Building a visual speech recognizer

Master's thesis

Karin F. Driel

August 18, 2009

TU Delft

Delft University of Technology

Building a visual speech recognizer

Thesis,
submitted in partial fulfillment of the requirements for the degree of
Master of Science
in
Media & Knowledge Engineering

Karin F. Driel
Born on December 8, 1984, in Delft, The Netherlands

Delft University of Technology

Man-Machine Interaction group
Faculty of Electrical Engineering, Mathematics and Computer Science
Mekelweg 4
2628 CD Delft, The Netherlands
http://ewi.faculteiten.tudelft.nl

Summary

This thesis describes how an automatic lip reader was realized. Visual speech recognition is a precondition for more robust speech recognition in general. The development of the software comprised the following steps: gathering of training data, extracting meaningful features from the obtained video material, training the speech recognizer and finally evaluating the resulting product.

First, research was done to gain insight on the theoretical aspects of automatic lip reading, the state of the art, speech corpus development, face tracking and feature extraction.

Gathering training data came down to the recording and composing of a new audio-visual speech corpus for Dutch. With frontal and side images of 70 different speakers recorded at a frame rate of 100 frames per second this is the most diverse corpus currently in existence. Analysis of the new data corpus shows an increase in quality compared to other corpora.

Visual information is obtained by searching the video footage. Using Active Appearance Models, points of an a priori defined model of the lower half of the face are tracked over time. Based on the model point coordinates, distance and area, features are computed that are used as input to the speech recognizer.

Training was accomplished by presenting labeled training data to viseme-based Hidden Markov Models that model speech production. In a few steps the model parameters were adjusted, so that it could be used to perform recognition of visual speech signals from then on. The recognizer was implemented using tools from the Hidden Markov Model Toolkit.

The results of a visual speech recognizer based on training data from a single person depend on the utterance type of the unlabeled data. For the simple word-level task of digit recognition 78% was recognized correctly with a word recognition rate of 68%. For letter recognition tasks it did not perform nearly as well, but considering the limitations that the use of visemes over phonemes imposes, these results are at the expected level. The data corpus and visual speech recognizer will be a valuable asset to future research.

Karin F. Driel
Student number: 1149784
E-mail address: karindriel@gmail.com

Thesis committee

Prof. Dr. Drs. Leon J.M. Rothkrantz
Dr. Ir. Pascal Wiggers
Ir. Hans J.A.M. Geers
Ir. Alin G. Chiţu

Samenvatting

Dit afstudeerverslag beschrijft hoe een automatische liplezer tot stand kwam. Visuele spraakherkenning is een voorwaarde voor betrouwbaardere spraakherkenning in de nabije toekomst. De ontwikkeling van deze software omvatte de volgende stappen: het verzamelen van trainingsdata, het extraheren van zinvolle visuele informatie uit de verkregen video-opnamen, het trainen van de spraakherkenner en tenslotte de evaluatie van het eindproduct.

Eerst is er onderzoek gedaan ter voorbereiding op het liplezen. Het huidige niveau van automatische liplezers, spraakdatabases en methodes om het gezicht te volgen in een video en hier kenmerken aan te onttrekken, kwamen aan bod.

Het verzamelen van trainingdata kwam neer op de opname en samenstelling van een audio-visuele spraakdatabase voor het Nederlands. Met opnamen van de voorkant en het profiel van 70 proefpersonen op een snelheid van 100 beelden per seconde biedt deze verreweg de meeste mogelijkheden van het moment. Uitgebreide analyse van de nieuwe database wijst op een sterke kwaliteitsverbetering ten opzichte van bestaande audio-visuele spraakcorpora.

Visuele informatie wordt verkregen door het doorzoeken van videobeelden. Door middel van Active Appearance modellen worden de punten van een vooraf gedefiniëerd model van de onderste helft van het gezicht gevolgd door de videobeelden heen. Aan de hand van de puntcoördinaten worden afstanden en oppervlaktes berekend die als invoer voor de spraakherkenner dienen.

Training vond plaats door de presentatie van gelabelde trainingdata aan op visemen gebaseerde Hidden Markov modellen die de spraakproductie modelleren. In een aantal stappen worden de modelparameters aangepast zodanig dat nieuwe videodata kan worden “gelezen” met behulp van het Viterbi algoritme.

De resultaten van de spraakherkenner gebaseerd op trainingsdata van één persoon lopen uiteen naar gelang de aard van de te herkennen uitspraak. De telwoorden 0-9 geven een herkenning van 78% en een nauwkeurigheid van 68%. Bij taken zonder grammatica van vaste lengte blijft het resultaat achter, maar uitgaande van de beperkingen die alleen al het gebruik van visemen ten opzichte van fonemen met zich meebrengt, was dit te verwachten. De data corpus en visuele spraakherkenner zullen voor verder onderzoek van grote waarde zijn.

Karin F. Driel
Student nummer: 1149784
E-mail adres: karindriel@gmail.com

Afstudeercommissie

Prof. Dr. Drs. Leon J.M. Rothkrantz
Dr. ir. Pascal Wiggers
Ir. Hans J.A.M. Geers
Ir. Alin G. Chiţu

Preface

At the Department of Man Machine Interaction at Delft University of Technology, there is an ongoing project on multimodal speech recognition. For this Master's project one aspect of multimodal speech recognition has been tackled, namely visual speech recognition.

People are subconsciously lip reading all the time. This is apparent from practical situations – it's easier to understand someone in a noisy bar if you're looking at their face, and scientific evidence (like the well-known McGurk effect). There are some things we hoped to learn from this project. First of all, it would be interesting to see whether a computer could be taught to read lips as well as human professionals. The ultimate goal would be a combination of visual and acoustic speech recognition. Though beyond the scope of this thesis, it has the best potential to bring the performance of speech recognition to a level where it could be more integrated in our daily lives.

