden Markov Models (CDHMM)	7
2.1.3	Basic problems for hidden Markov models (HMMs)	9
2.2	Speech synthesis	10
2.2.1	Applications	10
2.2.2	Text-To-Speech synthesis (TTS)	10
2.2.3	Text analysis	11
2.2.4	Grapheme To Phoneme (G2P) conversion	12
2.2.5	Unit selection speech synthesis	14
2.3	Hidden Markov Model (HMM) based speech synthesis	15
2.3.1	Speaker dependent HMM based speech synthesis system	17
2.3.2	Context clustering	17
2.3.3	Duration modeling	21
2.3.4	Parameter generation	21
2.3.5	Hybrid systems	24
2.4	MusicXML	25
2.5	Singing synthesis	27
2.5.1	Articulatory synthesis of singing	27
2.5.2	Conversion from speaking voice to singing voice	28
2.5.3	Formant based synthesis of singing	29
2.5.4	Diphone based singing synthesis system VOCALOID	29
2.6	Hidden-Markov-Model (HMM) based singing synthesis	30
2.6.1	Context and time-lag modeling	31
2.6.2	Rich context modeling, vibrato modeling, and Fundamental Frequency (F0)-shifting	31
2.6.3	Adaptive F0 modeling	32
2.6.4	Syllable allocation and duplication	32
2.6.5	Vocoding	34
2.6.6	HMM-based SINGing voice SYNthesis system (SINSY)	35
2.7	Musical Instrument Digital Interface (MIDI)	36
2.8	Alignment of MIDI data and audio	36

3	A Hidden-Markov-Model (HMM) based opera singing synthesis system for German	41
3.1	Recording	41
3.1.1	Singer and song selection	41
3.1.2	Phonetically balanced singing corpus	43
3.2	Implementation of a German frontend for Sinsy	44
3.2.1	Text analysis	44
3.2.2	Lexicon and letter-to-sound conversion	44
3.2.3	Syllable duplication	45
3.3	Alignment	48
3.3.1	Conversion between notes, midi notes, and frequencies	48
3.3.2	Aligning waveforms and midi data	49
3.3.3	Splitting opera recordings into utterances	49
3.3.4	Alignment of singing speech and labels	51
3.4	Training of acoustic models	51
3.4.1	Data	51
3.4.2	Training	51
3.4.3	F0 extraction methods	54
3.5	Voice development pipeline	56
4	Evaluation	59
4.1	Different mezzo voices for evaluation	59
4.2	Objective evaluation metric	60
4.3	Results of objective evaluation	60
4.4	Subjective evaluation	63
4.5	Results of subjective evaluation	63
4.6	Analysis	65
5	Conclusion	69
6	Future work	71
	Bibliography	73
A	Context dependent label format	79

List of Abbreviations

AI	Artificial Intelligence
AMTV	Acoustic Modeling and Transformation of Language Varieties
CDHMM	Continuous Density Hidden Markov Models
DDHMM	Discrete Density Hidden Markov Models
DTW	Dynamic Time Warping
EM	Expectation Maximization
F0	Fundamental Frequency
FFT	Fast Fourier Transform
FSA	Finite State Automaton
FST	Finite State Transducer
FTW	Telecommunications Research Center Vienna
FWF	Austrian Science Fund
G2P	Grapheme To Phoneme
GMM	Gaussian Mixture Model
HMM	Hidden Markov Model
HTS	HMM-based Speech Synthesis System (H Triple-S)
LSP	Line Spectral Pair
LTS	Letter To Sound
MCD	Mel Cepstral Distortion
MLF	Master Label File
MDL	Minimum Description Length
MFCC	Mel Frequency Cepstral Coefficients

MGC Mel Generalized Cepstral Coefficients
MIDI Musical Instrument Digital Interface
ML Maximum Likelihood
MRI Magnetic Resonance Image
MusicXML Music Extended Markup Language
NII National Institute of Informatics
PDF Probability Density Function
RAPT Robust Algorithm for Pitch Tracking
MSE Mean Squared Error
SAMPA Speech Assessment Methods Phonetic Alphabet
SINSY HMM-based SINGing voice SYnthesis system
SPTK Signal Processing ToolKit
TTS Text-To-Speech synthesis
WER Word Error Rate
XML eXtended Markup Language

List of Tables

2.1	Diphthong duplication rules (Table from [44]).	33
2.2	Example Dynamic Time Warping (DTW) alignment between two sequences. . . .	39
3.1	German diphthong duplication rules.	47
3.2	<i>Alignment methods for aligning original recordings with MIDI files.</i> . . .	49
3.3	Songs recorded for the mezzo voice. The table also shows the maximum and minimum F0 according to the MusicXML file.	52
4.1	Different parameters for the evaluation used in training.	59
4.2	Different parameters for the evaluation used in synthesis.	59
4.3	8 methods used in the subjective evaluation.	63
4.4	Two methods resulting in best and worst synthesis according to Mel Cepstral Distortion (MCD).	66
A.1	Description of features.	82

List of Figures

2.1	Discrete Density Hidden Markov Models (DDHMM).	6
2.2	Normal distribution.	8
2.3	Continuous Density Hidden Markov Models (CDHMM).	9
2.4	Finite state transducer (FST) F for cardinals (numeric-to-written).	12
2.5	Finite state transducer (FST) F^{-1} for cardinals (written-to-numeric).	12
2.6	Decision trees for G2P conversion of Standard German.	13
2.7	Diphone unit graph for the Viennese word “nein” [n a:].	15
2.8	Five state HMM for the phone y: with Gaussian mixture observation Probability Density Function (PDF).	16
2.9	Re-usage of data with sub-word HMMs (below) as compared to word HMMs (above).	16
2.10	Speaker dependent HMM-based speech synthesis system (Figure redrawn from [25]).	17
2.11	Data-driven state tying for [l] quinphones.	18
2.12	Decision-tree based state tying.	19
2.13	Part of decision-tree for Mel-cepstrum of 3rd state (central state in 5-state HMM) for variety independent / speaker dependent model with full feature set.	20
2.14	Topology for implicit (above) and explicit (below) state duration. Geometric distribution (right).	21
2.15	Duration synthesis.	23
2.16	Hybrid speech synthesis system where speech generated from HMMs is used in the target cost function.	24
2.17	Part of the musical score from a Latin song (Figure from [1]).	25
2.18	From gestural scores to trajectories in the articulatory synthesizer (Figure from [33]).	27
2.19	Vocal conversion system diagram (Figure from [34]).	28
2.20	Formant singing synthesis system (Figure from [38]).	29
2.21	Basic speaker dependent HMM-based singing synthesis system (Figure redrawn after [40]).	30
2.22	Two different syllable allocation methods (Figure from [44]).	33
2.23	Syllable duplication (Figure from [44]).	33
2.24	Inheritance diagram for <code>sinsy::IConf</code> .	35
2.25	Masking of MIDI file (top). DTW alignment (bottom) (Figure from [56]).	37
2.26	Different pattern for computing the cost in DTW.	38
3.1	Classification of opera songs according to lyrical - dramatical and slow - fast dimension.	42
3.2	F0 range for mezzo and opera songs shown on the piano roll.	43

3.3	Alignment of MIDI and phone labels on utterance level for the utterance “Wenn mein Schatz Hochzeit macht”.	45
3.4	Alignment of original mezzo Song #1 with MIDI song.	50
3.5	Part of the decision tree for log F0 models for the center state (4th state of 7-state) of the HMM.	53
3.6	Part of the decision tree for spectral models for the center state (4th state of 7-state) of the HMM.	54
3.7	Part of the decision tree for duration models for the center state (4th state of 7-state) of the HMM.	55
3.8	Voice development pipeline.	58
4.1	Cepstral distortion per sentence (left), normalized cepstral distortion for FFT length (right).	60
4.2	Normalized cepstral distortion for synthesis durations (left), normalized cepstral distortion for F0 extraction method (right).	61
4.3	Normalized cepstral distortion for F0 expansion (left), normalized cepstral distortion for training data alignment method (right).	62
4.4	Normalized cepstral distortion for the 16 different methods (training/synthesis condition combinations).	63
4.5	Results of subjective experiments for different Fast Fourier Transform (FFT) length (left), and different synthesis durations (right).	64
4.6	Results of subjective experiments for different F0 extraction method (left), and F0 expansion (right).	65
4.7	Results of subjective experiments for different training methods.	65
4.8	Alignment of MIDI and phone labels on utterance level for the utterance “Seh ich zwei blaue Augen stehn”. Original (top), synthesized (bottom).	67
4.9	Alignment of MIDI and phone labels on utterance level for the utterance “Sagt, holde Frauen die ihr sie kennt”. Original (top), synthesized (bottom).	68

1 Introduction

By singing synthesis we refer to the task of generating an acoustic signal of a singing person. The synthesis output is thereby controlled by a text input and a musical score that is aligned with the text. The textual data is also divided into syllabic sequences. This input data can be given as Music Extended Markup Language (MusicXML) [1] file.

In a corpus-based approach machine learning methods are used to learn a singing model from pre-recorded singing data. A general problem for corpus-based approaches to singing synthesis is the modeling of contexts that are not covered in the training data. An additional problem is the alignment of singing speech and Fundamental Frequency (F0) that has to be done in a natural way that does not exactly follow the musical notation.

Opera singing synthesis poses additional modeling problems due to the large variation in F0. A further challenge is the modeling of duration that can vary significantly between and within different opera pieces.

One main result of this thesis will be acoustic models for Hidden Markov Model (HMM)-based opera singing synthesis. All modeling will be based on a German opera singing corpus that was recorded within the Acoustic Modeling and Transformation of Language Varieties (AMTV) [2] research project. This corpus contains recordings of several opera pieces for each voice type (mezzo, soprano, bass, and tenor) as well as a phonetically balanced corpus of opera singing. For this thesis we will build voices for all corpora and evaluate different models of the mezzo corpus. For the acoustic models we will only use the recorded opera songs, since no MusicXML transcription is available for the phonetically balanced corpus.

The acoustic models will be integrated into an existing open-source singing synthesis system [3], which currently only supports Japanese. The singing synthesis system will get a musical score and German text as input in the form of a MusicXML file and produce an acoustic opera performance as output. The system is based on the HMM-based Speech Synthesis System (H Triple-S) (HTS) [4].

The main results of this thesis are:

- Extension of an existing HMM-based singing synthesis system [3] with a module for German.
- Automatic alignment of singing speech and music for an existing opera singing corpus.
- Acoustic model training for opera singing synthesis.

- Objective and subjective evaluation of opera singing synthesis.

This system will allow users to synthesize any type of German singing speech (including operas) with state-of-the-art HMM-based synthesis techniques. With this system we will also have the framework and development pipeline that allows us to quickly create new German singing synthesis voices from given recordings.

For this thesis we will use statistical modeling methods for parametric speech synthesis based on HMMs. For acoustic model training HMM states are clustered with decision trees where separate trees are estimated for F0, spectrum, and duration. The clustered models are optimized according to the Maximum Likelihood (ML) criterion. The questions used in the clustering cover wide contexts such as current, previous and following phones, notes for current, previous, and following phones, and syllable and word contexts. Appropriate clustering questions for German will be designed, that contain German phones, syllable stress, and word information that is not required for Japanese.

For the evaluation of the developed methods we will use standard objective evaluation metrics. In the evaluation we will measure the spectral distortion between original opera singing performances and synthesized performances using MCD as metric. For spectral distortion the original opera singing performances and synthesized performances are aligned, and then the spectral difference between the aligned frames is measured. Spectral distortion will be used to evaluate different F0 extraction methods for training. It will show the influence of F0 extraction on the overall synthesis quality, as well as the spectral difference between synthesized and original samples. We will select a set of 8 test sentences per singer that are not used for training that will be used in the evaluation.

Additionally to the objective evaluation we will also perform a subjective evaluation of synthesis methods where listeners have to make preference ratings for synthesis samples that are generated using different methods. Subjective evaluation methods are state-of-the-art in the evaluation of speech synthesis systems, since they allow to find quality differences that are not found by objective metrics [5].

The thesis is structured as follows: In Chapter 2 we will introduce the state-of-the-art technologies that are necessary for an HMM-based singing synthesis system. Therefor we will introduce HMMs in general in Section 2.1, followed by an introduction of speech synthesis in general (Section 2.2), and HMM-based speech synthesis in particular (Section 2.3). After explaining the MusicXML format in Section 2.4 we will introduce singing synthesis in general (Section 2.5) and HMM-based singing synthesis in particular (Section 2.6). Finally we will introduce the MIDI format in Section 2.7 and describe how we align MIDI sequences and audio data (Section 2.8).

In Chapter 3 we will explain the process of developing the German opera synthesizer and voices. In Section 3.1 we describe the recording and corpora development process. Section 3.2 shows how we extended the existing HMM-based singing synthesis

system [3] with a module for German. Section 3.3 describes the alignment process in detail. Finally we describe the acoustic model training in Section 3.4 and show the whole development pipeline in Section 3.5.

Chapter 4 presents the objective and subjective evaluation results for the different mezzo voices. In Section 4.1 we describe the parameters that are used for defining different voices. Section 4.2 defines the objective evaluation metric that is used for objective evaluation in Section 4.3. Section 4.4 defines the subjective evaluation method that is used for subjective evaluation in Section 4.5. In Section 4.6 we analyze the evaluation results. Chapter 5 concludes the thesis.

2 State-of-the-art

2.1 Hidden Markov Model (HMM)

Hidden Markov Model (HMM)s are well known models for modeling of time-series and are used in speech recognition since many years [6, 7]. More recently HMMs are also used in speech synthesis [8, 9]. We will introduce Discrete Density Hidden Markov Models (DDHMM) and Continuous Density Hidden Markov Models (CDHMM).

2.1.1 Discrete Density Hidden Markov Models (DDHMM)

An DDHMM λ is defined by two matrices of *transition probabilities* \mathbf{A} and *observation probabilities* \mathbf{B} ($\lambda = (\mathbf{A}, \mathbf{B})$), which are defined over a finite set of states $S = \{s_1, \dots, s_N\}$ (start state s_1 , end state s_N) and a finite set of observations $O = \{o_1, \dots, o_M\}$.

An example where the states denote the temperature of some liquid, and the observations the color of the liquid with 5 states and 4 possible observations: $N = 5$, $S = \{s_1, s_2, s_3, s_4, s_5\} = \{s_1, \text{cold}, \text{warm}, \text{hot}, s_5\}$, $M = 4$, $O = \{o_1, o_2, o_3, o_4\} = \{\text{blue}, \text{lightblue}, \text{violet}, \text{red}\}$

$$\mathbf{A}_{N \times N} = \mathbf{A}_{5 \times 5} = \begin{bmatrix} 0 & a_{1,2} & a_{1,3} & a_{1,4} & 0 \\ 0 & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} \\ 0 & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} \\ 0 & a_{4,2} & a_{4,3} & a_{4,4} & a_{4,5} \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (2.1)$$

$$\mathbf{B}_{(N-2) \times M} = \mathbf{B}_{3 \times 4} = \begin{bmatrix} b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} \\ b_{3,1} & b_{3,2} & b_{3,3} & b_{3,4} \\ b_{4,1} & b_{4,2} & b_{4,3} & b_{4,4} \end{bmatrix} \quad (2.2)$$

Equation 2.1 shows the transition matrix for this example, where 0 means that there is no transition from state i to state j . Equation 2.2 shows the observation matrix for the respective example, where each row contains the observation probabilities for a state with observation. We cannot make observations in s_1 and s_5 , the start and end

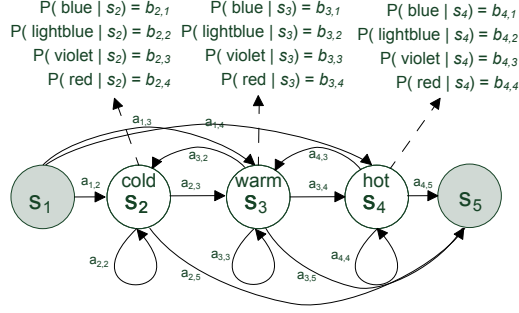


Figure 2.1: Discrete Density Hidden Markov Models (DDHMM).

state.

$$\mathbf{A} = \begin{bmatrix} 0 & a_{1,2} & a_{1,3} & a_{1,4} & 0 \\ 0 & a_{2,2} & a_{2,3} & 0 & a_{2,5} \\ 0 & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} \\ 0 & 0 & a_{4,3} & a_{4,4} & a_{4,5} \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (2.3)$$

Equation 2.3 shows a modified transition matrix for the above example where there is no possibility to go directly from s_2 (cold) to s_4 (hot) or vice versa. The DDHMM can also be represented in a state machine format as shown in Figure 2.1.

With this DDHMM we can compute the probability of observation sequences ($P(\mathbf{O}_1^T)$) like "blue, red, lightblue, red" or "red, red, red, red". Or we can compute the probability of an observation sequence given a state sequence ($P(\mathbf{O}_1^T | \mathbf{S}_1^T)$). This can be interpreted as the probability that the state sequence generated the observation sequence.

Concerning the probability of state sequences the state at time t is only dependent on the state at time $t - 1$ (Markov property):

$$P(S_t | S_{t-1}, S_{t-2}, \dots) = P(S_t | S_{t-1}) \quad (2.4)$$

The set of state transition probabilities (probability to go from state i to state j) is given by the matrix \mathbf{A} where

$$a_{ij} = P(S_t = s_j | S_{t-1} = s_i), \quad 1 \leq i, j \leq N \quad (2.5)$$

$$a_{ij} \geq 0, \quad \sum_{j=1}^N a_{i,j} = 1 \quad (2.6)$$

However, the state sequence is not directly observable (it is hidden). What we observe is a sequence of T observations $\mathbf{o}_1^T = o_1, \dots, o_T$.

In the case of a finite alphabet with M discrete observation symbols o_1, \dots, o_M the observation probabilities of being in state j while observing symbol k are given by the matrix \mathbf{B} with

$$b_{j,k} = b_j(k) = P(o_k|s_j) = P(O_t = o_k|S_t = s_j), \quad 1 \leq k \leq M \quad (2.7)$$

$$b_j(k) \geq 0, \quad 2 \leq j \leq N-1, \quad 1 \leq k \leq M \quad (2.8)$$

$$\sum_{k=1}^M b_j(k) = 1, \quad 2 \leq j \leq N-1 \quad (2.9)$$

2.1.2 Continuous Density Hidden Markov Models (CDHMM)

For modeling the time series data in a speech synthesis system with HMMs we need to model a continuous parameter space. For this modeling Continuous Density Hidden Markov Models (CDHMM) are used.