Not to be underestimated is the amount of work involved with the processing of video to perform visual speech recognition. A lot of it involves manual work. First there was the recording of the data itself; then there was the transcription of the data. Then annotating of training images had to be performed by hand to train the tracking model, and applying the trained model also required supervision. Lastly we had to select the data for training the recognizer by hand, and perform some more manual labor while fine-tuning the recognizer. Luckily our efforts have been paid off with the vast audio-visual speech corpus that is now at our disposal, and the other results that will be discussed throughout this thesis.

Visual speech recognition has some practical applications that could help the speaking-disabled, like my uncle who is paralyzed from the neck down. I hope my work has contributed to the quick realization of a real-time implementation people like him could benefit from.

I would like to thank Leon Rothkrantz and Alin Chiţu for their guidance, the people in the student lab for their friendship, my partner Hans van Gurp and my father Marijn Driel for their moral support and efforts to proof-read. And last but not least, I would like to thank all the people who agreed to become part of the new data corpus, especially Ank Voets who has by now sat through five recording sessions, some of which lasted a few hours.

Table of contents

1 Introduction

Over the years, we have seen computers become much faster and smarter. Computers of only a couple of years old are nothing compared to the latest ones in terms of computing power and graphic capabilities. This is why it is almost unbelievable that we keep working with the same keyboard and mouse interface that causes us to develop repetitive strain injury and headaches. Interfacing with a computer would be much more natural and healthy if we could just talk to it.

So, what is stopping us from incorporating automatic speech recognition into our daily lives? The answer to that is the limited performance: especially when the environment is the smallest bit noisy, automated speech recognition does not achieve a perfect recognition yet. One step outside a quiet laboratory or office will make the accuracy rapidly degrade. At least one thing to look forward to is that solving the noise problem will take us one step closer to that idealistic "Star Trek" computer interface we all dream of.

However, the options are limited. If the problem lies in the quality of the sound despite attempts to filter out noise, one might want to start looking at other media, like vision, to make recognition more robust. When people are trying to understand someone in a noisy environment, they subconsciously start looking at the face and interpret the speech information lying within. The face seems to provide a rich source of information about speech. Experienced speech readers do not even need sound at all to understand speech. Teaching a computer to read lips may provide that extra information channel needed to achieve robust recognition.

This thesis describes how such an automatic lip reader was developed. The reader should keep in mind that training a speech recognizer, both acoustic and visual, is tedious work and a "quick fix" for any problems encountered is not realistic. With every change of plans data has to be rearranged and training has to be done over, taking hours. To train a visual speech recognizer large quantities of speaker video data are required. Mainly due to the storage limitations of the past, these were not available before. Therefore, recording training data has been an integral part of this graduation project.

1.1 Applications of automatic lip reading

The societal relevance of automatic lip reading may not be as obvious as that of speech recognition in general, but there are some cases where pure automatic lip reading is the best candidate to solve the problem. Generally speaking, a visual speech recognizer could tackle every speech recognition task where there is video but no or non-retrievable sound. An example of this is mute or deteriorated film. Especially extensive tasks like subtitling film archives would benefit from automatic speech processing as opposed to hiring a human lip reader. Figure 1.2 shows a similar application.

An automatic lip reading application would open up a world of opportunity to help the disabled. Just like the hearing impaired benefit from lip reading as a skill, automatic lip reading could help in training applications. For people who just recently lost their hearing and have not learned lip reading or sign language (see Figure 1.1) yet, a mobile speech recognizer could help in daily life. While the most important part would be the recognition from sound, a lip reading attachment would make it robust, as daily life is a very noisy environment. Another group of disabled that could be aided by visual speech recognition are the speaking impaired; those who can move their lips but lost the ability to produce an adequate sound level could use an automatic lip reader to make themselves understandable by the people around them.

Figure 1.1: Fragment of the Dutch NOS "Jeugdjournaal" news bulletin of July 31, 2009, supported by a sign language interpreter. (http://nos.nl)

Another opportunity lies in long-distance communication. Private phone conversations in public places could remain private if speakers could whisper over the phone. If the phone used is not a video phone, humans already have trouble understanding whispering over the phone. For a speech recognizer (think of automatic dictation applications) this would be practically impossible. An automatic lip reader could provide a solution in both cases. The only speech recognizer that will understand whispering is an automatic lip reader.

Another application that pops to mind is video surveillance. Video recorded by surveillance cameras in public places, shops or trains, isn't usually accompanied by sound. Now if the video is recorded at a sufficiently high resolution, an automatic lip reader could still make out what was said (e.g. verbal threats), providing evidence of a possible irregularity. If recognition is done live an alarm could be triggered and security guards could arrive at the scene in time. A combination of automatic surveillance done by motion detection (to detect physical aggression) and lip reading has the most potential of replacing human watchmen eventually.

Figure 1.2: Zinedine Zidane headbutting Marco Materazzi after having been insulted during the 2006 FIFA World Cup football final. After video evidence suggested that Materazzi had verbally provoked Zidane, three British media newspapers claimed to have hired lip readers to determine what Materazzi had said. [1]

One of the major areas where a visual speech recognizer would be of value is research. It is expected that multimodal speech recognition is the key to robust recognition. Apart from the problems that arise when trying to combine the modalities, there will always be a need to objectively compare the methods used for video processing alone. The performance of a pure lip reader based on the

techniques that are desired to be used in a bimodal recognizer, allows the evaluation of just that method, so other aspects of multimodal processing can be evaluated separately. Furthermore, from a scientific point of view it's interesting to see whether it is possible to build an automatic lip reader that performs as well as a human lip reader.

1.2 Problem definition

The purpose of this graduation project was to develop a visual speech recognizer for the Dutch language that would preferably be able to run in real time. By making certain improvements like using a more extensive data corpus, we aimed for results that would exceed those achieved by our predecessors.

A large portion of the work would involve the recording of a new data corpus extensive enough to train a recognizer (visual or bimodal) thoroughly. The most common reason for an automatic lip reader to perform suboptimal is insufficient quality training data being available.

This project was part of the ongoing project of multimodal human computer interaction conducted at the department of Man-Machine Interaction at Delft University of Technology. Our automatic lip reader will give insight on the potential of the visual modality for such a recognizer.

The goals we set for this project are the following:

1. *Exploring the potential of a lip reading system based on Hidden Markov Models*
2. *Exploring the possibilities for implementing a lip reader that can be run real-time*
3. *Evaluating feature extraction methods discussed in literature according to the criteria of performance, speed and speaker independence*
4. *Obtaining a visual speech corpus that is sufficient in size and quality to train and test a pure lip reader from scratch*
5. *Preparing and implementing the separate parts that make up the automatic lip reader (data formatting, feature extraction, language model)*
6. *Evaluating the results obtained from experiments using the trained lip reader, according to expectations*

The approach we took to reach the goals we set for this project is outlined in Table 1.1, which shows out methodology. It also includes the numbers of the chapters containing the results of the specific actions taken.

Table 1.1: Methodology

Action	Goal	Chapter
Researching related topics	Getting a grasp on the theory, avoiding common mistakes, avoiding reinventing the wheel	2
Researching Hidden Markov Models	Being able to define a good model, preparing for implementation	3, 4
Comparing feature extraction methods	Being able to choose a good method with respect to performance, speed and speaker independence	2
Recording a new speech corpus	Gathering sufficient training data to train a lip reader	5
Processing recordings	Ensuring the quality of the training data	5
Annotating video frames	Training the lip tracking model, data parameterization of raw video	6
Implementing feature extraction from points	Providing the speech recognizer with some meaningful speech features	7
Evaluating feature performance	Being able to determine which part of the recognizer leaves room for improvement	7
Implementing an automatic lip reader	Being able to determine the performance of an automatic lip reader following our design	8
Reporting everything into detail	Enabling others to continue the work	9, 10

1.3 Research challenges

The scientific community faces many challenges when trying to integrate different modalities into speech recognition. Questions waiting to be answered are, for example:

1. *Is it possible to build an automatic lip reader comparable to or even better than a human lip reader?*
2. *Is it possible to build an automatic lip reader that performs as well as an acoustic speech recognizer?*
3. *In which way should we integrate the results of automatic lip reading and acoustic speech recognition?*
4. *Can we make an automatic lip reader that performs real-time?*
5. *What are the quantitative and qualitative requirements of the data we use to train an automatic lip reader?*
6. *Can the methodology to train an acoustic speech recognizer be directly applied to train an automatic lip reader?*
7. *Which feature extraction method should be chosen as the standard for automatic lip reading in general?*

By the end of this thesis, we will hopefully have (partially) answered some of these questions and have brought science that much further.

1.4 Thesis outline

The structure of this thesis follows the flow of this project. The first part revolves around the development, the second around the realization, and finally the evaluation of obtained results.

Development begins with the study of existing systems and theory. These, and the resulting design choices, are addressed in chapter 2 (Related Work). Chapter 3 (Basics of speech recognition) explains the statistical approach of speech recognition, as this is also the approach that we will adopt; we discuss the theory, algorithms and tools. In chapter 4 (Visual speech recognition) the aspects more specific to visual speech recognition are looked into: we choose the speech units and make predictions about the performance. The result of the development phase is a set of tools, an approach and a model for visual speech recognition, that shows how we envision training and recognition.

In the realization phase, the different aspects of the visual speech recognizer are implemented and finally linked together. In chapter 5 (Data acquisition) we describe how a new speech corpus was composed. The chapters thereafter explain how we poured this data into a format a speech recognizer can handle. Data parameterization starts with chapter 6 (Lip tracking), where we used Active Appearance Models to track points on the face, while in chapter 7 (Feature extraction) we explain how these points were used to extract features for training and testing the recognizer. In chapter 8 (Implementation) we describe how all the pieces of the puzzle were put together, and a functioning recognizer was realized. The realization phase resulted in an implementation of a visual speech recognizer, which we will evaluate in the final phase.

Evaluation starts by analysis of recognizer performance, the results produced of which are described in chapter 9 (Experiments and results). In chapter 10 (Conclusions and recommendations) we present the final findings for this research and make recommendations for future work.

2 Related work

Automatic speech recognition has many aspects ranging from signal processing to probabilistic models. Some more details about speech recognition will be given in chapter 3. The aspects that are unique to visual speech recognition will be highlighted in chapter 4.

All the way though this project, the available literature has helped us decide on the best courses of action. This chapter starts out with some definitions of recurring terms (2.1). Next, section 2.2 provides an overview of the current state of the art in visual speech recognition. Some related topics are highlighted in section 2.3.

Since the bottleneck of most other automatic lip reading projects appears to be a lack of data, we investigated some existing data corpora to decide which one to use (2.4). In section 2.5 we explore different feature extraction methods that could be used on this data: while for audio there are pretty straightforward ways to derive meaningful features, for video this is not such a trivial task just yet.

2.1 Definitions

In this section, a few easily confusable terms that occur a number of times throughout this thesis are defined.

Visual speech recognition

First of all, the terms automatic lip reader and visual speech recognizer are used interchangeably, with automatic lip reading being the way we hope to accomplish visual speech recognition. "Visual speech recognizer" was chosen for the title of this thesis because it is more general and for "automatic lip reader" there exist different spellings.