In CDHMM [10, 7] we will most times use the normal (=Gaussian) distribution defined by (mean μ , variance σ^2 , standard deviation σ)

$$p(x) = \mathcal{N}(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (2.10)$$

The Maximum Likelihood (ML) estimates $\hat{\mu}$, $\hat{\sigma}^2$ of μ and σ^2 are given by

$$\hat{\mu} = \frac{1}{N} \sum_{k=1}^N x_k \quad \hat{\sigma}^2 = \frac{1}{N} \sum_{k=1}^N (x_k - \hat{\mu})^2 \quad (2.11)$$

In many applications (speech synthesis and recognition, gesture recognition) we have an (uncountably) infinite number of possible observations. In CDHMMs the observation probability can be using a normal (=Gaussian) probability density function. For one variable: $b_j(o_k) = \mathcal{N}(o_k; \mu_j, \sigma_j^2)$. For multiple variables: $b_j(\mathbf{o}_k) = \mathcal{N}(\mathbf{o}_k; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$

$$\mu = 0, \sigma^2 = 0.49; \quad \boldsymbol{\mu} = \begin{bmatrix} 0 \\ 0.5 \end{bmatrix}, \boldsymbol{\Sigma} = \begin{bmatrix} 0.7 & 0 \\ 0 & 1.2 \end{bmatrix}, \begin{bmatrix} 0.7 & 0.5 \\ 0.5 & 1.2 \end{bmatrix}, \begin{bmatrix} 0.7 & -0.5 \\ -0.5 & 1.2 \end{bmatrix} \quad (2.12)$$

The CDHMM is defined over a finite set of states $S = \{s_1, \dots, s_N\}$ (start state s_1 , end state s_N) and an infinite set of observations $O = \{o_k \in \mathbb{R}\} \vee O = \{\mathbf{o}_k \in \mathbb{R}^n\}$. A CDHMM λ is defined by a matrix of *transition probabilities* \mathbf{A} and *observation*

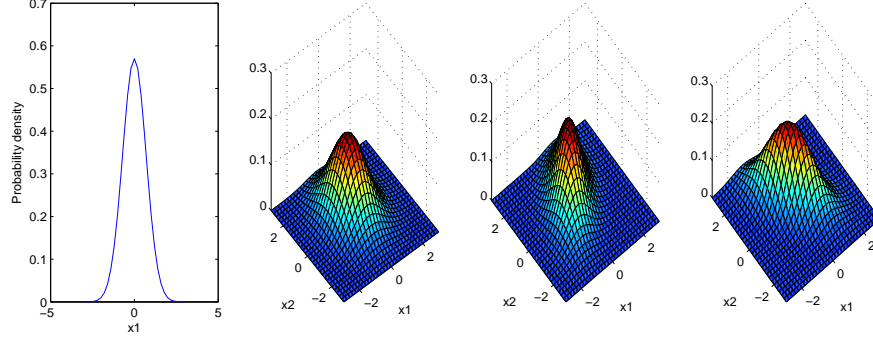


Figure 2.2: Normal distribution.

probabilities that are defined by N mean values μ_j (or N mean vectors $\boldsymbol{\mu}_j$) and N variances σ_j^2 (or co-variance matrices $\boldsymbol{\Sigma}_j$)

$$\lambda = (\mathbf{A}, (\mu_1, \dots, \mu_N), (\sigma_1^2, \dots, \sigma_N^2)) \quad \vee \quad (2.13)$$

$$\lambda = (\mathbf{A}, (\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_N), (\boldsymbol{\Sigma}_1, \dots, \boldsymbol{\Sigma}_N)). \quad (2.14)$$

Example shown in Figure 2.3 (the states denote the phonemes within a word, and the observations are the Mel-cepstral frequencies of the phonemes):

- $N = 6$, $S = \{s_1, s_2, s_3, s_4, s_5, s_6\} = \{s_1, \text{v}, \text{l}, \text{y}, s_6\}$
- HMM for the word "will" with German and Viennese pronunciation.

$$\mathbf{A}_{N \times N} = \mathbf{A}_{6 \times 6} = \begin{bmatrix} 0 & a_{1,2} & 0 & 0 & 0 & 0 \\ 0 & a_{2,2} & a_{2,3} & 0 & a_{2,5} & 0 \\ 0 & 0 & a_{3,3} & a_{3,4} & 0 & 0 \\ 0 & 0 & 0 & a_{4,4} & 0 & a_{4,6} \\ 0 & 0 & 0 & 0 & a_{5,5} & a_{5,6} \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (2.15)$$

In CDHMMs, the state sequence is also not directly observable (it is hidden). What we observe is a sequence of T observations $\mathbf{o}_1^T = o_1, \dots, o_T$. In the case of infinitely many observations, the observation probabilities of being in state j while observing symbol o_k are given by

$$b_j(o_k) = p(o_k | s_j) = p(O_t = o_k | S_t = s_j) \quad (2.16)$$

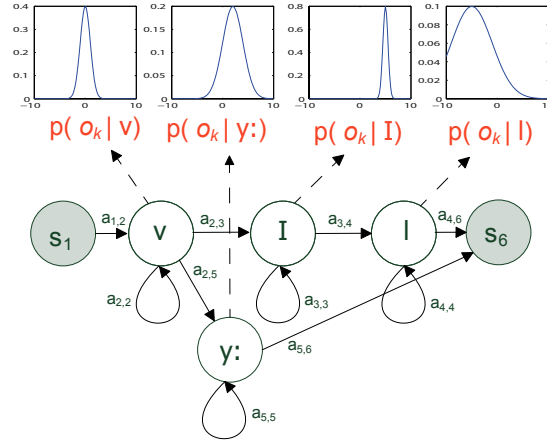


Figure 2.3: Continuous Density Hidden Markov Models (CDHMM).

$$= \mathcal{N}(o_k; \mu_j, \sigma_j^2), \quad o_k \in \mathbb{R} \quad (2.17)$$

$$b_j(o_k) \geq 0, \quad 2 \leq j \leq N-1, \quad o_k \in \mathbb{R} \quad (2.18)$$

$$\int_{-\infty}^{\infty} b_j(o_k) = 1, \quad 2 \leq j \leq N-1 \quad (2.19)$$

2.1.3 Basic problems for hidden Markov models (HMMs)

Three basic problems that need to be solved for HMMs are [7]:

1. Given an observation sequence \mathbf{o}_1^T and a model λ how can we compute the probability of the model producing the observation sequence, i.e., $P(\mathbf{o}_1^T | \lambda)$? (**word recognition** \rightarrow **forward algorithm**)
2. Given an observation sequence \mathbf{o}_1^T and a model λ how can we compute an optimal state sequence \mathbf{s}_1^T with $\mathbf{s}_1^T = \arg\max_{\mathbf{S}_1^T} P(\mathbf{o}_1^T | \mathbf{S}_1^T, \lambda)$? (**decoding, recognition** \rightarrow **Viterbi algorithm**)
3. How do we adjust the model parameters $\lambda = (\mathbf{A}, \mathbf{B})$, $\lambda = (\mathbf{A}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$ to maximize $P(\mathbf{o}_1^T | \lambda)$? (**Maximum Likelihood (ML) training** \rightarrow **Baum-Welch algorithm, Expectation Maximization (EM) algorithm**)

2.2 Speech synthesis

Speech synthesis is the task of generating a speech signal from a discrete representation, which is most of the times written text. Speech synthesis already has a very long history [11]. Today the most important approaches are parametric synthesis from HMMs, concatenative synthesis, and hybrid systems [12].

The main problems that have to be solved for speech synthesis from a user perspective are intelligibility and naturalness. From a more system oriented perspective flexibility, and the ability to model all types of speech are important.

The task to produce intelligible, i.e. understandable speech that has the same Word Error Rate (WER) as natural human speech, was already solved with diphone based speech synthesis systems [13]. With these systems a set of diphones is recorded for a language, which comprise a few thousand units, and during synthesis time these diphones are concatenated and their duration and F0 is adapted.

The task to produce naturally sounding speech was solved with the invention of unit selection based speech synthesis [14]. With this method a large corpus of diphones in different contexts is recorded and speech is generated by finding the most suitable diphone sequence. The design of the recording corpus is a set-cover problem [15].

From a system developer perspective unit selection based speech synthesis has the disadvantage of being very inflexible in terms of adapting a certain voice or changing its characteristics. A higher flexibility is achieved by HMM-based parametric speech synthesis [8, 9] where interpolation and adaptation methods can be used to change model parameters [16, 17, 18].

A task that is still unsolved is the generation of conversational speech, which is speech like in a natural human to human conversation. For achieving this it is necessary to be able to realize variety switching, prosody, and non-linguistic particles (filled pauses, hesitations, laughing, whispering) as well as the control of all these parameters from discrete textual input. This is a very hard problem and it can be argued that it is among the Artificial Intelligence (AI)-complete problems that also comprise natural language understanding and image understanding.

2.2.1 Applications

There are already numerous applications of speech synthesis, like web readers (<http://wien.at>), screen readers for blind users [19], spoken dialog systems used in call center automation and information systems, car navigation systems, personal digital assistants (Siri), and virtual reality applications.

2.2.2 Text-To-Speech synthesis (TTS)

In speech synthesis we generally have to generate speech from a textual representation. This is practical since a lot of digital textual data is available today and can serve

as input. For some applications like speech-to-speech translation other than textual representations on the concept level might be more adequate [20]. But even such systems often synthesize from text in the end and use the conceptual representation as an intermediary one. Therefore and since textual input is the largest available source of input TTS remains the main paradigm.

A TTS system consists of the following three building blocks:

1. Text analysis: Numbers, abbreviations, etc.
2. Grapheme To Phoneme (G2P) conversion
 - dictionary look-up
 - decision tree based grapheme-to-phoneme rules
3. Prosody prediction (pauses, durations, F0) and waveform generation
 - Concatenative: Unit selection speech synthesis
 - Parametric: HMM based speech synthesis
 - Concatenative and parametric: Hybrid systems

2.2.3 Text analysis

In text analysis we transform a written text into a form that is closer to read speech. The following example transformation illustrates this.

- **Example:** *Sie haben am 5.2.2011 51 Einheiten bestellt. Wollen sie mit ATM bezahlen?*
- **Transformed:** *Sie haben am fünften zweiten zweitausendelf einundfünfzig Einheiten bestellt. Wollen sie mit A T M bezahlen?*

We need to be able to analyze date, number (ordinal, cardinal, telephone number, zip-code, credit card number etc.), and acronyms. For dates and numbers grammars for a specific language have to be designed. For acronyms lists of acronyms can be used. Furthermore we need to predict which acronyms are spoken and which are spelled out (BP vs. BIP).

For transforming numbers we first check if we have a cardinal or ordinal number (i.e. 5. \rightarrow *fünfter* / *fünftens*). If a cardinal number is detected we can use a grammar to transform the numeral to a text string. Figure 2.4 shows a part of a German cardinal number grammar given by a Finite State Transducer (FST) [21].

A FST is a Finite State Automaton (FSA) that accepts a string and outputs / recognizes another string. FSAs are isomorphic to regular languages, and FSTs are isomorphic to regular relations. FSTs are closed under inversion (F^{-1}) meaning that

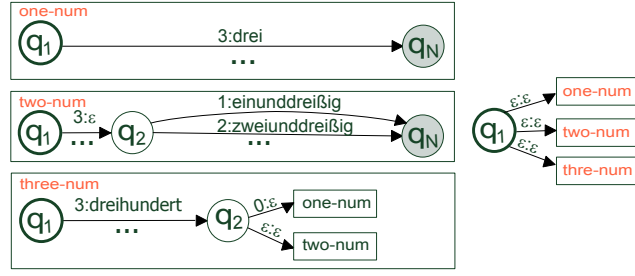


Figure 2.4: Finite state transducer (FST) F for cardinals (numeric-to-written).

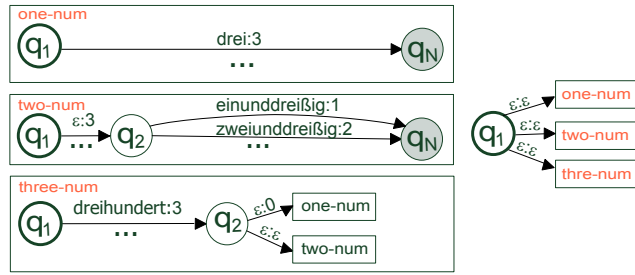


Figure 2.5: Finite state transducer (FST) F^{-1} for cardinals (written-to-numeric).

the inversion of an FSTs F is again a FST [22]. Figure 2.5 shows the respective inverted FST F^{-1} for our cardinal number conversion. With inversion we can transform a generator FST into a recognizer FST.

2.2.4 Grapheme To Phoneme (G2P) conversion

In the second step we have to convert from the textual (grapheme, letters) representation to the phonetic representations (phonemes). This is done by Grapheme To Phoneme (G2P) conversions or Letter To Sound (LTS) conversion. The whole process itself can be done in two steps. First we look up the words in a dictionary. If the word is not found we use LTS rules to predict the pronunciation. Here we can use hand-written rules or rules derived automatically. One method for predicting phones from letters is to learn decision trees.

Figure 2.6 shows part of three rules of a decision tree for converting the letters y , x , and w in German. The decision tree for x only consists of one leaf, which means that x always has to be converted to the two phone sequence $k s$. The letter y is transformed depending on the context of preceding and following letters (characters). "n.n=t" is the question if the character after the next character is a t , "p=s" is the question if the previous character is a s , "p.p=#" is the question if the character before the

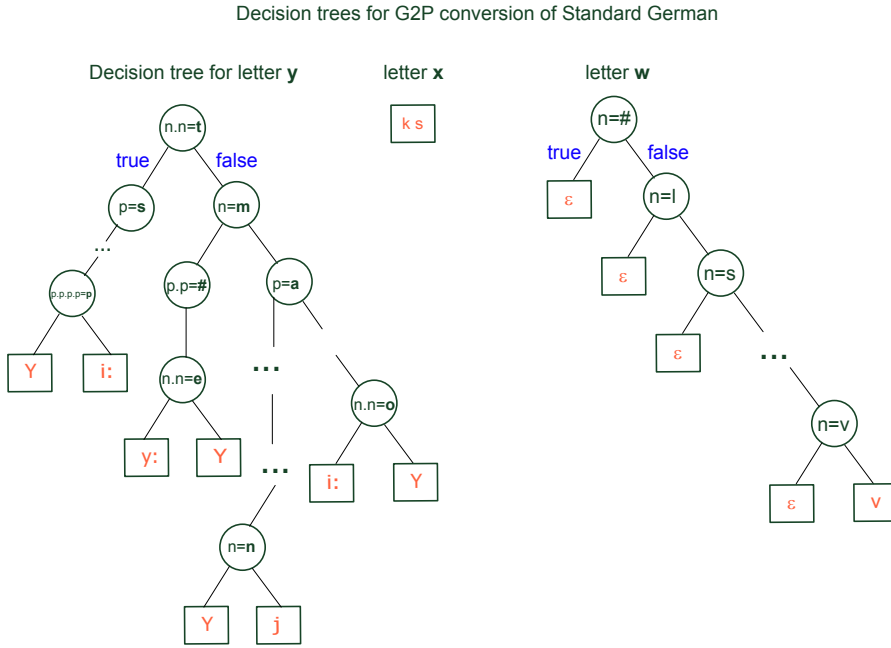


Figure 2.6: Decision trees for G2P conversion of Standard German.

previous character is the word beginning and so on. Depending on the context of the letter **y** it is transformed to one of the phones **Y**, **i:**, **y**, **j**. For the letter **w** we also use the empty phone ϵ , which means that for these contexts the letter does not generate a phone.

For learning decision trees we first have to construct an alignment of the training data by giving a set of allowable grapheme-to-phoneme mappings (graphemes in blue, phonemes in red). The training data consists of a phonetic lexicon.

begin

$$w_i = \text{p o s t b e a m t e} \rightarrow \text{p O s t b @ G S a m t @} \quad (\text{a} \rightarrow \text{G S a})$$

$$w_j = \text{r a c h e} \rightarrow \text{r a x } \epsilon \text{ @} \quad (\text{c} \rightarrow \text{x}, \text{h} \rightarrow \epsilon).$$

For each letter we then construct the set of training feature vectors and target values. If we consider the previous two and following two letters of a letter then we get for the letter **a** the following two features $a_i = [\text{b e a m t}]$ and $a_j = [\# \text{ r a c h}]$, where $\#$ denotes word beginning. The target values for the respective features are $f(a_i) = \text{G S a}$ and $f(a_j) = \text{a}$. Given a set of features F we can define the purity of the feature set $g(f(F))$ by the ratio of different phones by number of phones. The feature set is more pure if it maps the features to a small number of phones. Using this as

training data, we can devise a learning algorithm.

Algorithm 1 Algorithm for decision-tree based clustering. Set of features $F = \{1, \dots, n\}$. $f(S) = \{p_1, \dots, p_n\}$ returns the phones for the features. g computes the purity of the phone set.

```

1: procedure DECISIONTREE( $F$ )
2:   if stopping criterion is met then
3:     return  $F$ 
4:   else
5:     Split all features  $F$  using all  $m$  questions.  $\{\{F_{1,1}, F_{1,2}\}, \dots, \{F_{m,1}, F_{m,2}\}\}$ 
6:      $j = \operatorname{argmax}_i (g(f(F_{i,1})) + g(f(F_{i,2})))$ 
7:     Add question  $j$ 
8:     DECISIONTREE( $F_{j,1}$ )
9:     DECISIONTREE( $F_{j,2}$ )
10:  end if
11: end procedure

```

2.2.5 Unit selection speech synthesis

For the third step prosody prediction and waveform generation three approaches are used in state-of-the-art systems. In unit selection synthesis the Viterbi algorithm is used to find the best sequence of units from a large database. The algorithm uses two cost functions, a concatenation cost defined between two speech units in the database and a target cost defined between a unit in the database and a linguistic target description.

The concatenation cost is defined as $C^c(s_{i-1}, s_i) = \sum_{k=1}^p w_k^c C_k^c(s_{i-1}, s_i)$ where C_k^c are spectral and acoustic features that measure the distance between diphones. As concatenation cost we often use the Euclidean distance between Mel Frequency Cepstral Coefficients (MFCC) (MFCC + Δ features) of diphones ($\mathbf{cep}_i \in \mathbb{R}^{26}$).

$$\|\mathbf{cep}_{i-1} - \mathbf{cep}_i\| = \sqrt{\sum_{j=1}^{26} (\mathbf{cep}_{i-1}(j) - \mathbf{cep}_i(j))^2} \quad (2.20)$$

The target cost is defined as $C^t(t_i, s_i) = \sum_{j=1}^p w_j^t C_j^t(t_i, s_i)$ where C_j^t are costs defined on phonetic and prosodic contexts that measure the distance between target unit and database unit. For the target cost we need to use symbolic features since we compare a unit description (target unit) with a concrete database unit. Therefore we can use phonetic and prosodic context, and predicted duration and F0 as features.

The optimization problem of finding the optimal sequence of units (states) with the

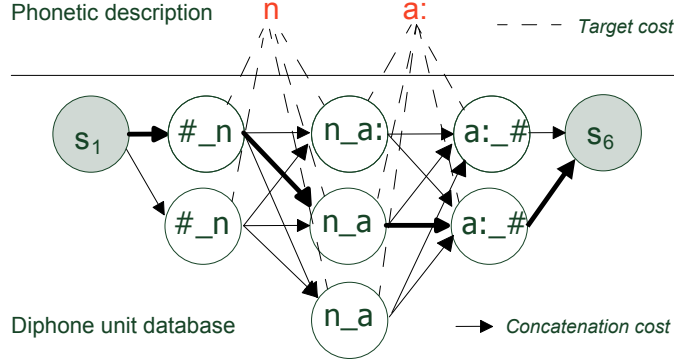


Figure 2.7: Diphone unit graph for the Viennese word “nein” [n a:].

Viterbi algorithm can be defined as

$$\hat{S}_{1:n} = \underset{S_{1:n}}{\operatorname{argmin}} C(T_{1:n}, S_{1:n}) = \sum_{i=1}^n C^t(t_i, s_i) + \sum_{i=2}^n C^c(s_{i-1}, s_i). \quad (2.21)$$

Figure 2.7 shows a database of (diphone) units for the Viennese word [n a:]. The concatenation costs are in solid lines, target costs are in dashed lines. The best path found by the Viterbi algorithm is shown in bold solid lines. The units along this path are then concatenated to synthesize the word, hence the name concatenative synthesis.

The Viterbi algorithm is a dynamic programming method [23]. It breaks up the general problem into sub-problems by using the HMM structure. Then it solves all the sub-problems. In this way it can find the best possible path in the graph. In real world systems we have to use a heuristic version of the Viterbi algorithm due to the possibly large number of sub-problems. In a Viterbi beam search [24] we restrict the number of paths that are considered for expansion to the n paths with the highest probability / cost (active path pruning) and / or the paths that have at least probability p (maximal cost c) (beam pruning). A Viterbi beam search does not always find the best path.

2.3 Hidden Markov Model (HMM) based speech synthesis

In HMM-based speech synthesis we derive model parameters from a speech database, which are then used for synthesis. For modeling spectral and F0 parameters jointly we use a multi-stream probability distribution,

$$b_j(o) = \prod_{s=1}^S \left[\sum_{m=1}^M c_{j sm} \mathcal{N}(o_s, \mu_{j sm}, \Sigma_{j sm}) \right]^{\gamma_s} \quad (2.22)$$

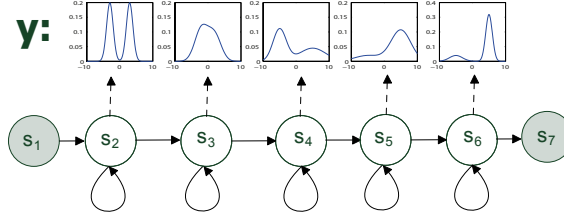


Figure 2.8: Five state HMM for the phone **y:** with Gaussian mixture observation PDF.

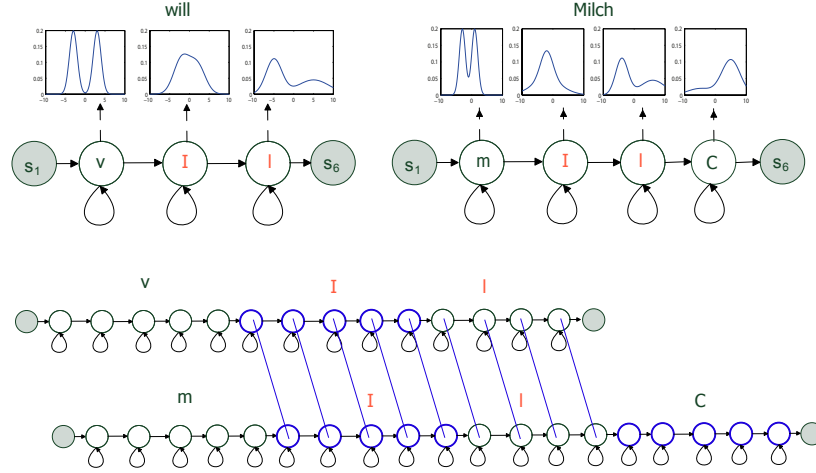


Figure 2.9: Re-usage of data with sub-word HMMs (below) as compared to word HMMs (above).

$$1 \leq j \leq N, \quad \sum_{s=1}^S \gamma_s = S \quad (2.23)$$

where S is the number of streams, M is the number of mixtures for the Gaussian Mixture Model (GMM), and γ_s are the stream weights.