Visemes

The basic units of acoustic speech that can be distinguished are often referred to as phonemes. Likewise, a set of visually indiscriminable phonemes can be referred to as a visual phoneme or "viseme". In human lip reading, visemes are considered the basic information one should have at their disposal to be able to read lips. The term also recurs in speech therapy and speech animation.

Measures for recognizer performance

Once the output of a speech recognizer is ready, it can be compared to known labels to determine the "percent correct" and "percent accuracy" performance measures. [2] An optimal string match is found using dynamic programming, where a score is calculated for the match with respect to the reference and penalty values for each occurring error are added to the final score. The optimal string match is the label alignment which has the lowest possible score.

Once the optimal alignment has been found, the "percent accuracy" and the "percentage correct" (that ignores insertion errors) can be found according to the number of present deletion (D), substitution (S) and insertion (I) errors, with a total of N sentences or words.

$$\text{Percent Correct} = \frac{N - D - S}{N} \times 100\% \quad (2.1)$$

$$\text{Percent Accuracy} = \frac{N - D - S - I}{N} \times 100\% \quad (2.2)$$

The accuracy is also referred to as word recognition rate (WRR). Also word error rate (WER) is used, which is equal to 1 - WRR. It is generally agreed that performance accuracy at a rate below 95% is not acceptable for applications.

The speed measure often used to evaluate a speech recognizer is the "real time factor" (RTF). As long as the processing time is more than the duration of the speech input, the RTF is greater than 1 and the real-time requirement has not been met.

$$RTF = \frac{\text{processing time}}{\text{input duration}} \quad (2.3)$$

2.2 State of the art in automatic lip reading

Laying aside the problem of lip reading for a while, speech recognizers can be classified according to their vocabulary size or input speech type: word-level (single words), sentence-level (according to a grammar), or continuous (ongoing signal). Word-level speech recognition is the easiest task, while continuous is the hardest. In continuous speech recognition the task of identifying meaningful chunks is left up to the recognizer. This imposes a need to detect the onset and offset of speech. Sentence level recognition is the minimum requirement for simple dialogue systems.

The state of the art in automatic speech recognition is at the level of continuous speech recognition for applications like dictation systems (even though they require speakers to work in a noise-clean environment, have a profile that matches the training data and have opportunity to perform speaker adaptation). For automatic lip reading, we noticed that current systems perform at no more than word level. The reason for that is probably that most of them exist only as a proof of concept for a certain feature extraction technique or to evaluate the quality of an audio-visual speech corpus. For this reason those existing lip reading systems will be discussed in the relevant sections. The next section is about some PhD work done on automatic lip reading.

Previous work

Jacek Wojdeł, who was a PhD student at Delft University of Technology, researched visual speech recognition for his promotion some years ago. His work can be seen as preparation for our project. The approach he took was also similar. In this section we will discuss his findings. [3]

His work began with comparing different feature extraction methods and developing his own, Lip Geometry Estimation (LGE), which will be discussed in section 2.5.3. He then explored the possible approaches to building a recognizer. A type of artificial neural network that is able to handle the temporal dimension is considered, but the final choice is the Hidden Markov Model (HMM), which is a statistical model that can model speech quite naturally.

Strings of digits (limited vocabulary) were used in a lip reading experiment. Binding the number of digits per utterance with a grammar improved the accuracy results significantly, probably due to an undertrained silence model. Using intensity features along with LGE boosted the results even more. However, training on data from five different people as opposed to a single person brought the recognition rate down significantly. This suggests that the features are not speaker-independent after all.

Some of his work involved lip reading options for continuous speech. To perform onset/offset detection an Artificial Neural Network (ANN) was used. Results were verified using the presence of audio. A problem was the difference in timing between the video and audio channels. Vowel/consonant discrimination could also be done using an ANN.

To explain these results, the disadvantages that are implied by the use of visemes, were given as a possible reason. A second reason for the lack of performance might have been that the viseme models for "silence" were undertrained. This could be solved by using a larger speech corpus. Also, a larger HMM could be used, but the more free parameters an HMM contains, the larger the data set needs to be in order to train them properly. Because of the temporal dimension, other, trajectory-based models might be better suited than HMM to model visual speech.

Eventually, the project was dropped due to unsatisfactory performance and continued as a multi-modal project, for fusion of audio and visual media. The lip reader they wanted to use for late integration feature fusion reached a performance of only 10%. One reason for this might be that coarticulation between visemes is more evident than for phonemes. To model this coarticulation, a triphone model would be needed where a monophone model would be sufficient for phonemes (see section 8.2 for details). There wasn't enough data to train a larger model, however. The other reason is the lack of context. Context information is essential for continuous speech recognition, especially for a lip reader that is theoretically always outperformed by an acoustic speech recognizer (see section 4.2.2).

The final conclusions of this work are the following:

- The expected performance of lip reading lies far below that of acoustic speech recognition
- Although a small data corpus was recorded, a bigger one would be needed for further research
- The feature extraction method that will probably be most successful is geometry based (e.g. LGE), extended with intensity features
- Search has to go on for a robust feature extraction method that is invariant to different speakers, orientation, lighting conditions and occlusions.

2.3 Related topics

This section will scratch the surface of some topics that are related to automatic lip reading. We will talk about speech animation, sign language recognition and audio-visual speech recognition.

2.3.1 Speech animation

Lip synchronization of computer generated talking faces often shows an amazing accuracy. Although the 3D models of the lips used for these cannot be mapped directly onto the typical 2D video footage in actual data corpora, it's interesting to see the parallels between both areas.

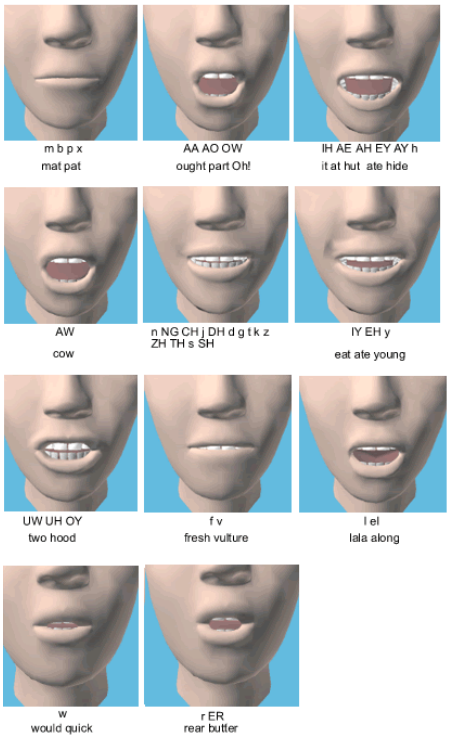


Figure 2.1: Viseme set of Annosoft Lipsync Tool 3.0 (http://www.annosoft.com)

Applications of speech animation can be found in entertainment and long-distance communication. In entertainment it is used to make virtual actors of movies and video games "talk" (see Figure 2.1). In communication, talking faces can provide an extra medium where bandwidth costs make an actual video phone infeasible. A talking head is then made to synchronize with the speech. In a similar application, talking faces could enable hearing-impaired to talk on the phone, be it with a small delay.

Generally, "visemes" (in the sense of mouth positions) are used to provide the key frames for synthesis of visual speech. Research topics in speech animation are modeling the speech apparatus accurately, lip synchronization, and interpolation between animation key frames. Examples of speech animation software are Baldi (CSLU toolkit, the more recent version is known as CUAnimate), Xface (http://xface.itc.it), and many more.

2.3.2 Sign language recognition

In much the same way as automatic lip reading, sign language can be recognized using an HMM implementation, as seen in [4]. Here visual speech recognition is applied to sign language instead of the face. A practical difficulty is the segmentation of the video; the position of the hands needs to be clear. This can be done by applying a color filter for skin tone. An advantage of using sign language over normal speech is that sign language is especially designed for the visual medium, making it potentially much more accurate. Although the applicability in daily life is limited - most people don't even know sign language; much less produce it - understanding the deaf without knowledge of sign language is made possible.

2.3.3 Audio-visual speech recognition

The type of multimodal speech recognition that appears to have the most potential is audio-visual speech recognition, because sound and vision are the two most prominent channels through which speech is communicated. Often the chosen approach is to use the video signal to enhance the results of acoustic speech recognition.

At the time of the work described in section 2.2, some additional work was done on audio-visual speech recognition [5]. A bimodal speech recognizer needs to combine

the two modalities. Ways to do this are early integration (combining the audio/video features), late integration (combining output of 2 recognizers) or intermediate integration (the road in-between). Both early and intermediate recognition requires an HMM to be trained on both media.

It was decided to extend a speech recognizer trained on the Polyphone speech corpus [6] with a state-synchronous multi-stream HMM architecture. The Polyphone recognizer was first trained using the audio from the audio-visual speech corpus to adjust to the better audio quality. The lip reading was added using feature fusion (early integration) and the system was retrained using bimodal input. In the end, the lip reading originally did not boost the performance, but when the signal-to-noise ration (SNR) of the signal was less than 8 dB, it helped the system cope.

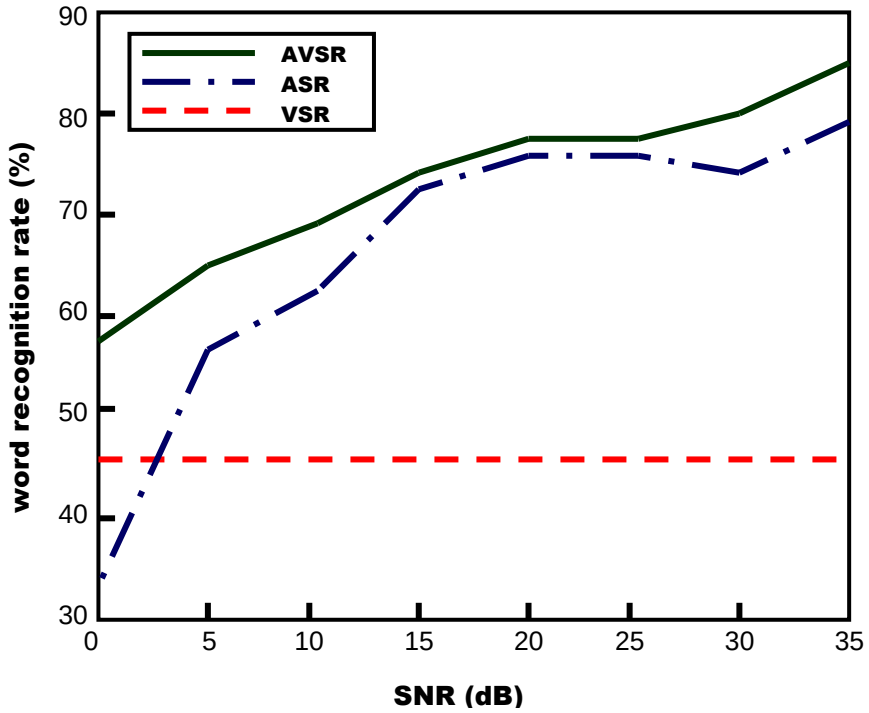


Figure 2.2: How a typical audio-visual speech recognition (AVSR) system benefits from visual speech recognition (VSR) at low signal-to-noise ration (SNR) for audio, compared to normal automatic speech recognition (ASR). [7]

2.4 Overview of audio-visual speech corpora

One of the main bottlenecks in speech recognition, and even more so in visual speech recognition, is the lack of a sufficiently large and representative data corpus. In our case, we would obviously need a speech corpus to contain video, but in evaluating a speech corpus with the application of automatic lip reading in mind, there are several other criteria that are not to be overlooked. Some of the important issues spotted were the resolution of the video recordings, the video sample rate, the richness of the language pool and last but not least the size of the corpus. All of these need to be considered to evaluate a data corpus.