We also change from whole word models as described in Section 2.1 to sub-word models, where one phoneme is modeled by an HMM with 5 states (5 emitting states plus 2 non-emitting states) as shown in Figure 2.8.

Sub-word models are used to re-use the training data within words / utterances and across words / utterances as shown in Figure 2.9. Sub-word modeling is also necessary to be able to synthesize / recognize text that contains a large (possibly unlimited) vocabulary. For sub-word modeling the EM-training formulas from Section 2.1 have to be adapted (embedded training).

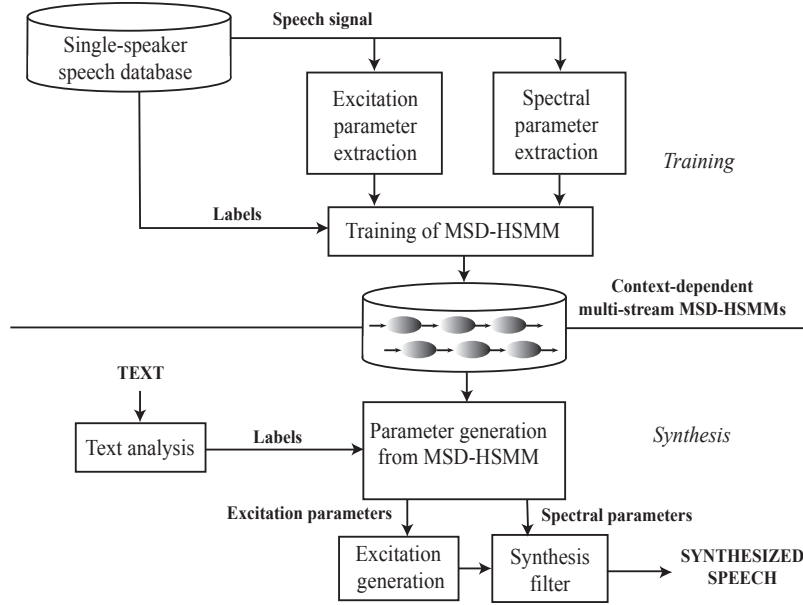


Figure 2.10: Speaker dependent HMM-based speech synthesis system (Figure redrawn from [25]).

2.3.1 Speaker dependent HMM based speech synthesis system

In this thesis we use a speaker dependent HMM-based system for training models for spectral (Mel-cepstrum), excitation parameters (F0), and duration. Figure 2.10 shows the components of a speaker dependent synthesis system [9]. Starting from a single speaker database with labels we extract excitation (F0) and spectral (MFCC) parameters and train context-dependent HMM models. For synthesis we transform a text into a sequence of full context labels and then we use the Maximum Likelihood (ML) parameter generation algorithm [8], which will be discussed later, to generate a sequence of excitation and spectral features. This sequence of features is then used by a vocoder to create synthesized speech. For training full context models we apply context clustering.

2.3.2 Context clustering

To model context dependencies a variety of contexts like previous and following phones, syllable features etc. is taken into account. To deal with the curse of dimensionality [26], which appears when a wide context is taken into account, we have to apply clustering methods to tie states [27, 28, 29]. In data-driven clustering multiple states are tied to the same probability distribution. This makes the models smaller and increases the training data per state.

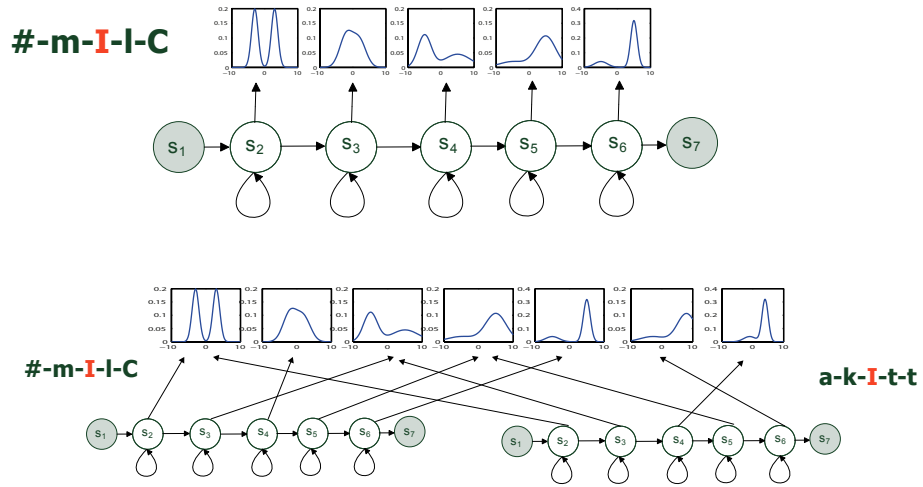


Figure 2.11: Data-driven state tying for [I] quinphones.

Figure 2.11 shows how different states of two full context models **#-m-I-l-C** (**I** in the left phone context **#-M** and the right phone context **l-C**) and **a-k-I-t-t** (**I** in the left phone context **a-k** and the right phone context **t-t**) are tied to use the same PDFs (below) versus an untied full context model.

To tie states and to deal with unseen data (i.e. unseen quinphones) decision-tree based clustering is performed where the whole possible feature space is clustered. For clustering of quinphones the quinphone context, acoustic-articulatory features, as well as syllable and word level features can be used. The clustering questions can be based on features from any linguistic level.

- preceding, current, and succeeding phones;
- acoustic and articulatory classes of preceding, current, and succeeding phones;
- the part of speech of the preceding, current, and succeeding words;
- the number of syllables in the preceding, current, and succeeding accentual phrases;
- the position of the current syllable in the current accentual phrase;
- the number of words and syllables in the sentence;
- the specific language variety in the case of clustering of dialects (i.e. Viennese dialect or Standard Austrian German).

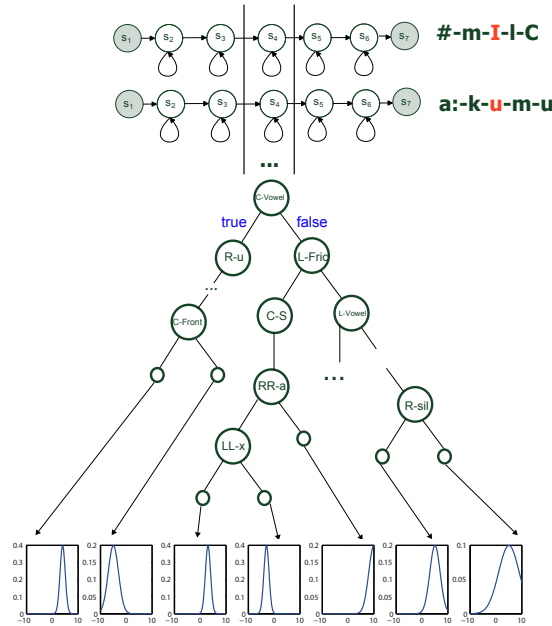


Figure 2.12: Decision-tree based state tying.

In shared decision-tree clustering we train one decision tree per state, which is the method mostly used in speech synthesis. In phonetic decision-tree clustering we train one decision tree per state and phone. This method is mostly used in speech recognition [30].

For clustering we can use the same clustering algorithm that was used in Subsection 2.2.4 for G2P conversion. We only need to define an impurity on distributions and replace the features by the full context model states.

Figure 2.12 shows the result of shared decision-tree clustering for quinphone models. We can see that there is one separate decision tree trained for the 3rd state of all quinphone models. By traversing the tree answering the specific questions for one full context model we find a leaf node that tells us which model we should use for this state. Separate decision trees are trained for each state and for spectrum, F0 and duration.

Figure 2.13 shows part of a concrete decision tree for the Mel-cepstrum of the 3rd state (central state in 5-state HMM) that was trained on speaker data from Standard Austrian German and Viennese dialect data. In this model we also introduced the question if the utterance was Standard or dialect (“Is-Viennese-Dialect”). As can be seen from this figure the question already appears at the top of the decision tree, with the first question splitting the vowels and non-vowels, and thereby creating two separate sub-trees for Viennese and Austrian German vowels.

$$l(i) = -\log P_{\hat{\theta}^{(i)}}(\mathbf{x}) + \frac{\alpha_i}{2} \log n + \log I \quad (2.24)$$

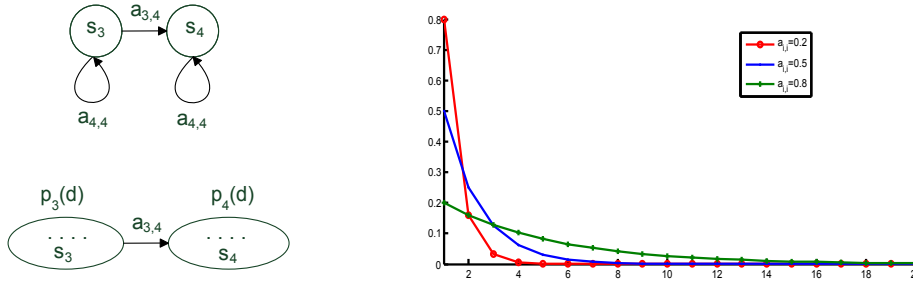


Figure 2.14: Topology for implicit (above) and explicit (below) state duration. Geometric distribution (right).

2.3.3 Duration modeling

HMMs include an implicit modeling of state durations through state transition probabilities as described in Section 2.1. With this method we can only model geometric distributions as shown in Figure 2.14 (right). Figure 2.14 shows the geometric distribution for three values of self-transition probabilities (0.2, 0.5, 0.8). We can see that we have a higher probability for shorter state durations than for longer ones. The probability of d observations in state i is given by a geometric distribution:

$$p_i(d) = a_{ii}^{d-1}(1 - a_{ii}) \quad d \in \mathbb{N} \quad (2.25)$$

Modeling duration with this approach is often sufficient for speech recognition, but for speech synthesis we need an accurate duration model since the duration of phones (states) is an essential part of prosody [30]. For accurate duration modeling we want to be able to model a general class of duration distributions. One such general class is the normal distribution, which allows us to model the duration of a state by its mean and standard deviation.

$$p_i(d) = \frac{1}{\sqrt{2\pi\sigma_i^2}} e^{\left(-\frac{(d-\mu_i)^2}{2\sigma_i^2}\right)} \quad (2.26)$$

We call this type of modeling explicit duration modeling, which leads to hidden semi-Markov models (HSMMs). The estimation formulas defined in Section 2.1 have to be adapted to use explicit duration PDFs. For estimating the duration PDFs we also use decision tree based clustering as it is used for spectrum and F0 models.

2.3.4 Parameter generation

One main innovation, which makes HMM-based speech synthesis possible, is the development of parameter generation algorithms that allow for the derivation of a sequence

of parameters from an HMM that maximizes the likelihood [31]. These parameter generation also takes dynamic features into account. It can be used for generating any feature sequence from an HMM, e.g. also for visual or motion features. Given an HMM λ the parameter algorithm proceeds as follows.

1. Select a sequence of phone HMMs for the text to be synthesized.
2. Find the most likely sequence of observations given the selected phone models.
3. Take into account dynamic features (derivative of parameters) otherwise only means are selected.

Since the joint optimization of state and observation sequence is often computationally too expensive, we can find an approximate solution by splitting the problem into two sub-problems, finding the optimal state sequence \mathbf{S}^* and finding the optimal observation sequence \mathbf{O}^* given the optimal state sequence. The whole optimization can be described as follows.

$$\begin{aligned}
\mathbf{O}^* &= \underset{\mathbf{O}}{\operatorname{argmax}} P(\mathbf{O}|\lambda, T) \\
&= \underset{\mathbf{O}}{\operatorname{argmax}} \sum_{\mathbf{S}} P(\mathbf{O}, \mathbf{S}|\lambda, T) \quad (P(A) = \sum_B P(A, B)) \\
&\approx \underset{\mathbf{O}}{\operatorname{argmax}} P(\mathbf{O}|\mathbf{S}^*, \lambda, T).
\end{aligned}$$

$$\mathbf{S}^* = \underset{\mathbf{S}}{\operatorname{argmax}} P(\mathbf{S}|\lambda, T) \tag{2.27}$$

The overall goal is to find the optimal observation sequence \mathbf{O}^* . This can be done by maximizing the probability of observation sequences \mathbf{O} given a certain HMM λ and a time T (without loss of generality). By summing over all possible state sequences of length T we can solve this optimization. This is computationally too expensive, that's why we condition on the most likely state sequence \mathbf{S}^* . \mathbf{S}^* can be found by a separate optimization step.

For finding \mathbf{S}^* we maximize the probability of state sequences given a certain HMM λ and a time T . T can be set to the sum of mean values $\sum_{k=1}^K \mu_k$ to get the average speaking rate if the explicit state duration is a normal distribution. In this case the optimal state sequence \mathbf{S}^* is simply the one where we are μ_k times in state S_k . If we want to have a different duration than average (slower or faster) we can get the state duration d_k as

$$d_k = \mu_k + \rho \sigma_k^2 \quad 1 \leq k \leq K \tag{2.28}$$

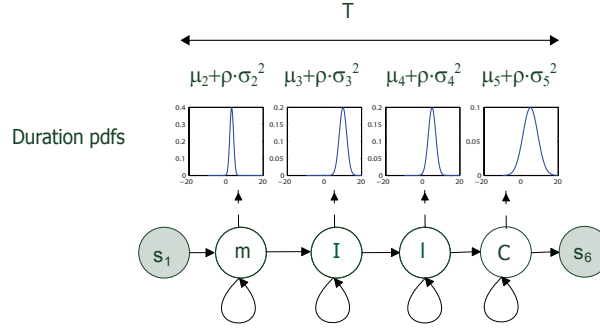


Figure 2.15: Duration synthesis.

$$\rho = \frac{\left(T - \sum_{k=1}^K \mu_k\right)}{\sum_{k=1}^K \sigma_k^2} \quad (2.29)$$

Given a certain duration T that we want to achieve we can compute a parameter ρ that can be used to modify the standard deviation of the duration distributions in the appropriate way. d_k is then the duration of states k for the best state sequence \mathbf{S}^* of length T . If we only want to find the static observation features we can just take d_k times the mean values of state S_k of the respective spectrum and F0 models. Figure 2.15 shows the process of duration synthesis.

Now we want to find the optimal observation sequence \mathbf{O}^* given the optimal state sequence \mathbf{S}^* with T states for static and dynamic parameters. \mathbf{C} are the static feature vectors, \mathbf{O} contains static and dynamic feature vectors, i.e. $\mathbf{O} = \mathbf{WC}$. \mathbf{W} is a given matrix that computes \mathbf{O} (static and dynamic features) when applied to the static features. Since dynamic features are computed from static features via linear regression this computation can also be written in matrix form. Maximizing $P(\mathbf{O}|\mathbf{S}^*, \lambda, T)$ with respect to \mathbf{O} is the same as maximizing $P(\mathbf{WC}|\mathbf{S}^*, \lambda, T)$ with respect to $\mathbf{C} = [c_1, \dots, c_T]$. This maximization can be achieved by setting the derivative to zero.

$$\frac{\partial P(\mathbf{O}|\mathbf{S}^*, \lambda, T)}{\partial \mathbf{C}} = \mathbf{0} \quad (2.30)$$

The derivative gives us the static parameter sequence \mathbf{C} that is optimal in terms of maximal likelihood concerning static and dynamic features given state sequence \mathbf{S}^* . Taking the derivative and setting to zero gives

$$\mathbf{W}^T \Sigma_q^{-1} \mathbf{WC} = \mathbf{W}^T \Sigma_q^{-1} \mu_q \quad (2.31)$$

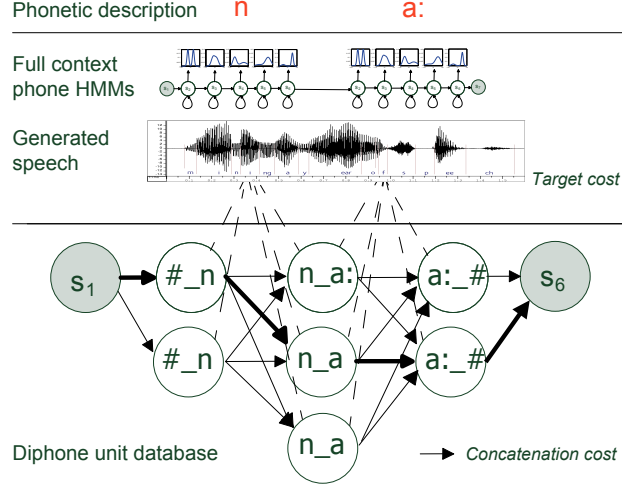


Figure 2.16: Hybrid speech synthesis system where speech generated from HMMs is used in the target cost function.

$$\mathbf{C} = (\mathbf{W}^T \boldsymbol{\Sigma}^{-1} \mathbf{W})^{-1} \mathbf{W}^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_q \quad (2.32)$$

where \mathbf{C} is a $MT \times 1$ static feature vector per state, $\boldsymbol{\mu}_q$ is a $3MT \times 1$ sequence of mean vectors per state, $\boldsymbol{\Sigma}_q^{-1}$ is a $3MT \times 3MT$ sequence of inverse of diagonal covariance vectors per state, and \mathbf{W} is a $3MT \times MT$ weight matrix. In this way we can compute the static observation sequence \mathbf{C} that is optimal given the dynamic features. \mathbf{C} gives us the optimal spectrum and F0 parameters that we can use to synthesize speech.

2.3.5 Hybrid systems

A general problem of HMM-based speech synthesis systems is their tendency of over-smoothing, which mostly comes from the averaging of spectral features. Unit selection based speech synthesis, which was described in Subsection 2.2.5, does not have this problem since the natural speech signal is used. Unit selection does however produce errors at bad concatenation points. To combine the best of both worlds hybrid systems have been proposed recently where trajectories from HMM-based synthesis are used to define the target cost. In terms of computation these systems need to generate parameters first and then do the Viterbi search over the unit database.



Figure 2.17: Part of the musical score from a Latin song (Figure from [1]).

2.4 MusicXML

MusicXML [1] is an eXtended Markup Language (XML) format that can be used to describe musical scores. As XML format it has the advantage of being easily parsable in many computer languages through defined libraries. In Chapter 3 we develop a parser that splits up large MusicXML files of whole opera songs into utterance size chunks. It was invented by the company MAKEMUSIC [1] and is a de facto standard that can be processed by the main music editing programs.

Version 3.0 of the MusicXML format was released in August 2011. Version 3.0 includes both a Document Type Definition (DTD) and W3C XML Schema Definition (XSD) [1].

MusicXML is available under a public license. For our manual editing of MusicXML we used the MuseScore [32] program that is available for Windows and Linux. It can be used to play MusicXML files as MIDI files, and can also transform MusicXML files to MIDI files from the command line.

Figure 2.17 shows part of the musical score of a Latin song. This song only consists of a singing part. In opera songs we typically have a singing and a piano part. We can see several syllables where syllable duplication is necessary since multiple notes are attached to one syllable. This is indicated by the slur symbol \smile or \frown . The one syllable word **Quem** at the beginning is associated with two notes. Suppose the phonetic transcription of **Quem** is **k w e m**, then it should be sung as **k w e - e m** with the respective notes G4 and F4 on the now two syllable word.

The MusicXML file starts with some header information and information for the different parts of the score. The whole part is divided into measures (`<measure> ... </measure>`). In this example there is only one measure. Each measure can have attributes that define the divisions, key, and clef, which are in this case

```
<attributes>
  <divisions>8</divisions>
  <key>
    <fifths>0</fifths>
```

```

    <mode>major</mode>
  </key>
  <clef>
    <sign>G</sign>
    <line>2</line>
  </clef>
</attributes>

```

The measures then contain the different notes with their pitch, slur, and lyric information

```

<note>
  <pitch>
    <step>G</step>
    <octave>4</octave>
  </pitch>
  <duration>8</duration>
  <type>quarter</type>
  <notations>
    <slur number="1" placement="below" type="start"/>
  </notations>
  <lyric number="1">
    <syllabic>single</syllabic>
    <text>Quem</text>
  </lyric>
</note>

```

The first note in this case is the quarter note G4. It is also indicated that a slur starts here with information for placement of the slur in the editor. The lyric contains syllabic information saying that it is a single syllable. For multi-syllabic words like **Chri - sti** in this example the syllabic element can have the value begin, middle, or end. The note after this note contains no information on lyrics but an information that the slur end there.

2.5 Singing synthesis

By singing synthesis we refer to the task of generating an acoustic signal of a singing person. The synthesis output is thereby controlled by a text input and a musical score that is aligned with the text. The textual data is also divided into syllabic sequences. This input data can be given as MusicXML [1] file. Several methods for synthesis of singing have been proposed in the literature, which we will shortly discuss here.