Table 2.1: Overview of existing speech corpora according to [8]

Corpus	Language	Sessions	Number of speakers	Audio Quality	Video Quality	Language Quality	Stated purpose
TULIPS1	English	1	12: 9 male, 3 female	11.1kHz, 8bits controlled audio	100x75, 8bit, 30fps mouth region	first 4 digits in English	small vocabulary isolated words recognition
AVletters	English	1	10: 5 male, 5 female	22kHz, 16bits controlled audio	80x60, 8buts, 25fps mouth region	the English alphabet	spelling English alphabet
AVOZES	English	1	20: 10 male, 10 female	48kHz, 16bits controlled audio	720x480, 24bits, 29.97fps entire face, stereo view	digits from '0' to '9' continuous speech application driven utterances	continuous speech recognition for Australian English
CUAVE	English	1	36: 19 male, 17 female	44kHz, 16bits controlled audio	720x480, 24bits 29.970fps passport view	7,000 utterances connected and isolated digits	continuous speech recognition
Vid-TIMIT	English	3	43: 24 male, 19 female	32kHz, 16bits controlled audio	512x384, 24bits, 25fps upper body	TIMIT corpus 10 sentences per person	automatic lip reading, face recognition
DAVID	English	12	258: 132 male, 126 female (in 4 groups)	?	entire face, upper body, profile view multi corpora: controlled and degraded background, highlighted lips	vowel – consonants alternation, English digits	speech or person recognition
IBM LVCSR	English	1	290 - unknown gender	22kHz, 16bits ?	?	connected digits isolated words	audio-visual speech recognition
AVICAR	English	5	100: 50 male, 50 female	48kHz, 16bits, 8channels 5 levels of noise car specific	4 cameras from different angles, passport view car environment	isolated digits, isolated letters, connected digits, TIMIT sentences	speech recognition in a car environment
DUTAVSC	Dutch	10 to 14	8: 7 male, 1 female	48kHz, 16bits, controlled audio	384x288, 24bits, 25fps lower face view	spelling, connected digits, application driven utterances, POLYPHONE corpus	audio-visual speech recognition, lip reading

2.4.1 Criteria to evaluate speech corpora

The first criterion to evaluate an audio-visual speech corpus is the language of a data corpus. As much as one would like to, speech data of one language cannot be mapped onto, added to or transferred to a data set of another language. While there are a number of English speech corpora that include video, for Dutch there is only DUTAVSC, the Dutch Audio-Visual Speech Corpus, recorded at this university several years ago. [9]

The size of the corpus in terms of number of recording subjects is also important. Especially if one wants to make a speaker independent recognizer, a diverse group of subjects is key. It is also important that the group of subjects is composed of both males and females. If either group is underrepresented, it might lead to bad recognition with respect to that group. This is especially important for audio, since female voices are generally higher pitched. For lip reading we expect some variance in the measures of the facial features.

Other important factors are the quality of audio and video. Due to the requirements on video storage it might be tempting to compress the data heavily, introducing artifacts. But for automatic lip reading computer vision has to be performed, so the video should be of a decent quality concerning lighting and resolution. In section 2.4.4 we will go into the frame rate issue more deeply. It is one of the main reasons why we decided to devise a new data corpus.

Another thing is the language coverage of the corpus. Most audio-visual corpora shown in *Table 2.1* are meant to be used for simple word-level recognition tasks. We would like to see a corpus that can also be used for continuous speech recognition, or natural language. Also, the corpus had to be available in order for us to be able to use it in our research. Some corpora were available to us online (e.g. VidTIMIT (http://www.itee.uq.edu.au/~conrad/vidtimit)), but after consideration we decided that the video quality wasn't good enough (especially the resolution of the mouth). Another available corpus was DUTAVSC, which we will elaborate on in section 2.4.3.

2.4.2 Storage of visual speech data

An obstacle those wanting to implement visual speech recognizers often seem to run into is a lack of training data. However, there is a reasonably explanation for this: the amount of disk space needed to capture and store such a corpus. In comparison, the entire Polyphone corpus (1994, [10]) is 330 MB. The size of DUTAVSC (2001) is under 10 GB. The size of the new data corpus we will discuss in chapter 5 (2008), is 1580 GB. In short, the advancement in computer technology only recently enabled us to develop a larger data corpus.

2.4.3 Dutch Audio-Visual Speech Corpus

The DUTAVSC speech corpus is a small audio-visual data corpus that was recorded for the previous project. Therefore, we had full access to it. DUTAVSC stands for Dutch (or Delft University of Technology) Audio-Visual Speech Corpus [9] and contains 8 sessions of 8 different people speaking. This amounts to a total of over 4 hours of constant recordings (between 25 and 45 minutes per subject). The recorded subjects are all native Dutch speakers: 7 male subjects and one female.

The language covered by the data corpus consists of words and sentences. Per prompt set, of which between 10 and 14 were gathered for each subject, there were the utterance types given in Table 2.2 and illustrated in Figure 2.3. For some prompt sets subjects were asked to speak fast, speak extra clearly, or whisper.