2.5.1 Articulatory synthesis of singing

[33] proposes an articulatory singing synthesis system. The articulatory synthesizer takes gestural scores, which are fed into models of the vocal tract and the vocal folds that results in a dynamic features trajectory. This trajectory can then be used to control the synthesizer. The gestural score can either be developed by hand or rules can be used. One disadvantage of this synthesizer is the difficulty to obtain enough articulatory data to train models automatically, as can be done with acoustic recordings.

Figure 2.18 shows part of the articulatory synthesizer that transforms articulatory gestures to feature trajectories. The first three gestures control the vocal tract model, the remaining three gestures control the vocal fold model. The parameters of the vocal tract were determined by Magnetic Resonance Image (MRI) images of sustained speech sounds. Vocal tract parameters for co-articulated consonants were determined from dynamic MRI images. The vocal tract and vocal fold model is used to create a dynamic area function from the articulatory gestures.

An important feature of the synthesizer are F0 dependent target shapes for vowels, which reflect the fact that singers often change the vocal tract when singing a certain

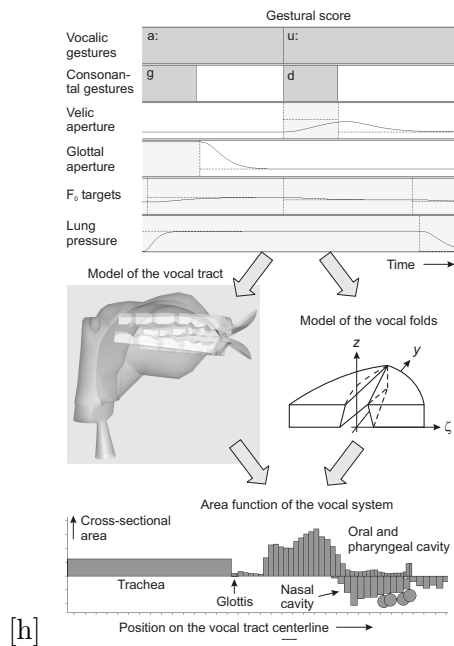


Figure 2.18: From gestural scores to trajectories in the articulatory synthesizer (Figure from [33])

vowel depending on the F0.

2.5.2 Conversion from speaking voice to singing voice

[34, 35] proposes a system that can convert a speaking voice into a singing voice by using the STRAIGHT vocoder [36]. This vocoder can also be used for HMM-based speech and singing synthesis and will be discussed in Subsection 2.6.5. Figure 2.19 shows a diagram of the vocal conversion system. First the speaking speech signal is analyzed by the STRAIGHT vocoder into spectral, F0, and aperiodicity parameters. Spectral and aperiodicity parameters are modified by a duration model and a spectral model. F0 parameters are taken from the musical score that is to be synthesized and are modified by a F0 control model. The F0 model modifies the flat F0 contour coming from the musical score by taking into account the following four phenomena.

1. Overshoot: deflection exceeding the target note after a note change.
2. Vibrato: a quasi-periodic frequency modulation (4-7 Hz).
3. Preparation: a deflection in the direction opposite to a note change observed just before the note change [35].

All four phenomena can be modeled by a second order system, where the parameters of the system can be learned from F0 contours extracted from natural singing signals. If incorporated into a singing synthesis system a model for the parameters can also be learned and parameters can be changed dynamically. The transfer function of the second order system is given as

$$H(s) = \frac{k}{s^2 + 2\zeta\omega s + \omega^2} \quad (2.33)$$

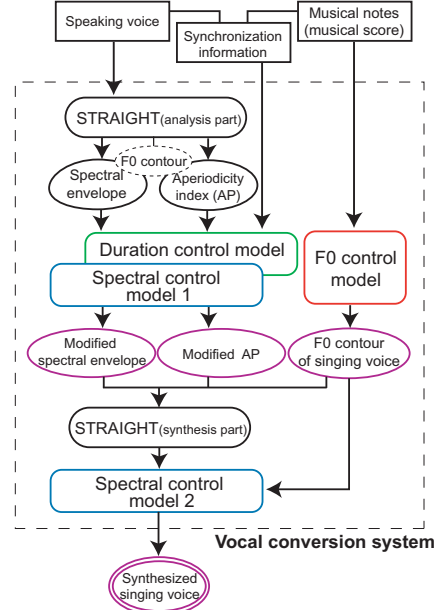


Figure 2.19: Vocal conversion system diagram (Figure from [34]).

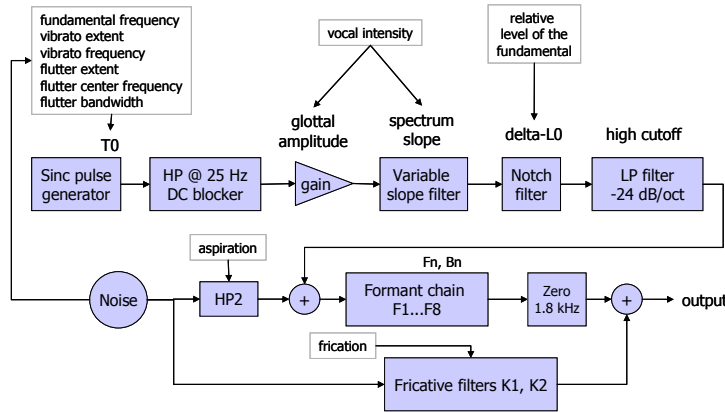


Figure 2.20: Formant singing synthesis system (Figure from [38]).

where ω is the natural frequency, ζ is the damping coefficient and k is the proportional gain of the system. By optimizing the system parameters on actual F0 contours using the non-linear least squares method optimal values for ω , ζ , and k are found for overshoot, vibrato and preparation models.

The duration model changes the duration dependent on empirically found parameters and duration constraints coming from the musical score. The first spectral model emphasizes the peak of the spectral envelope at the so called “singing formant”, which was shown by [37] to lie near 3 kHz.

Then the modified parameters are used to synthesize the signal with the STRAIGHT vocoder. After synthesis another spectral control model is applied that synchronizes the formant amplitude with the frequency modulation of the F0 contour.

2.5.3 Formant based synthesis of singing

[38] proposes a rule-based formant synthesis system that has the advantage that it can be used without any acoustical training data. Figure 2.20 shows the system diagram of the formant synthesis system. The system takes text data and musical score as input and generates 28 parameters at 100 Hertz. These parameters are then used to control the formant synthesizer.

2.5.4 Diphone based singing synthesis system VOCALOID

[39] proposes a singing synthesis system based on waveform concatenation. At recording time all possible combinations of consonant-vowel, vowel-consonant, vowel-vowel have to be recorded. This technology was developed by Yamaha and is licensed to other companies that sell commercial versions of singing synthesizers. In synthesis the

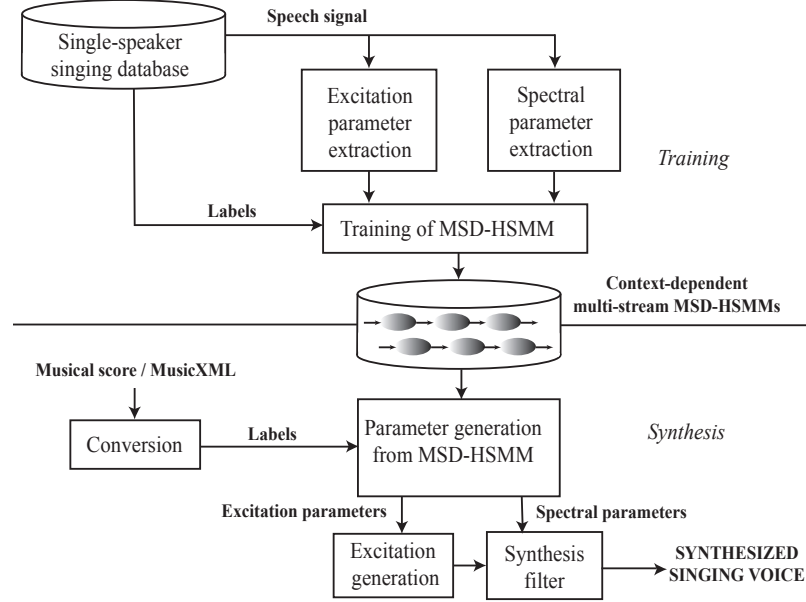


Figure 2.21: Basic speaker dependent HMM-based singing synthesis system (Figure redrawn after [40])

pitch of the selected units has to be changed to the desired pitch and the timbre has to be smoothed at concatenation points. All this is done in the frequency domain. For changing the pitch the power spectrum is divided into different regions, which are then scaled to the desired pitch.

2.6 Hidden-Markov-Model (HMM) based singing synthesis

The system that will be used in this thesis is based on HMMs. Acoustic singing synthesis has already been investigated within the HMM framework. HMM-based singing synthesis uses the parameter generation algorithm that was introduced in Subsection 2.3.4 to generate the necessary parameters. The basic HMM-based singing synthesis system is shown in Figure 2.21.

The system architecture of the singing system is similar to the one shown in Subsection 2.3.1 for speaker dependent speech synthesis. Instead of the textual input data a musical score or MusicXML transcriptions are used, and also different features are used for training of spectral, F0, and duration models.

2.6.1 Context and time-lag modeling

[41] first proposed to use the HMM framework for singing synthesis. As contextual features for clustering [41] introduced the following features:

- phoneme: The preceding, current, and succeeding phonemes.
- tone: The musical tones of the preceding, current, and succeeding musical notes (e.g. “A4”, “C5#”, etc.).
- duration: The durations of the preceding, current, and succeeding musical notes (in 100 ms unit).
- position: The positions of the preceding, current, and succeeding musical notes in the corresponding musical bar (in triplet thirty-second note) [41].

For alignment between musical score and voice, the system developed in [41] has introduced a time-lag model. In this way it can model the time difference between note timing from score information and actual timing from a singer that is not following the musical score exactly. A separate decision tree is trained for the time-lag models by comparing the timing from the musical score with the actual timing from a forced alignment of training data and models.

2.6.2 Rich context modeling, vibrato modeling, and F0-shifting

[40] introduces an HMM-based singing voice synthesis system that uses F0-shifted pseudo data for training and includes a simple periodic modeling of vibrato. [40] also defines an extended set of contextual features used for training the different models. The features according to [40] are:

- Phoneme
 - Quinphone: a phoneme within the context of two immediately preceding and succeeding phonemes.
- Mora
 - The number of phonemes in the (previous, current, next) mora.
 - The position of the (previous, current, next) mora in the note.
- Note
 - The musical tone, key, beat, tempo, length, and dynamics of the (previous, current, next) note.
 - The position of the current note in the current measure and phrase.

- The tied and slurred flag.
- The distance between the current note and the (next, previous) accent and staccato.
- The position of the current note in the current crescendo and decrescendo.
- Phrase
 - The number of phonemes and moras in the (previous, current, next) phrase.
- Song
 - The number of phonemes, moras, and phrases in the song [40].

Through the decision tree based clustering relevant features are selected for building up the decision trees for spectrum, F0, and duration. For training the model for F0 data, log F0 is shifted up or down in halftones, which largely increases the available amount of training data. Vibrato is assumed as a periodic fluctuation of F0 and two vibrato parameters are estimated from the training data and added to the observation vector for training a separate stream for the vibrato parameters. [40] also introduces the SINSY [3] synthesis system that can use MusicXML as input for synthesis.

2.6.3 Adaptive F0 modeling

Since the correct modeling of F0 is of special importance in singing synthesis [42] proposes a method that models F0 differences between musical score and data within the adaptive HMM-based framework using speaker adaptive training [43].

2.6.4 Syllable allocation and duplication

[44] extends a Japanese singing synthesis system for the English language. This system also includes a vibrato modeling component, that extracts vibrato features for training and also synthesizes these features for F0 generation. Otherwise it is similar to the basic singing synthesis system discussed above.

As an extension for the English language [44] introduces syllabic stress as an additional feature that is used in clustering. This is achieved by introducing several placeholders for language independent contexts in previous, current, and next syllable, which is used for English as feature indicating stress or no stress, and is undefined for Japanese.

Syllable allocation refers to the task of synchronizing the syllable structure found in MusicXML with the syllable structure from the lexicon. The MusicXML syllable structure is on the grapheme (letter) level, while the syllable structure from the lexicon is on the phonemic level. Mostly these two transcriptions agree, but there are some

Table 2.1: Diphthong duplication rules (Table from [44]).

Original	ey	ay	ow	aw	oy
Duplicated	eh, ey	aa, ay	ao, ow	aa, aw	ao oy

cases where they may be different. Figure 2.22 shows such a case where we have two syllables at the grapheme level in the MusicXML transcription and three syllables in the lexicon for the English word **everything**. If we apply the constraint that we have to assign at least one phonetic syllable to one grapheme syllable, there are two possible assignments.

[44] proposes two syllable allocation methods, one left-to-right method that results in the first allocation in Figure 2.22, and one score based method. With the score-based allocation method the number of characters in the MusicXML syllables are counted, where characters that often belong to vowels (a, e, i, o, u) are counted twice. Based on this score c_n a score w_n for each note is computed as

$$w_n = \frac{Sc_n}{\sum_{k=1}^N c_k} \quad (2.34)$$

where N denotes the number of notes in a word and S denotes the number of syllables from the lexicon. Using this score w_n the syllables are allocated with an iterative algorithm.

Syllable duplication is necessary for cases where multiple different notes are mapped to one syllable, which happens very often in opera singing. For these cases [44] proposes two different methods, a simple duplication method, and a rule based method. The simple duplication method cuts the syllable at the vowel (nucleus) and duplicates the vowel to the multiple notes. The remaining part of the syllable is mapped to

	every	-	thing
1:	[eh]		[v, r, iy th, ih, ng]
2:	[eh v, r, iy]		[th, ih, ng]

Figure 2.22: Two different syllable allocation methods (Figure from [44]).




One note	Two notes
 smile [s, m, ay, l]	  smi - le a: [s, m, ay] [ay, l] b: [s, m, a] [ay, l]

Figure 2.23: Syllable duplication (Figure from [44]).

the last note of the syllable. This results in the first mapping shown in Figure 2.23. The disadvantage of this method is that for diphthongs it simply copies them, which is not what happens in singing, so that we have a repetition of diphthongs (**ay ay** in this example). In rules based duplication a set of rules shown in Table 2.1 is used for diphthong such that the diphthong **ay** from our example is duplicated as **aa ay** (**aa** being a long **a**).

2.6.5 Vocoding

Different vocoding methods have been proposed for HMM-based synthesis. The vocoder is used to synthesize a speech signal from the parameters generated from the model. In the analysis part the vocoder is used to parametrize the speech signals, which are then used for training.

[45] presents an evaluation of different vocoders for singing synthesis. These vocoders can also be used in an HMM-based system. They evaluate the vocoders on a copy-synthesis task where a speech signal is analyzed and immediately re-synthesized using the vocoder. In this case the vocoder is seen as a speech coder and the effect of the codec can be measured.

They differentiate between different vocoder types and instantiations of these types where the ones in bold are evaluated in their study (Vocoder classification taken from [45]).

- Source-filter with residual modeling:
 - **Pulse vocoder** [46]
 - **Deterministic plus Stochastic Model (DSM)** [47]
 - Closed-Loop Training
 - Mixed Excitation
 - STRAIGHT
- Sinusoids+noise models:
 - **Harmonic plus Noise Model (HNM)** [48]
 - Harmonic/Stochastic Model (HSM)
 - Sinusoidal Parametrization
- Glottal modeling:
 - **GlottHMM** [49]
 - Glottal Post-filtering
 - Glottal Spectral Separation

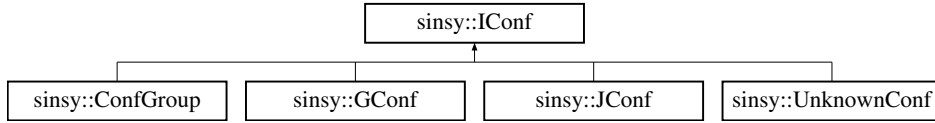


Figure 2.24: Inheritance diagram for sinsy::IConf

- Separation of Vocal-tract and Liljencrants-Fant model plus Noise (SVLN)

The **pulse vocoder** uses a simple source-filter model where excitation is modeled as Dirac pulse for voiced signals and white noise for unvoiced signals. The filter uses MGC coefficients [46]. This is the type of vocoder that we also use in our experiment in Chapter 3 since it is part of the SINSY [3] system and it is open-source.

The study in [45] showed that high F0 values create problems for all types of vocoders such that perceptual preferences are statistically insignificant for singing voices. This suggests that all vocoders need to be improved for singing synthesis.

2.6.6 HMM-based SINGing voice SYnthesis system (SINSY)

The work in this thesis is based on SINSY version 0.90 released on 25 December, 2013. At the same time also a Japanese HTS voice version 0.90 was released, which is the basis for our voice development. The architecture of the system is shown in Figure 2.21. As input the system accepts MusicXML. Supported musical symbols are tie, slur, staccato, accent, dynamics, crescendo, decrescendo, and breath mark.

SINSY is written in C++. The handling of different languages is done by extending the sinsy::IConf class. Figure 2.24 shows the inheritance diagram for the IConf class. sinsy::JConf handles the conversion of Japanese MusicXML data. sinsy::GConf was added by us to support the conversion of German data. It will be described in more detail in Chapter 3. Appendix A shows the context dependent label format used in SINSY.

The HTS singing voice version 0.90 was released to support the development of new voices for SINSY. It contains labels, clustering question, and wav files for Japanese as well as scripts for feature extraction and training. The training process uses Makefiles. Appendix A shows the list of features that are used for clustering.

The feature extraction consists of the following steps:

1. Extracting Mel Generalized Cepstral Coefficients (MGC) or MGC-Line Spectral Pair (LSP) coefficients from raw audio
2. Extracting log F0 sequence from raw audio
3. Composing training data files from MGC and log F0 files

4. Generating monophone and full-context Master Label File (MLF)
5. Generating a full context model list file
6. Generating a training data script

The training process for a singing voice consists of ≈ 30 steps where there are several loops of embedded estimation (reestimation), which was explained in Section 2.1 and context clustering (explained in Subsection 2.3.2) followed by synthesis steps (explained in Subsection 2.3.4) with different models. The models also include estimation of global variance [50] and estimation of semi-tied covariance matrices [51].

2.7 Musical Instrument Digital Interface (MIDI)

MIDI (short for Musical Instrument Digital Interface) is a technical standard that describes a protocol, digital interface and connectors and allows a wide variety of electronic musical instruments, computers and other related devices to connect and communicate with one another. A single MIDI link can carry up to sixteen channels of information, each of which can be routed to a separate device.

MIDI carries event messages that specify notation, pitch and velocity, control signals for parameters such as volume, vibrato, audio panning, cues, and clock signals that set and synchronize tempo between multiple devices. These messages are sent to other devices where they control sound generation and other features. This data can also be recorded into a hardware or software device called a sequencer, which can be used to edit the data and to play it back at a later time [52].

The MIDI format offers a compression of the musical data and can be used for aligning musical scores with audio data. In our work we translate the MusicXML files into MIDI format, which contains concrete timing information. The MIDI format from the MATLAB [53] Midi Toolbox [54] that we are using contains the following information for each note: onset (in beats), duration (in beats), MIDI channel, MIDI pitch, velocity, onset (in seconds), and duration (in seconds). In the case of the singing voice part the notes are not overlapping. The MIDI notes generated from the transcription can then be aligned to a real musical performance using dynamic programming, which can help us in the alignment of our data.

2.8 Alignment of MIDI data and audio

For alignment of MIDI files and recordings we used a method presented in [55]. We use a MATLAB implementation that can be found on [56]. For the alignment the

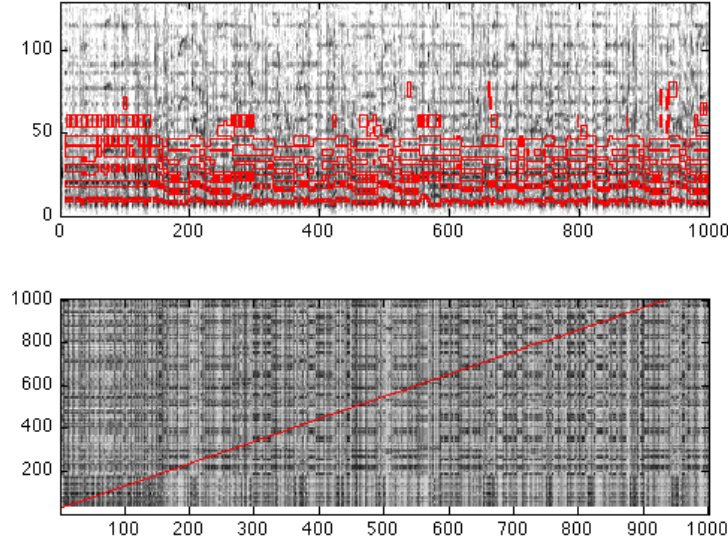


Figure 2.25: Masking of MIDI file (top). DTW alignment (bottom) (Figure from [56]).

spectrum of the MIDI file is masked to find the cells that contain the most energy, which is shown in Figure 2.25 at the top. Then DTW is used to align the masked spectrum of the MIDI file with the spectrum of the audio file, which is shown in Figure 2.25 at the bottom. By knowing the borders of notes from the MIDI file we can find the borders of the notes in the original recording using the alignment. In this way we get a transcription of the original audio. We can also create a new MIDI file with the note durations from the original audio file.