The recording subjects were asked to read prompts shown on the screen of a laptop in front of a digital video camera. The operator controlled the progress of the prompts. The camera used was a SONY TRV20E digital camcorder on standard DV tapes equipped with Cassette Memory chips, placed on a tripod. An external computer microphone was used, which was hung on the speakers' neck. The audio was recorded at using a sampling of 44 kHz with 16 bit resolution. For use in these

experiments the audio files were converted to 8 bit A-law format. The video was sampled at 25 Hz.

Table 2.2: DUTAVSC prompt set

Number	Utterance type
1	Sentence of 10 short unrelated words
10	Phonetically rich sentences from Polyphone
3	Random digit sequences of length 10
4	Spelled words
5	Sentences with fixed grammar (see Figure 2.4)

Each of the recorded sessions was edited using video editing software and cut into smaller sequences. The video sequences were then converted from a standard DV format to MPEG1 stream. Moreover, from all of the scenes audio data was extracted and saved externally. Furthermore, the proper transcriptions of the utterances were added.

After the recordings were made, data from 5 out of 8 subjects was transcribed and used for experiments on audio-visual speech recognition. [5] From each subject 4 or 5 sessions were used. This data set was split in a training set of approximately 500 utterances from all speakers and a test set containing 30 utterances from all speakers.

During our own project we transcribed the data of the remaining 3 subjects, with the initial purpose of using it for our project. The dataset will be made available online for researchers throughout the world.

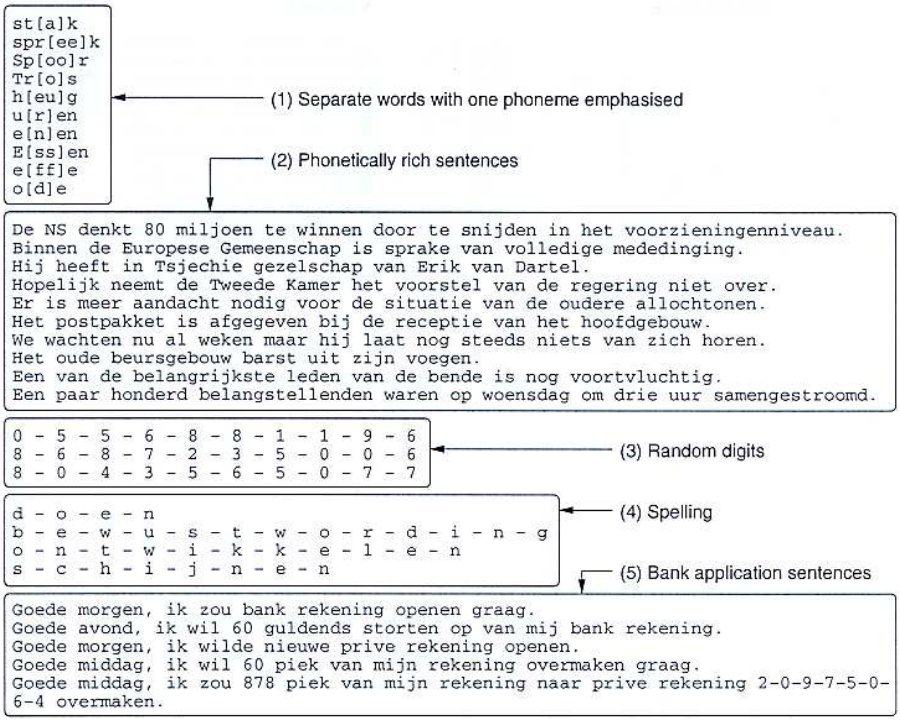


Figure 2.3: Example prompts illustrating language coverage of DUTAVSC according to [5]

```
$number10 = twee | drie | vier | vijf | zes | zeven | acht | negen;
$number20 = tien | elf | twaalf | dertien | veertien | vijftien | zestien | zeventien |
achttien | negentien;
$number100 = [(1 | $number10) en] (twintig | dertig | veertig | vijftig | zestig |
zeventig | tachtig | negentig);
$number =
        [$number10] honderd [en] ($number100 | $number20 | $number10 | 1) |
        [$number10] honderd |
        $number100 |
        $number20 |
        $number10 ;
$digit = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9;
$amount = $number (euro | euro's) | een euro | 1 euro;
$greeting = goedemorgen | goedemiddag | goedenavond;
$please = alstublieft | alsjeblieft;
$want = wil | wilde | wou;
$type = [prive] [bank] rekening;
$account =
        [mijn] $type [nummer] $digit $digit $digit $digit $digit $digit $digit $digit |
        (mijn | m'n | een) $type;
$action =
        $amount van $account [naar $account] overmaken |
        $amount op $account storten |
        $amount storten op $account |
        $amount opnemen van $account |
        $amount van $account opnemen|
        een [nieuwe] $type openen |
        $account sluiten;
([$greeting] ik $want [graag] $action [$please] |
        [$greeting] ik ($want | zou) $action graag |
        [$greeting] ik zou graag $action [$please])
```

Figure 2.4: Telebanking application grammar in EBNF used to generate prompt for DUTAVSC [5]

2.4.4 Video frame rate issues

One of the main reasons why we decided to record a new data corpus is the poor coverage of visual speech by traditionally used video sample rates. For the corpora in Table 2.1, audio was sampled at around 48 kHz, which results in 100 Hz once feature extraction is performed using MFCC with a Hamming window of 30 ms, while video was recorded at a frame rate of 25-30 Hz. For a human observer, 25 frames per second is assumed to be sufficient to perceive fluent motion [11]. However, human lip readers benefit from higher frame rate [12], so it can be assumed that a machine lip reader would as well.