Algorithm 2 shows the basic dynamic time warping algorithm. DTW is also the simplest speech recognition method that uses dynamic programming to compare a reference speech sequence with the speech sequence that one wants to recognize (template matching). It was used for early speech recognition on mobile phones. For recognition one has to record reference utterances that are then matched against newly recorded utterances. The advantage of the method is that it is language and speaker independent, one algorithm works for all languages and all speakers, and that no acoustic model and no language model is needed.

The $D[i, j]$ holds the cost matrix, which is computed for all i, j pairs. In heuristic versions of the algorithm the computation of the cost can be restricted to a band along the diagonal. First the cost matrix is initialized. Then we have two for loops computing the cost for the remaining cells. In our case the cost function $cost(s[i], t[j])$ is the spectral difference between MIDI and audio at point i, j .

Algorithm 2 DTW algorithm returning the distance between a source $s[1, \dots, n]$ and target sequence $t[1, \dots, m]$ ($D[0..n, 0..m]$).

```

1: for  $i \leftarrow 1, n$  do
2:    $D[i, 0] = \infty$ 
3: end for
4: for  $i \leftarrow 1, m$  do
5:    $D[0, i] = \infty$ 
6: end for
7:  $D[0, 0] = 0$ 
8: for  $i \leftarrow 1, n$  do
9:   for  $j \leftarrow 1, m$  do
10:     $D[i, j] = \text{cost}(s[i], t[j]) + \min(D[i-1, j], D[i, j-1], D[i-1, j-1])$ 
11:   end for
12: end for
13: return  $D[n, m]$ 

```

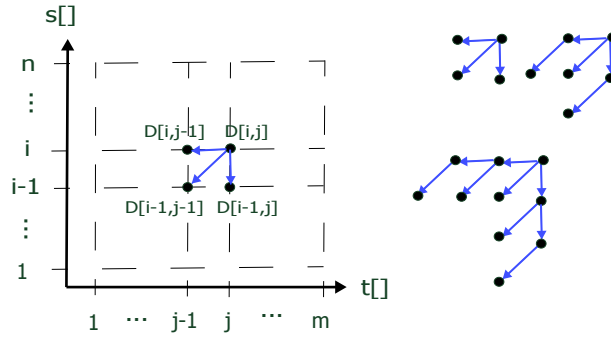


Figure 2.26: Different pattern for computing the cost in DTW.

Figure 2.26 shows different cost patterns that can be used in DTW. In the implementation that we are using the basic pattern at the left is used. For computing $D[i, j]$ we add the cost between $s[i]$ and $t[j]$, $\text{cost}(s[i], t[j])$ and the minimum of $D[i, j-1]$, $D[i-1, j]$, and $D[i-1, j-1]$. Other patterns shown in Figure 2.26 on the right can be used to cover longer distance dependencies between the compared sequences. At the end the DTW algorithm shown in Algorithm 2 returns the total cost between the two sequences, which is found at $D[n, m]$. This cost can be used for the recognition task.

For alignment we need to find the path of the lowest cost called the warping path. An example alignment is shown in Table 2.2 to demonstrate this. We see a two sequences s of length 10 and t of length 7 that are to be aligned. The elements of s

Table 2.2: Example DTW alignment between two sequences.

$s[10]$	2	∞	16	17	8	7	12	9	10
$s[9]$	4	∞	12	13	6	6	9	11	13
$s[8]$	3	∞	10	10	6	5	10	10	11
$s[7]$	5	∞	7	6	5	7	9	9	13
$s[6]$	8	∞	6	4	7	11	6	12	17
$s[5]$	7	∞	4	3	6	8	6	10	15
$s[4]$	6	∞	3	4	4	5	5	9	14
$s[3]$	4	∞	3	4	2	3	7	9	12
$s[2]$	5	∞	1	2	2	4	7	10	14
$s[1]$	6	∞	0	1	3	6	8	12	17
		0	∞	∞	∞	∞	∞	∞	∞
			6	7	4	3	8	2	1
			$t[1]$	$t[2]$	$t[3]$	$t[4]$	$t[5]$	$t[6]$	$t[7]$

can be found on the second column (e.g. $s[3] = 4$), the elements of t on the row before the last row. As a cost function we use the Euclidean distance between the elements, i.e. $cost(s[3], t[5]) = cost(4, 8) = 4$. Using this cost function we compute all elements of the cost matrix $D[i, j]$, which are shown in Table 2.2. After this we can find the warping path by backtracking from the last element $D[n, m]$, which is $D[10, 7]$ in our case by always looking for the minimal cost cell in the comparison pattern $D[i, j - 1]$, $D[i - 1, j]$, and $D[i - 1, j - 1]$. This minimal element is then the next element in the warping path. It can happen that there are multiple minimal elements. In that case we can take one of the minimal elements.

3 A Hidden-Markov-Model (HMM) based opera singing synthesis system for German

In this chapter we will describe how we extended an existing Japanese singing synthesis system [3] for the German language. Our German singing synthesis system is comparable with the current state-of-the-art for English [44] and is able to create German full context labels with German features like stress and word boundaries, duplicate syllables to deal with slur as described in Subsection 2.6.4, and do utterance chunking of MusicXML files.

The opera data was recorded in Vienna in a project funded by the National Institute of Informatics (NII), Japan [57]. In this project we recorded the four main singer types (mezzo, soprano, tenor, and bass). We will also describe the recording process and methods.

Furthermore we will also describe the development pipeline for creating an opera singing voice for that system. Here we will describe specific alignment and training scripts for acoustic models of opera singing that we developed.

3.1 Recording

The opera data was recorded in Vienna in a project funded by the National Institute of Informatics (NII), Japan [57]. In this project we recorded the four main singer types (mezzo, soprano, tenor, and bass).

3.1.1 Singer and song selection

For the recordings we consulted a professional opera singing teacher that did the concrete selection of songs and singers. Differently to speech recordings the selection of songs and singers for opera synthesis is tightly coupled. In standard speech synthesis we would select a corpus with an optimal phone coverage by finding an approximate solution for the associated minimum set-cover problem [15]. For the solution of this problem we can take different features like diphones, diphones in stressed syllables and so on into account [58]. For finding this corpus we need a large phonetically transcribed background corpus. This corpus would then be read by a speaker that is selected in a separate selection process.

For selecting an opera corpus we could go the same way, provided that we have a large amount of opera songs in MusicXML format. But with such a selection process we would end up with a selection of opera songs that no available opera singer has

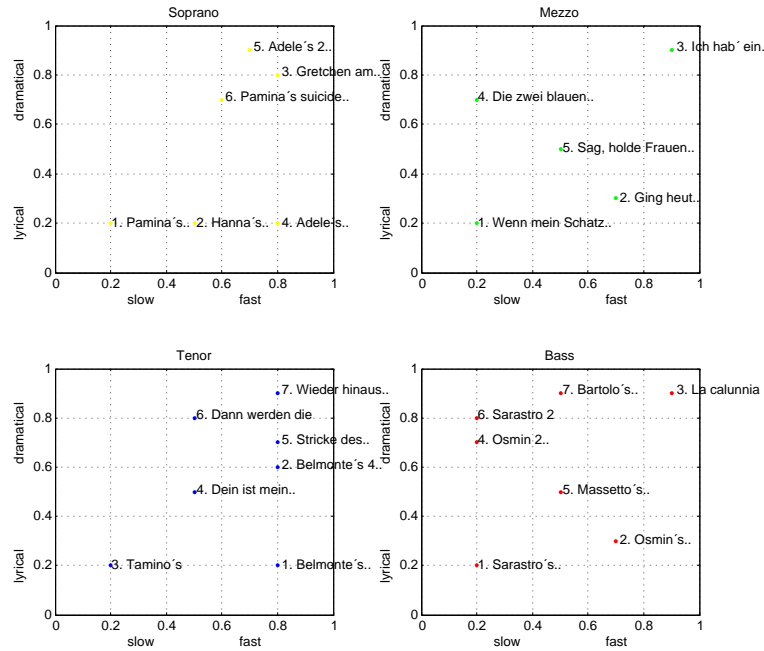


Figure 3.1: Classification of opera songs according to lyrical - dramatical and slow - fast dimension.

in his/her repertoire at the moment. So the selection of singers and songs has to go hand in hand by using a different strategy.

Therefore we decided to select a number of opera songs ($\approx 8-10$) for each singer category that are in the repertoire of that singer at the moment and that cover the space of opera songs along the lyrical - dramatic and slow - fast axis. We also checked that these songs cover the F0 range of that singer category.

Figure 3.1 shows the classification of opera songs along the two dimensions lyrical - dramatic and slow - fast. The classification of opera songs was done by a professional opera singing teacher. As can be seen in Figure 3.1 the opera songs that were then recorded cover the whole space for mezzo and bass but the slow and dramatic category is in general difficult to find and especially difficult for the soprano and tenor voice. This of course also shows that the two dimensions are not completely independent of each other. A further restriction in the selection of songs was the fact that we were only looking at songs in the German language.

Figure 3.2 shows the F0 range for the mezzo voice with bold numbers on the piano roll ranging from A3 with 220 Hz to A5 with 880 Hz. The colored bars show the F0 ranges for our 8 selected opera songs. Song 1 for example has a range from B3 (=246 Hz) to G5 (=783 Hz). We can see from Figure 3.2 that our songs cover the F0 range almost completely with the exception of one note, the highest note with 880 Hz, which

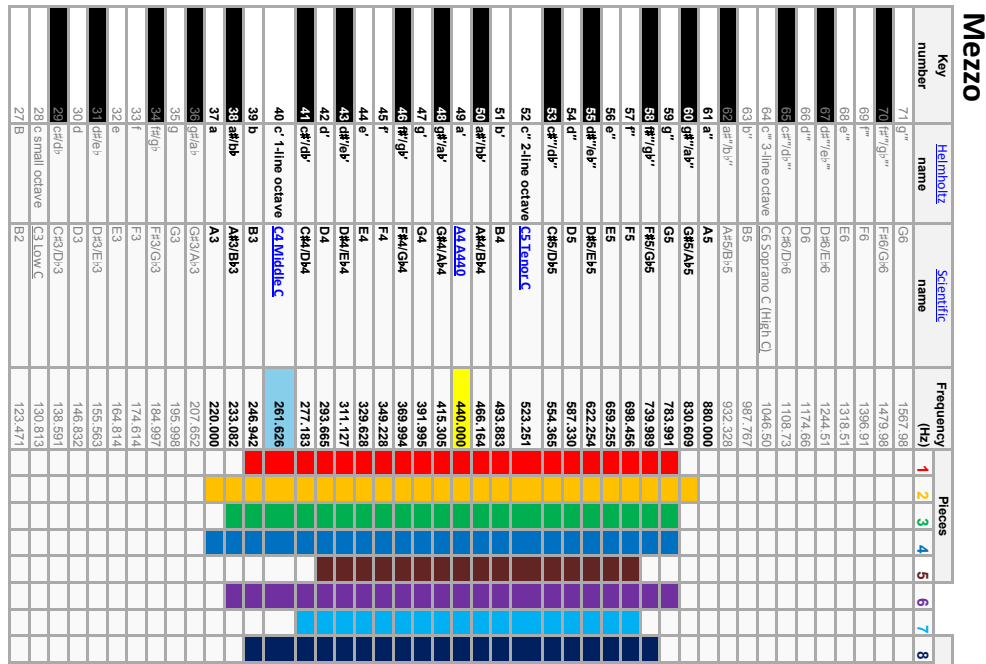


Figure 3.2: F0 range for mezzo and opera songs shown on the piano roll.

is not existent in the training data.

3.1.2 Phonetically balanced singing corpus

Additionally to the coverage of the features F0, lyric - dramatic, and slow - fast we were also thinking about how to achieve a good phonetic coverage. In speech synthesis phonetic coverage is achieved by transforming a large text corpus into phone sequences and then selecting those sentences from the corpus that achieve the best phonetic coverage in terms of number of diphones or other contextual factors. This is a set-cover problem [15].

In the opera singing context this could not be done since we did not have a large MusicXML corpus to select from, and we also had to consider the constraint of the repertoire of our singers. To still achieve phonetic coverage we recorded a sung version of an existing German phonetically balanced corpus. This corpus consists of approximately 200 sentences.

Each singer had to improvise a melody for a certain sentence together with the piano player. Then they performed the sentence together. These sentences were however not used in the modeling and experiments described in Chapter 3-4. For using them we would need to derive the MusicXML transcription from the audio. This is possible

in principle, but is an error prone task that needs manual correction. It adds an additional layer of complexity to the already difficult task of opera synthesis. If this corpus is however once transcribed in MusicXML there is no problem to include it in the training data.

3.2 Implementation of a German frontend for Sinsy

The main extensions to the SINSY system for German were

- analysis of the German input text (text analysis),
- conversion of the input words into phonetic sequences (lexicon and letter-to-sound conversion)
- duplication of syllables where syllables had more than one note (syllable duplication)

3.2.1 Text analysis

In TTS systems the task of text analysis consists of the conversion of numbers, dates etc. into a form that is close to written words (123 \rightarrow hunderteinundzwanzig). In parsing the data from the MusicXML file the task of text analysis consists in the reconstruction of words that can then be used to access the lexicon. The `<begin>`, `<middle>`, and `<end>` tag inside the MusicXML `<lyric>` tag are used to mark the specific syllables of the word. The `<single>` tag marks a word with just one syllable.

3.2.2 Lexicon and letter-to-sound conversion

We integrate a lexicon and rules for Letter To Sound (LTS) conversion from an open-source synthetic voice for Austrian German that was developed at Telecommunications Research Center Vienna (FTW) [59]. The LTS rules consist of a set of decision trees, one tree for each letter, that are used to convert a given input character sequence (word, syllable) into the corresponding output phones as described in Subsection 2.2.4.

Since we are getting a sequence of syllables from the MusicXML files we are using these syllables directly for LTS conversion. For each syllable we are using the decision trees for predicting the corresponding phone sequence. The other approach would be to put syllables together into words, apply LTS conversion to words and split the resulting phone sequence again into syllables. By skipping the syllabification step, where a sequence of letters is broken up into syllables we achieve a more robust prediction. Furthermore there are sometimes MusicXML files where the annotation is wrong, such that merging syllables into words leads to wrong or non-existing words. With our method that starts directly from syllables, we can also alleviate this problem.

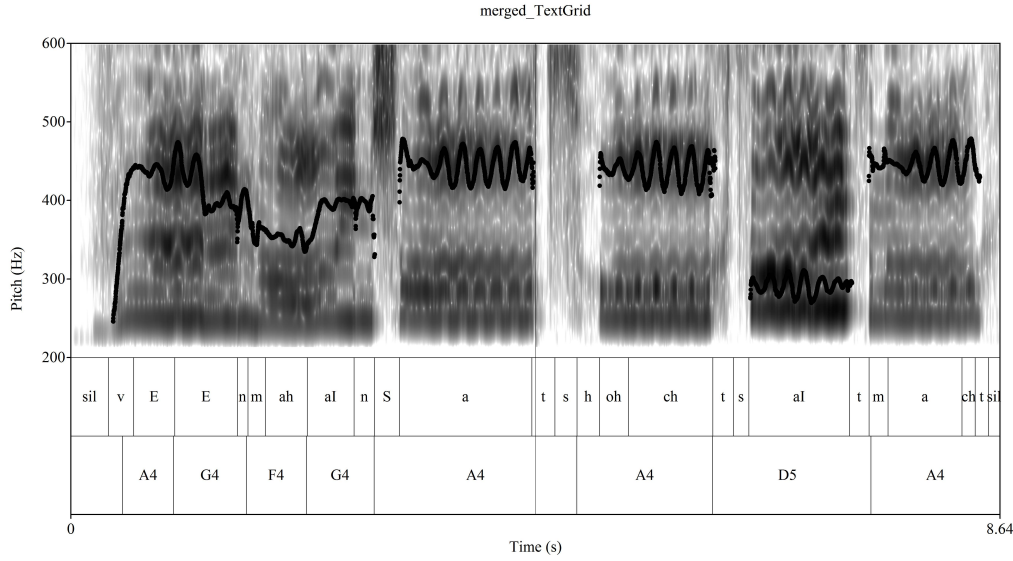


Figure 3.3: Alignment of MIDI and phone labels on utterance level for the utterance “Wenn mein Schatz Hochzeit macht”.

Through the integration of LTS rules into the system we are able to synthesize from any German MusicXML file.

3.2.3 Syllable duplication

Syllable duplication that was already described in Subsection 2.6.4 was not implemented in the open-source SINSY system, so we had to do this for German. Figure 3.3 shows an example alignment for the sentence “Wenn mein Schatz Hochzeit macht”, phonetically “v E n . m a l n . S a t z . h o h c h t s a l t . m a c h t” where ‘.’ signifies word boundaries here. The alignment also contains the silence symbol “sil” at the beginning and end of the utterance.

At the top we can see the spectrogram of the respective audio signal as well as the F0/pitch curve in Hz. Below we have the phonetic transcription aligned and the musical notes aligned. We see that the notes start with A4, which has 440 Hz. The F0 curve shows that this target is reached with vibrato around 440 Hz.

The transcription shown here is already after text analysis and letter-to-sound conversion. We can see that the word “wenn” (“v E n”) is distributed across two notes A4 and G4 such that the syllable duplication turns it into “v E E n” phonetically. The word “mein” (“m a l n”) is also distributed across two notes F4 and G4. Through syllable duplication it is turned into “m ah a l n” where we take into account that we do not duplicate diphthongs like “al”. The syllable duplication algorithm takes the

MusicXML file as input and transform it into a phone sequence as shown in Figure 2.23.

```

<note>
  <pitch>
    <step>F</step>
    <octave>4</octave>
  </pitch>
  ...
  <notations>
    <slur number="1" type="start" />
  </notations>
  <lyric number="1">
    <syllabic>single</syllabic>
    <text>mein</text>
  </lyric>
</note>
<note>
  <pitch>
    <step>G</step>
    <octave>4</octave>
  </pitch>
  ...
  <notations>
    <slur number="1" type="stop" />
  </notations>
</note>

```

The above MusicXML example shows how the `<slur>` tag is used to define that a certain word spans multiple notes. In this case the word “mein” spans A4 and G4. Using this file and the phonetic transcription of the word we have to distribute the phonetic syllables to the notes.

For the duplication of syllables with diphthongs we use the duplication rules shown in Table 3.1. When duplicating an “aI” n times for example, we generate $n - 1$ times an “ah” (a long “a”) followed by one “aI”. Prefix and postfix are used from the original

Table 3.1: German diphthong duplication rules.

Original	aI	aU	E6	Eh6	ih6	O6	OY	Y6
Duplicated	ah, aI	ah, aU	E, E6	E, Eh6	ih, ih6	O, O6	O, OY	Y, Y6

syllable. Our notation is closely related to the Speech Assessment Methods Phonetic Alphabet (SAMPA) standard [60].

Algorithm 3 shows the algorithm for syllable duplication. After preprocessing we have inserted pause symbols “pau” at the phones inside of slurs. For the example of the word “mein” mentioned above the sequence after preprocessing is “m aI pau n”. The algorithm now goes through all syllables and phones within a slur (slur begin to slur end). The first vowel is replaced if it is a diphthong, so the sequence is “m ah pau n” after this step. If we are at the last pause in the slur, which we can check by checking if we are at the last note, we replace the pause by the respective diphthong. This leaves us with the final sequence “m ah aI n”, which is the desired result.

Algorithm 3 Algorithm for duplicating syllables.

```

Check that slur does not span across words
for note  $\leftarrow$  slur begin, slur end do
  for syl  $\leftarrow$  syl begin, syl end do
    for phone  $\leftarrow$  phone begin, phone end do
      if phone is the first vowel found then
        Replace phone if it is a diphthong
      else
        if phone is a pause after vowel was found then
          if we are not at the last pause in the slur then
            Replace pause by respective phone
          else
            Replace pause by diphthong or respective phone
          end if
        end if
      end if
    end if
  if last note and last syllable in slur then
    Add remaining phones
  end if
end for
end for
end for

```

At the beginning the algorithm also checks if the slur does not span across words. In this case we cannot do syllable duplication with our algorithm. The algorithm does

of course also work is we have more than two notes that are to be distributed across a syllable. If we have to sing “m aI n” with four different notes, the conversion would be from “m aI pau pau pau n” to “m ah ah ah aI n”.

3.3 Alignment

3.3.1 Conversion between notes, midi notes, and frequencies

For the alignment of MIDI, waveforms, and labels we need to be able to convert between different symbolic representations, which requires the conversion between notes, MIDI notes and frequencies. MIDI notes are named from 0 to 127 (60=C4, 61=C#4, 62=D4,...,69=A4=440Hz). An octave contains 12 semitones.

Algorithm 4 Algorithm for creating a mapping between notes and MIDI notes.

```

notes = “C DbD EbE F GbG AbA BbB ”
for notenum  $\leftarrow$  0, 127 do
    octave = notenum / 12 - 1
    note = notes[(notenum % 12) * 2:(notenum % 12) * 2 + 2]
    notename = note+str(octave)
    notename = notename.replace(“ ”, “_”)
    note_midi_map[notename] = notenum
end for

```

Algorithm 4 converts MIDI notes in “notenum” into the respective notes. The general formula for computing the frequency from the MIDI notes is given as

$$f_n = f_0 * a^n \quad (3.1)$$

where f_0 is a fixed frequency of a given note, which is in our case fixed to A4=440 Hz, which has the MIDI number 69. n is the number of semitones you are away from the fixed note, which is either positive or negative. $a = 2^{\frac{1}{12}} = 1.059463094359....$

Algorithm 5 Function for computing the frequency from MIDI notes.

```

function MIDINOTE_TO_FREQUENCY(midinote)
    notediff = midinote - 69
    freq = 440 * 1.059463094359notediff
    return freq
end function

```

Table 3.2: *Alignment methods for aligning original recordings with MIDI files.*

p-sp	Piano MIDI aligned with Singing+Piano original
p-p	Piano MIDI aligned with Piano original
p-s	Piano MIDI aligned with Singing original
s-sp	Singing MIDI aligned with Singing+Piano original
s-p	Singing MIDI aligned with Piano original
s-s	Singing MIDI aligned with Singing original
sp-sp	Singing+Piano MIDI aligned with Singing+Piano original
sp-p	Singing+Piano MIDI aligned with Piano original
sp-s	Singing+Piano MIDI aligned with Singing original

3.3.2 Aligning waveforms and midi data

For aligning waveforms and midi data we use the algorithm implemented in [56] that was already described in Section 2.8. For the alignment we generate a MIDI file from the MusicXML file using MuseScore [32], which is then aligned with the original audio recording.

We are interested in an alignment of the opera singing, i.e. finding the borders of notes in the audio signal. For this alignment we can use different MIDI data coming from the piano notes, the singing notes, or both. In terms of the audio signal that we want to align, we can align the singing audio or the audio that contains singing and piano performance. Table 3.2 shows all possible alignment methods where we are only interested in methods having “s” or “sp” at the right side. These methods provide an alignment of the singing signal.

Figure 3.4 shows the performance of the different alignment methods on a whole mezzo song. We can see that there is some disagreement between the alignment methods. For our alignment on the utterance level we are using the “s-s” method, since we have only MusicXML transcriptions of the singing part for some of the mezzo songs and have seen a similar alignment performance of “s-s” and “s-sp” which would be our two options. A formal evaluation of different alignment methods would be very interesting but is beyond the scope of this thesis.

3.3.3 Splitting opera recordings into utterances

Whole opera songs are split into smaller utterance chunks to make the alignment more robust and acoustic feature extraction computationally less complex. The waveforms are cut manually into utterances and MusicXML files are annotated with “uttbegin” and “uttend” labels.

The MusicXML files are then split into utterance level files where we have to make the following adjustments:

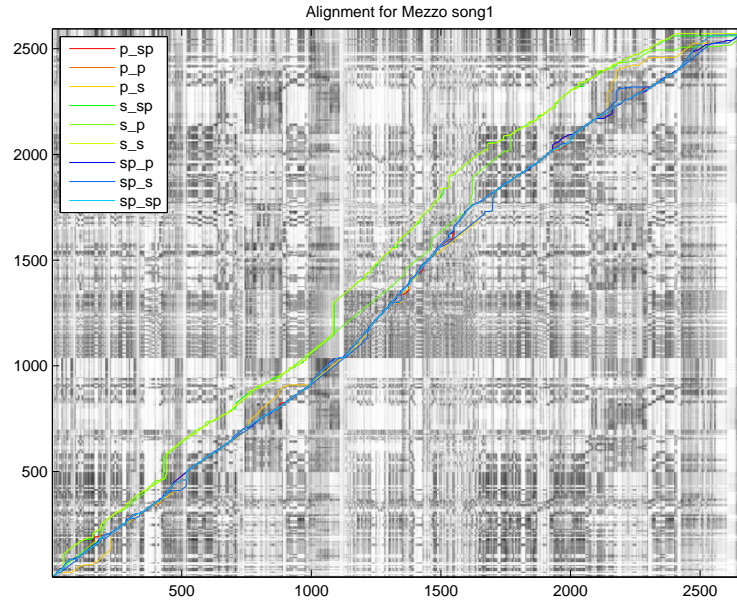


Figure 3.4: Alignment of original mezzo Song #1 with MIDI song.

- Introduce <rest> before the first syllable, and after the last syllable.
- Remove syllables that don't belong to the utterance.
- Introduce the correct attributes at the beginning of the utterance MusicXML file.

The attributes in the MusicXML files define the tempo and beat information as well as the key of the notes.

```
<attributes>
  <divisions>24</divisions>
  <key>
    <fifths>-3</fifths>
    <mode>minor</mode>
  </key>
  <time>
    <beats>3</beats>
    <beat-type>4</beat-type>
```

```
</time>  
</attributes>
```

3.3.4 Alignment of singing speech and labels

Using the German frontend for SINSY that was described in Section 3.2 we can transform utterance MusicXML files into full context label files that can be used for HMM training. The format of full context labels is described in Appendix A. The context contains phonetic and linguistic information as well as information on the current, previous and following notes.

Using the program Musescore [32] we can also create MIDI files from the utterance MusicXML files. These MIDI files are then aligned with the original opera singing waveforms using the method described in Section 2.8.

These aligned MIDI files are then used to set note durations in the full context label files. The durations within are set uniformly. If we have for example the note A4 with duration 0.3 seconds for the syllable “b e r” we set the duration for each phoneme to 0.1 seconds.

This alignment provides us with a first rough alignment of the data at the level of notes and uniform alignment at the level of phones that is then manually corrected for the mezzo voice to have correct alignments at the phone level as shown in Figure 3.4. We use this alignment also for the evaluation of the mezzo voice. Fully automatic alignment with monophone HMMs is done for the voices that are only used in training and are not used in the evaluation (soprano, tenor, bass).

3.4 Training of acoustic models

3.4.1 Data

For training the mezzo voices we had 8 different opera songs. After splitting the recordings into utterances we had 154 different utterances. 8 utterances were taken as test sentences and were not used for training. As shown in Table 3.3 we had songs from five different composers with a total duration of 25.8 minutes.

3.4.2 Training

For training acoustic models for opera singing we adapted an existing training script for Japanese acoustic model training [61] that was released in December 2014. The model training follows the speaker dependent singing synthesis system as shown in Figure 2.21.

To adapt the training script for German we had to generate clustering questions for German. Using the clustering questions from a speech synthesis training script for

Table 3.3: Songs recorded for the mezzo voice. The table also shows the maximum and minimum F0 according to the MusicXML file.

Song	Composer	Singer	Dur. (sec.)	Max. F0	Min. F0
Wenn mein Schatz Hochzeit macht	G. Mahler	Mezzo	214	246	783
Ging heut abend übers Feld	G. Mahler	Mezzo	235	220	783
Ich hab ein glühend Messer	G. Mahler	Mezzo	201	246	783
Die zwei blauen Augen	G. Mahler	Mezzo	318	220	783
Sagt, holde Frauen	Mozart	Mezzo	170	261	698
Ich wünsche dir Glück	Korngold	Mezzo	142	220	698
Sandmanns Arie	Humperdinck	Mezzo	140	261	879
Laue Sommernacht	A. Mahler	Mezzo	128	246	698
8	5	1	1.548 25.8 min.		

German [59] and adopting it to the Japanese singing training script we generated a training script.

Figure 3.5 shows a part of the decision tree for the log F0 for the center state of the HMM. This part shows the decision tree for the O-vowels were the first two questions are if the current phone is a vowel and if the current phone is an O-vowel. We can see that the F0 for O-vowels is clustered according to the current note and some linguistic features. The data dependency of this approach is a weakness here in case that we have to generate an F0 that was not present in the training data. However, we are able to cover some liveliness of the F0 variation as compared to the approach where we simply take the fixed notes as F0 values. The tree also shows that we can only model 14 different notes for the O-vowels, while there are 24 notes in the mezzo range which can be seen from Figure 3.2 and Table 3.3. Our modeling of the O-vowels only covers 58% of the notes, which is a problem with this data dependent modeling.

Figure 3.6 shows part of the decision tree for the spectral models of vowels where the current phones absolute scale is smaller than C5. We can see that in this decision tree more linguistic phonetic classes are used, although still not all vowels are covered, which also relates to the rather small amount of training data.

Figure 3.7 shows a part of the decision tree for duration modeling for vowels that have a duration of less than 40 centiseconds (10^{-2} seconds). This decision tree also shows a mix of linguistic and score related features, where duration features are located at the top nodes of the tree. The first question leading to this subtree is if the current

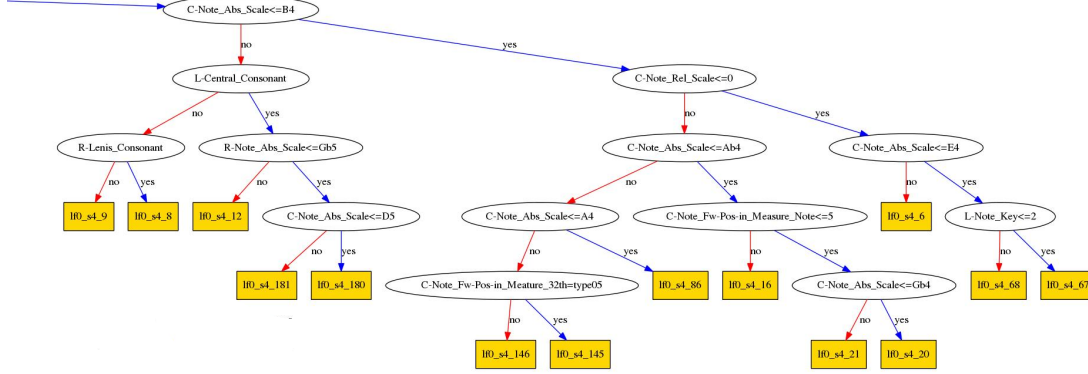


Figure 3.5: Part of the decision tree for log F0 models for the center state (4th state of 7-state) of the HMM.

phone is a vowel, the second question is if the note has duration of less than 40 centiseconds (10^{-2} seconds).

The training script [61] allows for the modification of different parameters. In our experiments we tried spectral estimation with two different FFT lengths 2048 and 4096. A longer FFT analysis window increases spectral resolution and decreases time resolution.

Furthermore we added the F0 extraction method YIN [62] to the training script by using the Matlab implementation from [63]. The name YIN refers to the “yin and yang” since the algorithm uses autocorrelation and cancellation. The Robust Algorithm for Pitch Tracking (RAPT) [64] F0 extraction method that is part of the script and implemented in the Signal Processing ToolKit (SPTK) [65] was also used.

For F0 extraction we determined the maximal F0 value by extracting it from the corresponding MusicXML file. This constraining of the extraction improves the performance of the F0 extraction significantly.

We also implemented F0 shifting as described in Subsection 2.6.2. For F0 shifting we use Sox [66] to increase or decrease the fundamental frequency of an opera singing utterance by one semitone. We also shift the notes in the full context label files by one semitone up or down. In this way we are able to triple the size of the training corpus. The models that are trained with this modified script can be used with our German version of the SINSY singing synthesis system.

For the adaptation of the training script we did the following:

- Creation of clustering questions for German.

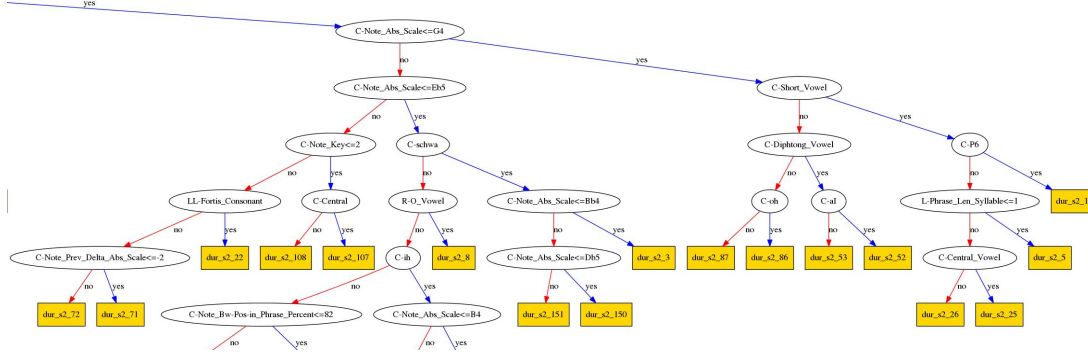


Figure 3.7: Part of the decision tree for duration models for the center state (4th state of 7-state) of the HMM.

Robust Algorithm for Pitch Tracking (RAPT)

[64] gives an overview of F0 extraction methods and also describes the RAPT algorithm. These extraction methods are also often called pitch extraction methods, although the term “pitch” refers to the perceived tone that strongly correlates with F0 but is a “nonlinear function of the signal’s spectral and temporal energy distribution” ([64],p.497). The pitch of a complex sound can be measured by letting listeners find the sinusoid with the same tone. According to [64] F0 extraction methods often perform three steps:

1. Pre-processing (Low-pass filtering etc.).
2. Extraction of F0 candidates for frames.
3. Selection of best F0 candidate for each frame.

The RAPT algorithm does not use any pre-processing. F0 candidates are extracted by defining an F0 range and using the normalized cross-correlation function, which is defined as

$$\Phi_{i,k} = \frac{\sum_{j=m}^{m+n-1} s_j s_{j+k}}{\sqrt{e_m e_{m+k}}} \quad k = 0, \dots, K-1, m = iz, i = 0, \dots, M-1 \quad (3.2)$$

where e_j is defined as

$$e_j = \sum_{l=j}^{j+n-1} s_l \quad (3.3)$$

and i is the frame index for M frames, k is the lag index, and $z = t/T$ with $T = 1/F_s$. $n = w/T$, and W is twice the longest expected glottal period. The signal s is assumed to be zero outside the window w . Values of k where $\Phi_{i,k}$ are close to 1.0 are candidates for the F0 of frame i .

In the RAPT algorithm $\Phi_{i,k}$ is first computed for a speech signal at reduced sampling rate to reduce computational cost. Then $\Phi_{i,k}$ is computed again on the signal with original sampling rate in the neighborhood of the best estimates from the first step. Then dynamic programming is used to select the best F0 and voicing state candidates, where each frame is either marked as voiced or unvoiced, where unvoiced frames have by definition no F0.

YIN F0 extraction

The YIN F0 extraction method [62] is also based on the cross-correlation function defined in Equation 3.2 with a number of modifications to prevent errors that are commonly made by the cross-correlation method.

3.5 Voice development pipeline

Figure 3.8 shows a system diagram for the voice development. Each block receives data with certain input format shown on the lines entering the block and produces data in a certain output format. The formats are waveforms (WAV), MusicXML files, MIDI files, and full context and monophone label files (LAB) in SINSY format. The full context label format is described in Appendix A, the monophone labels contain the phone symbol and start and end time of the respective phone. Blocks in red do require manual intervention, the red block with dashed lines is optional.

After the recording process (Step 1) we have WAV [67] and MusicXML files of the opera songs. Then we have to set markers for begin and end of utterances in WAV and MusicXML files (Step 2), which is done with Audacity[68] and Musescore [32].

After that we can automatically cut the WAV and MusicXML files into utterance chunks (Step 3). Cutting the data fully automatically would be possible by using the MIDI alignment method from Step 6, but we choose the manual procedure to avoid any errors at this stage. Cutting the WAV data is done with Audacity, cutting the MusicXML is done with a Python [69] script that we've developed.

Then we can generate full context and monophone label files for the utterances from the MusicXML files using our German implementation of the SINSY [3] system. The generation of full context and monophone label files is done with the German SINSY system.

Using the MIDI files generated from the MusicXML files (Step 5) we can align the MIDI with the original recordings (Step 6) to get note duration information for our

recorded data. The MIDI generation is done with Muscore, the MIDI and WAV alignment is done with a Matlab script [54].

Using the aligned MIDI data and the generated label files we can generate new label files that include the time alignment information (Step 7). For the alignment of MIDI and label files we have developed a Python script.

Now we can optionally manually correct these alignments (Step 8) as done for the mezzo voice or do automatic alignment using monophone models (Step 9) as done for the soprano, tenor, and bass voices and start the training (Step 10) or directly start training with the MIDI aligned labels. The manual correction can be done with the Praat [70] software package.

For the training process we only need the original utterance WAV files and the label files. Furthermore we need the questions for clustering, which were derived from the original SINSY script and our Austria German voice [59].

The whole training and synthesis process described in this chapter is done with open-source or freely available software packages and software developed during the work on this thesis.

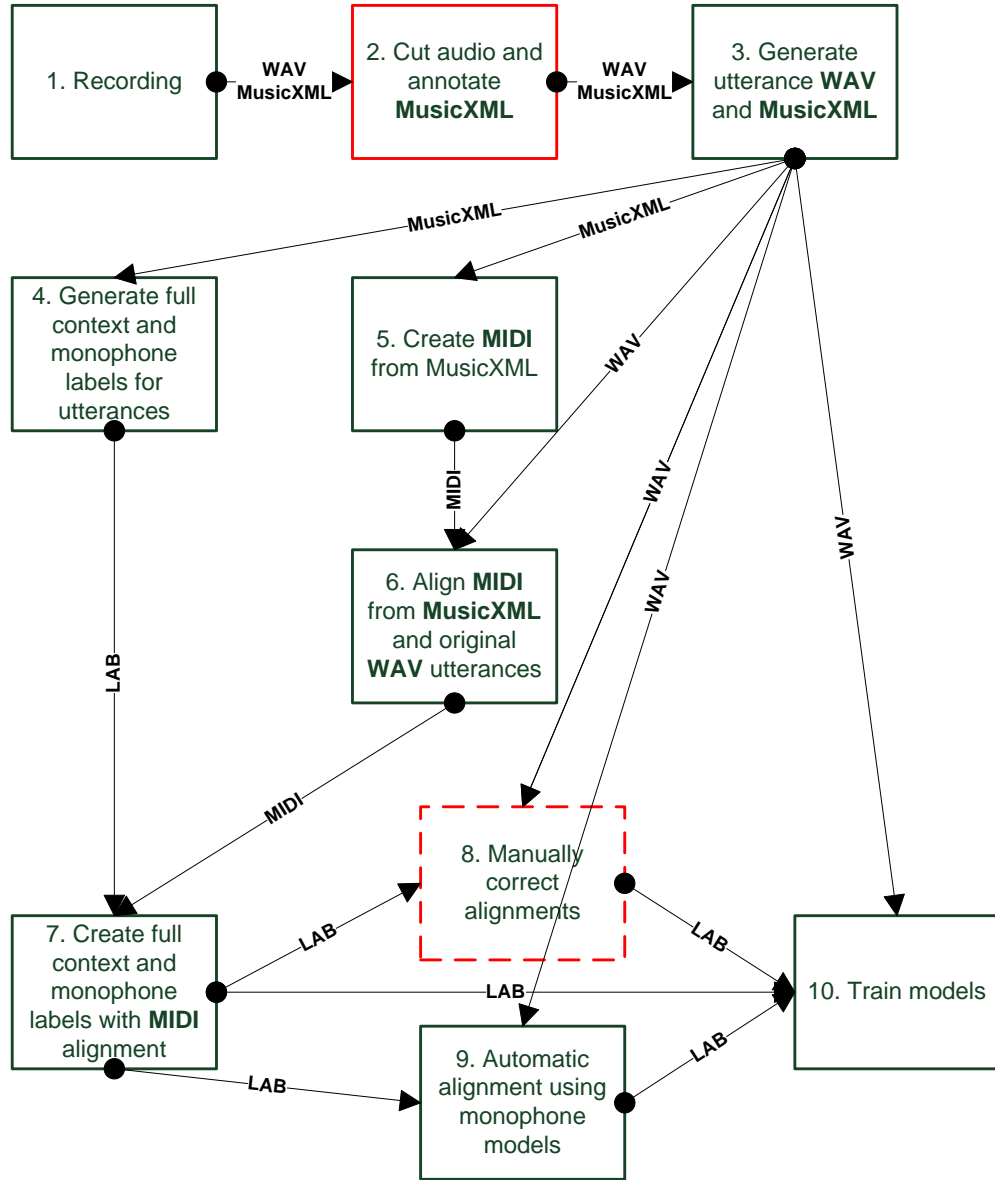


Figure 3.8: Voice development pipeline.

4 Evaluation

4.1 Different mezzo voices for evaluation

Table 4.1: Different parameters for the evaluation used in training.

Parameter	Values
FFT length	2048, 4096
F0 method	RAPT, YIN
F0 shifting	yes, no
Training data alignment	automatic, semi-automatic

Table 4.2: Different parameters for the evaluation used in synthesis.

Parameter	Values
Durations	SINSY prediction, MIDI prediction, original

Table 4.1 shows the different parameter combinations that we used in the evaluation for training different voices. We want to evaluate the influence of FFT length, F0 extraction method, F0 shifting and training data alignment on model training.

For training data alignment we used the automatically aligned labels from the MIDI alignment method as described in Section 3.3 as well as a set of labels where these automatically aligned labels were manually corrected, which we therefore call semi-automatically aligned labels.

By combining all possible parameters we have 16 different trained voices used in the evaluation.

Table 4.2 shows the different parameter combinations that we used in the evaluation for synthesizing. Durations for synthesis were taken from the SINSY prediction, the MIDI prediction or the original label files.

By using these 3 different synthesis labels and 16 different voices we get 48 versions for each test sentence. Each of these 48 synthetic versions of a test sentence is compared with the original recording using an objective error metric. Since we have 8 different test sentences (that were not part of training) we get 480 different synthetic singing samples.

4.2 Objective evaluation metric

As error metric between two waveforms we use Mel Cepstral Distortion (MCD) [71, 65], which is defined as follows

$$\text{MCD}(\mathbf{cep}_i - \mathbf{cep}_j) = \frac{10}{\ln(10)} \sqrt{2 \frac{1}{N} \sum_{k=1}^N (\mathbf{cep}_i(k) - \mathbf{cep}_j(k))^2} \quad (4.1)$$

where \mathbf{cep}_i and \mathbf{cep}_j are two sequences of cepstral parameters of length N .

This metric gives us the Mean Squared Error (MSE) of the cepstral parameters in Decibel (dB). To compute a distance between two waveforms we convert the waveforms into sequences of cepstral parameters, which are then time aligned using Dynamic Time Warping (DTW) and then compared using MCD. The time alignment is necessary since original and synthesized waveforms have different lengths for the synthesis based on SINSY prediction and MIDI prediction.

For the time alignment of feature sequences A (original) and B (synthesized) we compute the DTW path between the sequences as described in Section 2.8 which is a list of indices (i, j) of elements of A and B . To generate a sequence of elements of B with the same length of A we select the first element in B to which an element in A is mapped $\min_j(i, j)$.

4.3 Results of objective evaluation

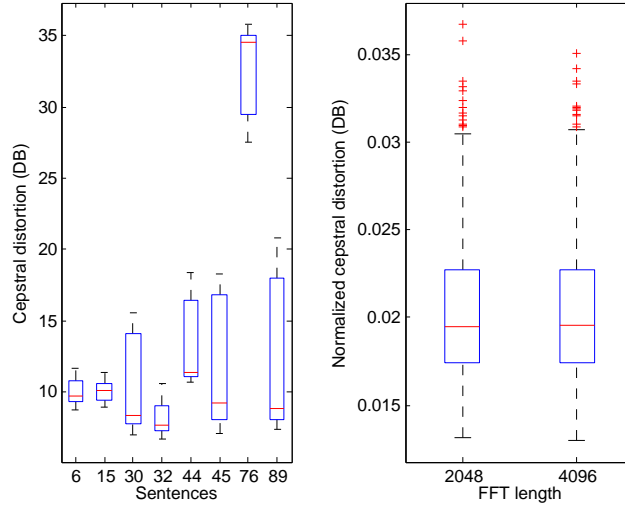


Figure 4.1: Cepstral distortion per sentence (left), normalized cepstral distortion for FFT length (right).

Figure 4.1 (left) shows the cepstral distortion for the 8 test sentences per sentence. We can see that there is one sentence that is particularly different from the respective original opera recording. Since we are only interested in differences between different methods, we normalized the cepstral distortion for all following comparisons.

Figure 4.1 (right) shows that there are no significant differences between training/synthesis method combinations when taking different FFT lengths (2048, 4096). A longer FFT length increases the spectral resolution and decreases the time resolution of the analysis.

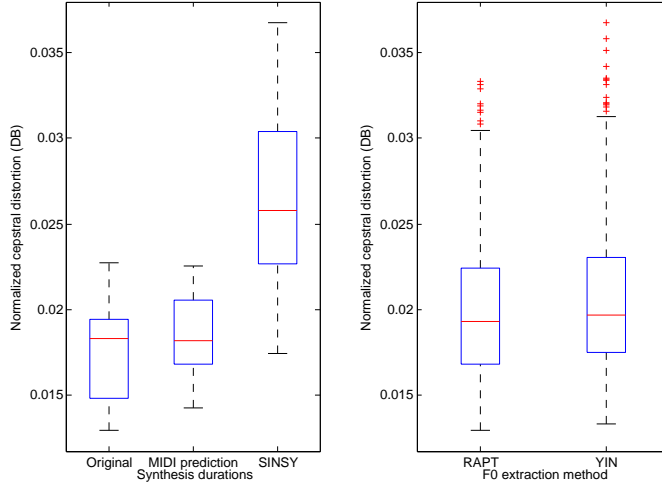


Figure 4.2: Normalized cepstral distortion for synthesis durations (left), normalized cepstral distortion for F0 extraction method (right).

Figure 4.2 (left) shows the normalized cepstral distortion when using different durations during synthesis time. The durations can come from the original label files that are generated through semi-automatic alignment, from the MIDI prediction where the recorded opera signal is aligned with a MIDI file, or from SINSY prediction where our German SINSY system is used to automatically generate full-context label files from MusicXML files.

Figure 4.2 (left) shows that there are significant differences between the different synthesis duration methods with $p < 0.05$ for the comparison between original and MIDI durations and $p < 0.001$ for the other two comparisons according to a Wilcoxon rank sum test. Not surprisingly we achieve the lowest error with the durations from the original label files, followed by synthesizing from MIDI predicted durations, and durations from SINSY prediction.

These significant differences show that the correct durations are essential for accurate singing synthesis. It is clear that the liveliness or originality of opera singing is

heavily influenced by the durations that a singer chooses for the different syllables. In the context of HMM modeling the correct durations also have a strong impact on the segmental quality and thereby the cepstral distance between original and synthesis.

Figure 4.2 (right) shows that there are weakly significant ($p < 0.15$) differences between the two different used F0 extraction methods RAPT and YIN where the RAPT method achieves a small improvement over the YIN method according to a Wilcoxon rank sum test.

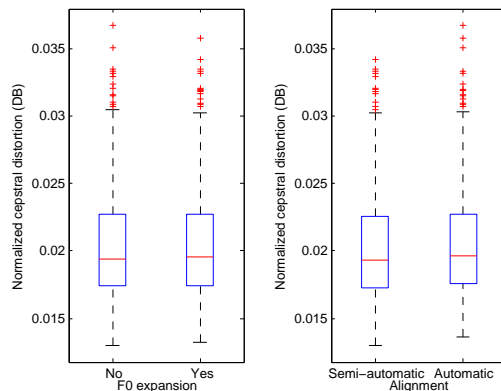


Figure 4.3: Normalized cepstral distortion for F0 expansion (left), normalized cepstral distortion for training data alignment method (right).

Figure 4.3 shows the results of the objective evaluation for the use of F0 expansion (left), and different training data alignment method (right). In F0 expansion we extend the training data corpus by additional data that is generated by increasing or decreasing the recorded samples by one semitone. Thereby we can triple the size of the training data. It shows however no significant differences in cepstral distortion.

Figure 4.4 shows the cepstral distortions for the different methods where a method is a training/synthesis combination. We can see some small differences, but none of them are significant.

Overall we can see that several modeling decisions have no influence on the objective quality of the synthesized samples. This has two reasons. The first lies in the small amount of data that we have available, and the second lies in the objective evaluation itself, which is a coarse method to evaluate quality differences in synthesis, especially when using only one scalar valued objective metric. Complex methods have been investigated for objectively measuring synthesis quality [72] but today still subjective methods are used [5] for evaluation of speech, although they are very time consuming.

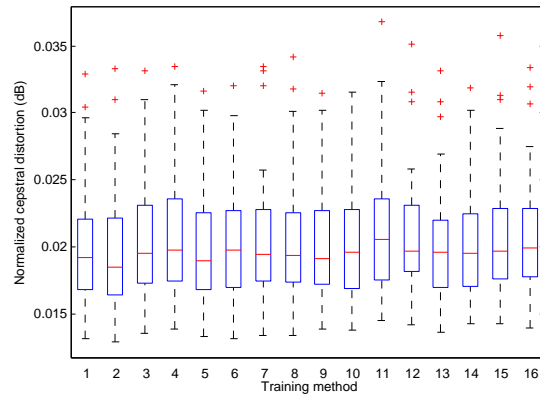


Figure 4.4: Normalized cepstral distortion for the 16 different methods (training/synthesis condition combinations).

4.4 Subjective evaluation

For the subjective evaluation we had 12 listeners that had to listen to pairs of synthesized samples and had to give a preference judgment on which sample they prefer in terms of overall quality. We did not evaluate how correct the synthesized samples are following the score.

For the subjective evaluation we only used 8 different methods out of the 16 methods to reduce the number of evaluation pairs. We only used the training methods that are using the semi-automatically aligned training labels. Table 4.3 shows the 8 different methods that were used in the subjective evaluation.

Table 4.3: 8 methods used in the subjective evaluation.

Param. \ Method	1	2	3	4	5	6	7	8
FFT length	2048	4096	2048	4096	2048	4096	2048	4096
F0 method	RAPT	RAPT	YIN	YIN	RAPT	RAPT	YIN	YIN
F0 shifting	no	no	no	no	yes	yes	yes	yes

4.5 Results of subjective evaluation

Figure 4.5 shows the result of the subjective evaluation for different FFT length on the left side. Here we only plot absolute values, which shows that the standard FFT length of 2048 is slightly better than the 4096 FFT length. For this figure we simply count how often a sample that was synthesized with one FFT length wins again a sample of the same utterance with the other FFT length.

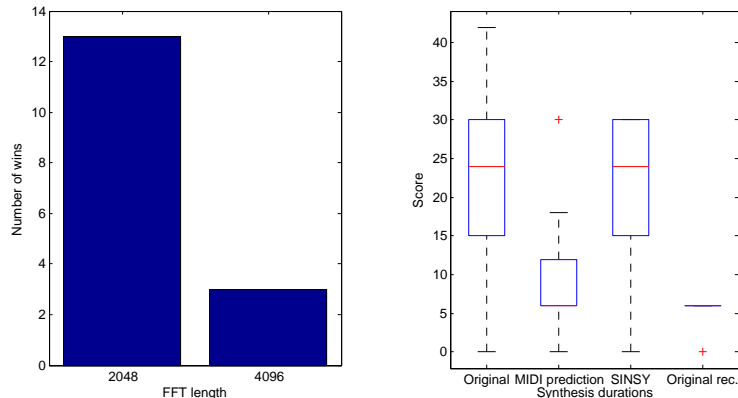


Figure 4.5: Results of subjective experiments for different FFT length (left), and different synthesis durations (right).

The right side of Figure 4.5 shows the result of the subjective evaluation for the different synthesis durations. Here we plot how often a certain synthesis method wins against the other methods for each train/synthesis combination. A Wilcoxon rank sum test shows us that all differences between synthesis durations are significant ($p < 0.001$). The original recordings have the best performance with winning all comparisons (rightmost bar). Using the original durations for synthesis is the second best method (leftmost bar). The third best method is the method that uses duration labels from the SINSY system, which are automatically predicted from MusicXML files and is thereby a full synthesis method.

The worst performance is shown by the MIDI predicted labels. This is contradicting the objective evaluation metric as shown in Figure 4.2 where the SINSY labels show the worst performance. This can be explained by the fact that the DTW based measure penalizes utterances with different duration stronger. Since the MIDI aligned labels have the same duration as the original ones, they get a higher similarity.

Figure 4.6 shows subjective evaluation results for the two different F0 extraction methods where we can see that the RAPT method has slightly more wins than the YIN method. For F0 expansion it is better to not use the expansion/shifting method, which contradicts results in the literature [40]. This can be due to the small amount of training data.

Figure 4.7 shows the result for the different training methods that are described in Table 4.3. Method 9 is the original recording. All methods are significantly different from the original recordings (Method 9) ($p < 0.001$) according to a Wilcoxon rank sum test. Methods 1-3 and 5-6 are also significantly different from Method 7 ($p < 0.05$). Method 7 uses the YIN F0 extraction method in combination with F0 expansion/shifting.

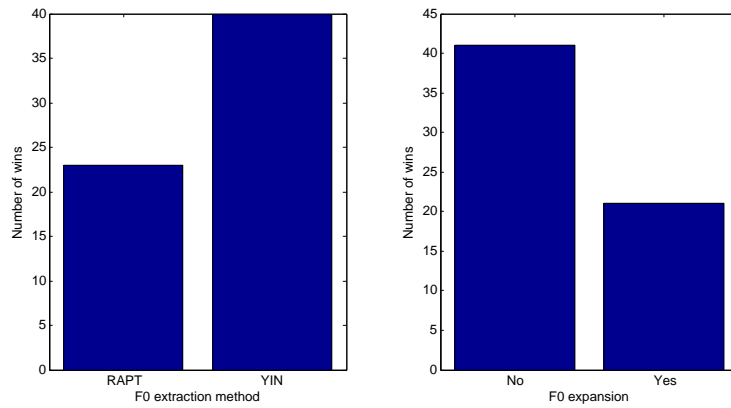


Figure 4.6: Results of subjective experiments for different F0 extraction method (left), and F0 expansion (right).

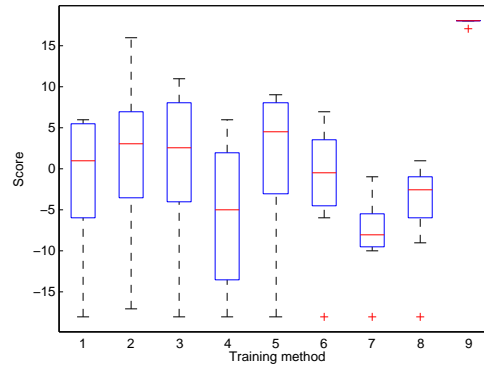


Figure 4.7: Results of subjective experiments for different training methods.

4.6 Analysis

Here we want to analyze the best and worst example according to our objective evaluation metric. The best utterance is synthesized with method 2 when using the original label files. The worst synthesized utterance uses method 11 with the SINSY predicted labels. The different parameter settings for the two methods are shown in Table 4.4.

Figure 4.8 shows the best synthesized utterance according to the MCD metric. We can see that the F0 lies approximately in the same range as the original one. However, we are missing the structure within the F0 such as the vibrato, which is not modeled by the synthesizer. But in the synthesized example we can also see some F0 dynamics that is modeling by the synthesizer, such that not only a flat F0 curve is generated.

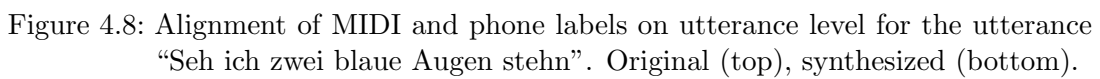
We can also see that a lot of spectral detail is lost in the synthesized example

Table 4.4: Two methods resulting in best and worst synthesis according to MCD.

Parameter	Method 2	Method 11
FFT length	4096	2048
F0 method	RAPT	YIN
F0 shifting	no	no
Training data alignment	semi-automatic	automatic

compared to the original one. This is a result of the vocoding as well as of our limited amount of training data.

Figure 4.9 shows the worst synthesized utterance according to the MCD metric. Here we can see that the duration of the synthesized sample has a large mismatch with the duration of the original one. For this synthesized sample we use the SINSY duration prediction. We can again see the missing spectral detail. In this example the synthesized F0 curve has a large deviation from the original one.



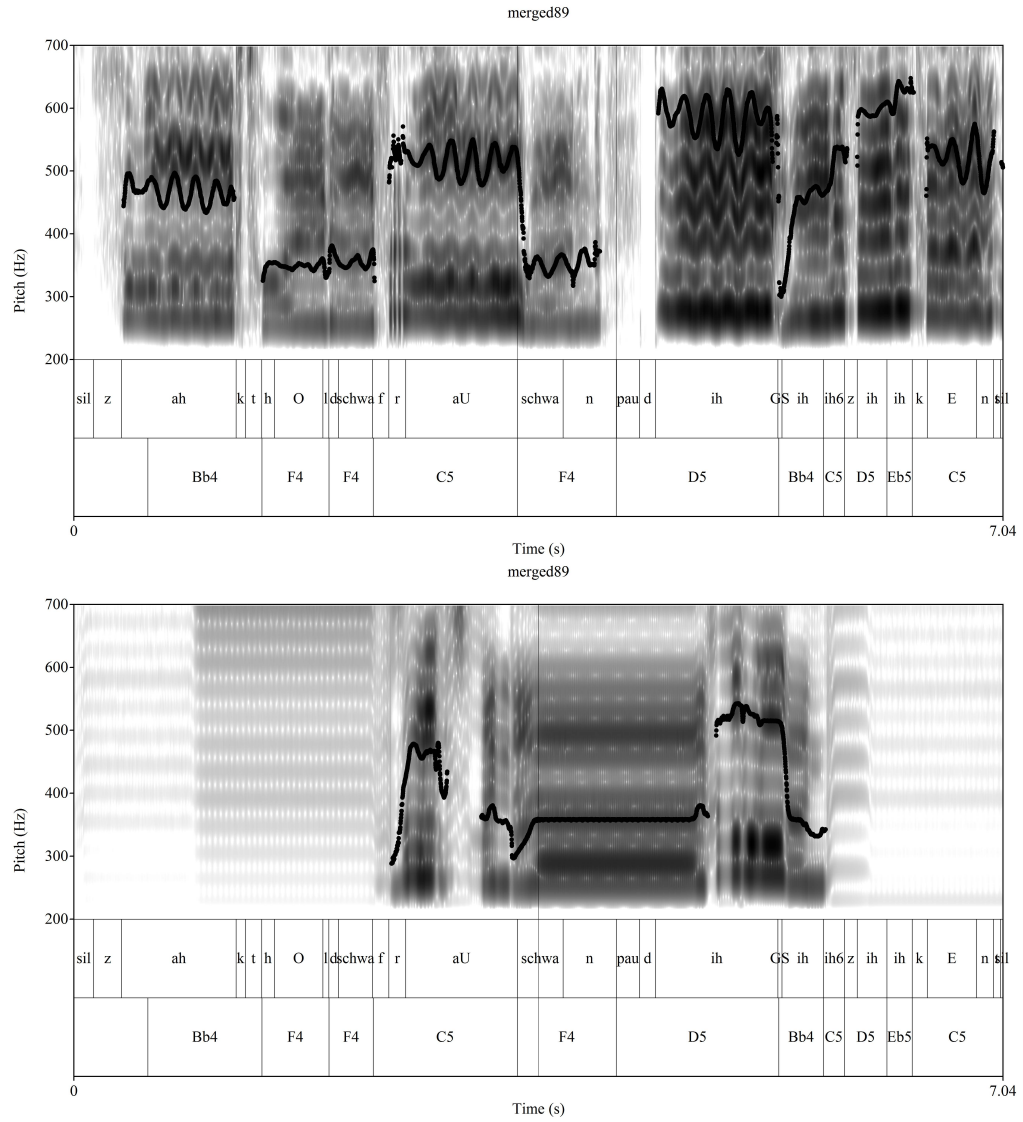


Figure 4.9: Alignment of MIDI and phone labels on utterance level for the utterance “Sagt, holde Frauen die ihr sie kennt”. Original (top), synthesized (bottom).

5 Conclusion

We showed how to develop a Hidden Markov Model (HMM) based opera singing synthesis system for German that is based on a Japanese singing synthesis system for popular songs. The subjective evaluation of the mezzo voice showed that moderate quality opera singing synthesis is feasible with the limited amount of training data at hand and that correct duration modeling is the most influential quality parameter at this stage.

The current system provides the basis for a future high quality system, and can also be used as a front-end for a general German singing synthesis system. Before our work such an HMM-based singing synthesis system was only available for Japanese and English. Furthermore our models can be used for general singing processing tasks like following the score of a singer on a phonetic level.

6 Future work

To improve the modeling of opera synthesis in our parametric framework more training data is needed. To extend the amount of training data we can use a phonetically balanced singing corpus, which contains all German phonemes or diphones. For processing of this data, which is not musically annotated, a method needs to be developed that allows for the automatic annotation of audio recordings with aligned phonetic, linguistic, and musical information.

A further possibility is to apply adaptive modeling and to train a background model with all speakers (mezzo, soprano, tenor, bass) or a subset of the speakers (mezzo, soprano) and to adapt our target model from this background model. For training the background model different clustering strategies can be used.

For a high quality opera singing synthesis systems we would also need to model the vibrato, which can be done in the current framework by introducing a new data stream that is separately clustered. Furthermore we need to improve the F0 extraction and modeling in general as well as the synthesis of fine spectral detail. To capture specific features of the so called singing formant a joint phonetic-musical score modeling could be employed. This joint modeling is at the moment done by the spectral clustering where musical and phonetic information is used.

Bibliography

- [1] Makemusic, “musicXML.” <http://www.musicxml.com/>, 2014.
- [2] FTW, “AMTV - Acoustic modeling and transformation of language varieties.” <https://portal.ftw.at/projects/amt>, 2013.
- [3] Sinsy, “HMM-based singing voice synthesis system.” <http://sinsy.sourceforge.net/>, 2013.
- [4] HTS, “HMM-based speech synthesis system (hts).” <http://hts.sp.nitech.ac.jp/>, 2013.
- [5] S. King, “Measuring a decade of progress in Text-to-Speech,” *Loquens*, vol. 1, no. 1, 2014.
- [6] L. Rabiner, “A tutorial on hidden markov models and selected applications in speech recognition,” *Proceedings of the IEEE*, vol. 77, pp. 257–286, Feb 1989.
- [7] L. Rabiner and B. Juang, *Fundamentals of speech recognition*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1993.
- [8] K. Tokuda, T. Yoshimura, T. Masuko, T. Kobayashi, and T. Kitamura, “Speech parameter generation algorithms for HMM-based speech synthesis,” in *Proc. ICASSP*, (Istanbul, Turkey), pp. 1315–1318, June 2000.
- [9] K. Tokuda, H. Zen, J. Yamagishi, A. Black, T. Masuko, S. Sako, T. Toda, T. Nose, and K. Oura, “The HMM-based speech synthesis system (HTS),” 2008.
- [10] B. Pfister and T. Kaufmann, *Sprachverarbeitung*. Springer, 2008.
- [11] D. Klatt, “Review of text-to-speech conversion for english,” *J. Acous. Soc. Amer.*, vol. 82, pp. 737–793, 1987.
- [12] H. Zen, K. Tokuda, and A. Black, “Statistical parametric speech synthesis,” *Speech Communication*, vol. 51, no. 11, pp. 1039–1064, 2009.
- [13] E. Moulines and F. Charpentier, “Pitch-synchronous waveform processing techniques for text-to-speech synthesis using diphones,” *Speech Communication*, vol. 9, pp. 453 – 467, 1990. Neuropeech ’89.

- [14] A. Hunt and A. Black, "Unit selection in a concatenative speech synthesis system using a large speech database," in *Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference on*, vol. 1, pp. 373–376 vol. 1, May 1996.
- [15] C. Papadimitriou, *Computational Complexity*. Addison Wesley, 1994.
- [16] T. Yoshimura, K. Tokuda, T. Masuko, T. Kobayashi, and T. Kitamura, "Speaker interpolation for HMM-based speech synthesis system," *Acoustical Science and Technology*, vol. 21, pp. 199–206, Jan. 2000.
- [17] M. Tachibana, J. Yamagishi, T. Masuko, and T. Kobayashi, "Speech synthesis with various emotional expressions and speaking styles by style interpolation and morphing," *IEICE Trans. Inf. Syst.*, vol. E88-D, pp. 2484–2491, Nov. 2005.
- [18] M. Pucher, D. Schabus, J. Yamagishi, F. Neubarth, and V. Strom, "Modeling and interpolation of Austrian German and Viennese dialect in HMM-based speech synthesis," *Speech Communication*, vol. 52, no. 2, pp. 164–179, 2010.
- [19] M. Pucher, D. Schabus, and J. Yamagishi, "Synthesis of fast speech with interpolation of adapted HSMMs and its evaluation by blind and sighted listeners," in *Proc. INTERSPEECH*, (Makuhari, Japan), 2010.
- [20] L. Mayfield, M. Gavalda, W. Ward, and A. Waibel, "Concept-based speech translation," in *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, vol. 1, pp. 97–100 vol.1, May 1995.
- [21] M. Mohri, "Finite-state transducers in language and speech processing," *Comput. Linguist.*, vol. 23, pp. 269–311, June 1997.
- [22] R. M. Kaplan and M. Kay, "Regular models of phonological rule systems," *Comput. Linguist.*, vol. 20, pp. 331–378, Sept. 1994.
- [23] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *Information Theory, IEEE Transactions on*, vol. 13, pp. 260–269, April 1967.
- [24] Y.-L. Chow and R. Schwartz, "The n-best algorithm: An efficient procedure for finding top n sentence hypotheses," in *Proceedings of the Workshop on Speech and Natural Language*, HLT '89, (Stroudsburg, PA, USA), pp. 199–202, Association for Computational Linguistics, 1989.
- [25] K. Tokuda, Y. Nankaku, T. Toda, H. Zen, J. Yamagishi, and K. Oura, "Speech synthesis based on hidden markov models," *Proceedings of the IEEE*, vol. 101, pp. 1234–1252, May 2013.

- [26] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification*. John Wiley & Sons, 2012.
- [27] S. Young, J. Odell, and P. Woodland, “Tree-based state tying for high accuracy acoustic modeling,” in *Proc. ARPA Human Language Technology Workshop*, pp. 307–312, Mar. 1994.
- [28] K. Shinoda and T. Watanabe, “Acoustic modeling based on the MDL criterion for speech recognition,” in *Proc. EUROSPEECH-97*, pp. 99–102, Sept. 1997.
- [29] K. Shinoda and T. Watanabe, “MDL-based context-dependent subword modeling for speech recognition,” *J. Acoust. Soc. Japan (E)*, vol. 21, pp. 79–86, Mar. 2000.
- [30] J. Dines, J. Yamagishi, and S. King, “Measuring the gap between hmm-based asr and tts,” in *INTERSPEECH*, pp. 1391–1394, ISCA, 2009.
- [31] K. Tokuda, T. Kobayashi, and S. Imai, “Speech parameter generation from HMM using dynamic features,” in *Proc. ICASSP*, pp. 660–663, May 1995.
- [32] Musescore, “Musescore.” <http://musescore.org/de>, 2014.
- [33] P. Birkholz, “Articulatory synthesis of singing,” in *INTERSPEECH 2007, 8th Annual Conference of the International Speech Communication Association, Antwerp, Belgium, August 27-31, 2007*, pp. 4001–4004, 2007.
- [34] T. Saitou, M. Goto, M. Unoki, and M. Akagi, “Vocal conversion from speaking voice to singing voice using straight,” in *INTERSPEECH*, pp. 4005–4006, ISCA, 2007.
- [35] T. Saitou, M. Goto, M. Unoki, and M. Akagi, “Speech-to-singing synthesis: Converting speaking voices to singing voices by controlling acoustic features unique to singing voices,” in *Applications of Signal Processing to Audio and Acoustics, 2007 IEEE Workshop on*, pp. 215–218, Oct 2007.
- [36] H. Kawahara, I. Masuda-Katsuse, and A. de Cheveigné, “Restructuring speech representations using a pitch-adaptive time-frequency smoothing and an instantaneous-frequency-based F0 extraction: Possible role of a repetitive structure in sounds,” *Speech Communication*, vol. 27, no. 3–4, pp. 187–207, 1999.
- [37] J. Sundberg, “Articulatory interpretation of the singing formant,” *J. Acoust. Soc. Am.*, vol. 55, pp. 838–844, 1974.
- [38] S. Ternström and J. Sundberg, “Formant-based synthesis of singing,” in *INTERSPEECH*, pp. 4013–4014, 2007.

- [39] H. Kenmochi and H. Ohshita, “Vocaloid - commercial singing synthesizer based on sample concatenation.,” in *INTERSPEECH*, pp. 4009–4010, ISCA, 2007.
- [40] K. Oura, A. Mase, T. Yamada, S. Muto, Y. Nankaku, and K. Tokuda, “Recent development of the HMM-based singing voice synthesis system - Sinsy,” in *SSW7*, (Kyoto, Japan), pp. 211–216, 2010.
- [41] K. Saino, H. Zen, Y. Nankaku, A. Lee, and K. Tokuda, “An HMM-based singing voice synthesis system,” in *Proc. Interspeech*, (Pittsburgh, PA, USA), pp. 2274–2277, 2006.
- [42] K. Oura, A. Mase, Y. Nankaku, and K. Tokuda, “Pitch adaptive training for HMM-based singing voice synthesis,” in *ICASSP*, (Kyoto, Japan), pp. 5377–5380, 2012.
- [43] J. Yamagishi and T. Kobayashi, “Average-voice-based speech synthesis using HSMM-based speaker adaptation and adaptive training,” *IEICE Trans. Information and Systems*, vol. E90-D, pp. 533–543, February 2007.
- [44] K. Nakamura, K. Oura, Y. Nankaku, and K. Tokuda, “HMM-based singing voice synthesis and its application to japanese and english,” in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pp. 265–269, May 2014.
- [45] O. Babacan, T. Drugman, T. Raitio, D. Erro, and T. Dutoit, “Parametric representation for singing voice synthesis: A comparative evaluation,” in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pp. 2564–2568, May 2014.
- [46] K. Tokuda, T. Kobayashi, T. Masuko, and S. Imai, “Mel-generalized cepstral analysis - a unified approach to speech spectral estimation,” in *The 3rd International Conference on Spoken Language Processing, ICSLP 1994, Yokohama, Japan, September 18-22, 1994*, 1994.
- [47] T. Drugman and T. Dutoit, “The deterministic plus stochastic model of the residual signal and its applications,” *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 20, no. 3, pp. 968–981, 2012.
- [48] D. Erro, I. Sainz, E. Navas, and I. Hernaez, “Harmonics plus noise model based vocoder for statistical parametric speech synthesis,” *Selected Topics in Signal Processing, IEEE Journal of*, vol. 8, no. 2, pp. 184–194, 2014.
- [49] T. Raitio, A. Suni, H. Pulakka, M. Vainio, and P. Alku, “Utilizing glottal source pulse library for generating improved excitation signal for hmm-based speech

- synthesis,” in *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pp. 4564–4567, IEEE, 2011.
- [50] T. Tomoki and K. Tokuda, “A speech parameter generation algorithm considering global variance for hmm-based speech synthesis,” *IEICE TRANSACTIONS on Information and Systems*, vol. 90, no. 5, pp. 816–824, 2007.
- [51] H. Zen, K. Oura, T. Nose, J. Yamagishi, S. Sako, T. Toda, T. Masuko, A. W. Black, and K. Tokuda, “Recent development of the hmm-based speech synthesis system (hts),” 2009.
- [52] MIDI, “Musical instrument digital interface (MIDI).” <http://en.wikipedia.org/wiki/MIDI>, 2013.
- [53] MATLAB, “Matlab - The language of technical computing.” www.mathworks.com/matlab/, 2014.
- [54] T. Eerola and P. Toiviainen, *MIDI Toolbox: MATLAB Tools for Music Research*. Jyväskylä, Finland: University of Jyväskylä, 2004.
- [55] R. J. Turetsky and D. P. W. Ellis, “Ground-truth transcriptions of real music from force-aligned midi syntheses,” in *ISMIR*, 2003.
- [56] D. Ellis, “Aligning MIDI scores to music audio.” <http://www.ee.columbia.edu/~dpwe/resources/matlab/alignmidiwav/>, 2008.
- [57] OPERA, “Acoustic modeling and statistical analysis of Vienna opera singers (opera) - NII internal project.” https://portal.ftw.at/projects/amtv/opern/pa/at_download/file, 2014.
- [58] M. Pucher, F. Neubarth, V. Strom, S. Moosmuller, G. Hofer, C. Kranzler, G. Schuchmann, and D. Schabus, “Resources for speech synthesis of viennese varieties,” in *LREC*, pp. 105–108, 2010.
- [59] FTW, “Austrian German voices for Festival.” <http://sourceforge.net/projects/at-festival/>, 2013.
- [60] J. C. Wells *et al.*, “Sampa computer readable phonetic alphabet,” *Handbook of standards and resources for spoken language systems*, vol. 4, 1997.
- [61] HTS, “Speaker dependent training demo - Japanese song.” http://hts.sp.nitech.ac.jp/archives/2.3beta/HTS-demo_NIT-SONG070-F001.tar.bz2, 2014.

- [62] A. De Cheveigné and H. Kawahara, “Yin, a fundamental frequency estimator for speech and music,” *The Journal of the Acoustical Society of America*, vol. 111, no. 4, pp. 1917–1930, 2002.
- [63] A. De Cheveigné, “Yin.” <http://audition.ens.fr/adc/sw/yin.zip>, 2013.
- [64] D. Talkin, “A robust algorithm for pitch tracking (rapt),” *Speech coding and synthesis*, vol. 495, p. 518, 1995.
- [65] SPTK, “Speech signal processing toolkit (sptk).” <http://sp-tk.sourceforge.net/>, 2014.
- [66] SOX, “SoX - Sound exchange.” <http://sox.sourceforge.net/>, 2015.
- [67] Wikipedia, “WAV.” <http://en.wikipedia.org/wiki/WAV>, 2014.
- [68] Audacity, “Audacity ist freie, plattformunabhängige open-source-software für die aufnahme und bearbeitung von audio..” <http://audacity.sourceforge.net/>, 2014.
- [69] Python, “Python.” <https://www.python.org/>, 2014.
- [70] Praat, “Praat: doing phonetics by computer.” <http://www.fon.hum.uva.nl/praat/>, 2014.
- [71] J. Kominek, T. Schultz, and A. W. Black, “Synthesizer voice quality of new languages calibrated with mean MEL cepstral distortion.,” in *SLTU*, pp. 63–68, 2008.
- [72] T. H. Falk and S. Möller, “Towards signal-based instrumental quality diagnosis for text-to-speech systems,” *Signal Processing Letters, IEEE*, vol. 15, pp. 781–784, 2008.

A Context dependent label format

This is the definition of the context-dependent label format used for training. Each phone is described in this format. Table A.1 describes the different features, other symbols are part of the syntax of the label format. The format is taken from the label format for HMM-based singing voice synthesis released by the HTS Working Group on December 25, 2013 [3].

```
p1@p2^p3-p4+p5=p6_ p7%p8^p9_p10~p11-p12!p13[p14$p15]p16
/A:a1-a2-a3@a4~a5/B:b1_b2_b3@b4|b5/C:c1+c2+c3@c4&c5
/D:d1!d2#d3$d4%d5|d6&d7;d8-d9
/E:e1]e2^e3=e4~e5!e6@e7#e8+e9]e10$e11|e12[e13&e14]e15=e16^e17~e18#e19_e20
;e21$e22&e23%e24[e25|e26]e27-e28^e29+e30~e31=e32@e33$e34!e35%e36#e37|
e38|e39-e40&e41&e42+e43[e44;e45]e46;e47~e48~e49^e50^e51@e52[e53#e54=e55!e56
~e57+e58!e59^e60
/F:f1#f2#f3- f4$f5$f6+f7%f8;f9
/G:g1_g2/H:h1_h2/I:i1_i2
/J:j1~j2@j3
```

p1	the language independent phoneme identity
p2	the phoneme identity before the previous phoneme
p3	the previous phoneme identity
p4	the current phoneme identity
p5	the next phoneme identity
p6	the phoneme identity after the next phoneme
p7	the phoneme flag before the previous phoneme
p8	the previous phoneme flag
p9	the current phoneme flag
p10	the next phoneme flag
p11	the phoneme flag after the next phoneme
p12	position of the current phoneme identity in the syllable (forward)
p13	position of the current phoneme identity in the syllable (backward)
p14	undefined context
p15	undefined context
p16	undefined context
a1	the number of phonemes in the previous syllable
a2	position of the previous syllable identity in the note (forward)
a3	position of the previous syllable identity in the note (backward)
a4	the language of the previous syllable
a5	the language dependent context of the previous syllable
b1	the number of phonemes in the current syllable
b2	position of the current syllable identity in the note (forward)

b3	position of the current syllable identity in the note (backward)
b4	the language of the current syllable
b5	the language dependent context of the current syllable
c1	the number of phonemes in the next syllable
c2	position of the next syllable identity in the note (forward)
c3	position of the next syllable identity in the note (backward)
c4	the language of the next syllable
c5	the language dependent context of the next syllable
d1	the absolute pitch of the previous note (C0-G9)
d2	the relative pitch of the previous note (0-11)
d3	the key of the previous note (the number of sharp)
d4	the beat of the previous note
d5	the tempo of the previous note
d6	the length of the previous note by the syllable
d7	the length of the previous note by 0.01 second
d8	the length of the previous note by one-third of the 32nd note
d9	undefined context
e1	the absolute pitch of the current note (C0-G9)
e2	the relative pitch of the current note (0-11)
e3	the key of the current note (the number of sharp)
e4	the beat of the current note
e5	the tempo of the current note
e6	the length of the current note by the syllable
e7	the length of the current note by 0.01 second
e8	the length of the current note by one-third of the 32nd note
e9	undefined context
e10	position of the current note identity in the current measure by the note (forward)
e11	position of the current note identity in the current measure by the note (backward)
e12	position of the current note identity in the current measure by 0.01 second (forward)
e13	position of the current note identity in the current measure by 0.01 second (backward)
e14	position of the current note identity in the current measure by one-third of the 32nd note (forward)
e15	position of the current note identity in the current measure by one-third of the 32nd note (backward)
e16	position of the current note identity in the current measure by % (forward)
e17	position of the current note identity in the current measure by % (backward)
e18	position of the current note identity in the current phrase by the note (forward)
e19	position of the current note identity in the current phrase by the note (backward)
e20	position of the current note identity in the current phrase by 0.01 second (forward)
e21	position of the current note identity in the current phrase by 0.01 second (backward)
e22	position of the current note identity in the current phrase by one-third of the 32nd note (forward)
e23	position of the current note identity in the current phrase by one-third of the 32nd note (backward)
e24	position of the current note identity in the current phrase by % (forward)
e25	position of the current note identity in the current phrase by % (backward)
e26	whether slur or not in between the current note and the previous note
e27	whether slur or not in between the current note and the next note
e28	dynamic mark of the current note
e29	the distance between the current note and the next accent by the note

e30	the distance between the current note and the previous accent by the note
e31	the distance between the current note and the next accent by 0.01 second
e32	the distance between the current note and the previous accent by 0.01 second
e33	the distance between the current note and the next accent by one-third of the 32nd note
e34	the distance between the current note and the previous accent by one-third of the 32nd note
e35	the distance between the current note and the next staccato by the note
e36	the distance between the current note and the previous staccato by the note
e37	the distance between the current note and the next staccato by 0.01 second
e38	the distance between the current note and the previous staccato by 0.01 second
e39	the distance between the current note and the next staccato by one-third of the 32nd note
e40	the distance between the current note and the previous staccato by one-third of the 32nd note
e41	position of the current note in the current crescendo by the note (forward)
e42	position of the current note in the current crescendo by the note (backward)
e43	position of the current note in the current crescendo by 1.0 second (forward)
e44	position of the current note in the current crescendo by 1.0 second (backward)
e45	position of the current note in the current crescendo by one-third of the 32nd note (forward)
e46	position of the current note in the current crescendo by one-third of the 32nd note (backward)
e47	position of the current note in the current crescendo by % (forward)
e48	position of the current note in the current crescendo by % (backward)
e49	position of the current note in the current decrescendo by the note (forward)
e50	position of the current note in the current decrescendo by the note (backward)
e51	position of the current note in the current decrescendo by 1.0 second (forward)
e52	position of the current note in the current decrescendo by 1.0 second (backward)
e53	position of the current note in the current decrescendo by one-third of the 32nd note (forward)
e54	position of the current note in the current decrescendo by one-third of the 32nd note (backward)
e55	position of the current note in the current decrescendo by % (forward)
e56	position of the current note in the current decrescendo by % (backward)
e57	pitch difference between the current and previous notes
e58	pitch difference between the current and next notes
f1	the absolute pitch of the next note (C0-G9)
f2	the relative pitch of the next note (0-11)
f3	the key of the next note (the number of sharp)
f4	the beat of the next note
f5	the tempo of the next note
f6	the length of the next note by the syllable
f7	the length of the next note by 0.01 second
f8	the length of the next note by one-third of the 32nd note
f9	undefined context
g1	the number of syllables in the previous phrase
g2	the number of phonemes in the previous phrase
h1	the number of syllables in the current phrase
h2	the number of phonemes in the current phrase
i1	the number of syllables in the next phrase
i2	the number of phonemes in the next phrase
j1	the number of syllables in this song / the number of measures in this song
j2	the number of phonemes in this song / the number of measures in this song

A Context dependent label format

j3		the number of phrases in this song
----	--	------------------------------------

Table A.1: Description of features.