In audio-visual speech recognition, one tries to merge the auditory and visual channels. If the sample rates for the auditory and visual channels are not equal, they have to be synchronized. This can be done by interpolation or copying video frames. This is without taking into account that for the auditory speech units and their visual counterparts, although dependent, their timing may be off: visual evidence of sound production may show before the sound is heard. [13]

Another problem lies in the model one would like to impose on the data. Figure 2.5 shows the duration of a great number of visemes appearing in some speech fragment. For DUTAVSC, the video was recorded taking 25 frames per second, so the majority of visemes (the visual counterparts of phonemes as will be discussed in chapter 4) occupy 2-3.5 time frames (0.08-0.14 seconds). This is troublesome, because Hidden Markov Models that represent a speech unit should have at least the as many states as the number of time frames required for the shortest instance of

that speech unit. As seen in the figure, the majority of the speech units cannot be modeled by an HMM with 3 states or less.

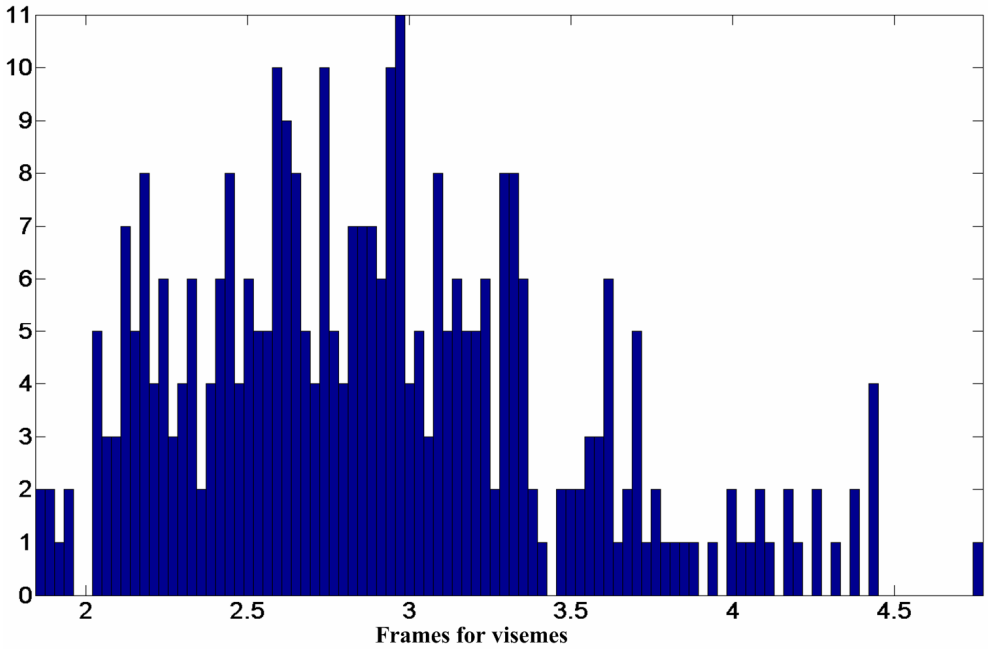


Figure 2.5: The number of time frames per viseme, in case of fast speech rate based on analysis of DUTAVSC and artificial data [14]

A study was done to see which sample rate would be needed to capture all speech information in a video signal [14]. The lip reading accuracy is estimated based on the Root Mean Square Deviation measure.

$$RMSD\left(\hat{\theta}\right) = \sqrt{MSE\left(\hat{\theta}\right)} = \sqrt{E\left(\left(\hat{\theta} - \theta\right)^2\right)} \qquad (2.4)$$

Both real high-speed data and synthetic data generated using CUAnimate (CSLU Toolkit) were analyzed using two types of features; mouth width and height, and optical flow. The conclusions with respect to speech rate are that the gain on low speech rate is not significant enough to justify the extra use of resources implied by recording at high speed. But when the speech rate increases, a recording rate of 24 to 30 frames per second is definitely insufficient.

2.4.5 Using side view images

In most work, we see visual features being extracted from images of the speaker taken from the front. There are however some possible applications in which the camera is not positioned directly in front of the speaker. Mapping identified face points on a 3D model of a human head for example, would require more than just the front view. Also, automatic lip reading from a telephone would almost certainly not result in the front view. Besides, it is not impossible to read lips from another viewpoint. Humans for example are very much capable of it.

Several experiments have been conducted to decide the success of features extracted from profile view video, for example [15]. Here, they extract simple geometric features from front (upper lip height, lower lip height, and lip width) and

profile view (maxima of the face contour resulting in both lips, nose and chin) and compare them. The conclusion is that profile view features (WRR: 40%) perform about 10% better than front view features (WRR: 30%).

In [16] an experiment is described where optical flow features extracted from side images are used in audio-visual recognition. They don't say anything about the individual performance of the lip reading, but the combination is more robust against noise.

2.4.6 Conclusion

After analysis of existing speech corpora, we arrived at the conclusion that none of them were sufficient to meet our goals and that recording a new data corpus was required. The new speech data corpus received the code name New DUTAVSC.

The recording of New DUTAVSC had to be carefully prepared. Because of the scale of the new corpus we needed to make sure that all material would be gathered in a correct way. The new corpus would have to follow the quality requirements we gathered after the research done on other speech corpora; a high frame rate, capturing of both frontal and profile view of the face and a rich utterance pool fit for continuous speech recognition experiments. Furthermore, we decided that it would be useful if people would be recorded at both normal and fast speech rate, and whispering (instead of low speech rate, as it appears to be more natural).

We decided to record at a frame rate of 100 Hz, which would give four times as much speech unit coverage as for DUTAVSC, which as recorded at 25 Hz. While one might decrease the frame rate of a recording during analysis, it is not possible to add data that was never recorded.

For our new data corpus, we decided to capture both frontal and profile views of the lower half of the face. This will keep all options open for mapping onto and recognition from a 3D model of the frontal hemisphere of the face, and any other research people might think of.

2.5 Methods for feature extraction

One aspect none of the existing visual speech recognizers have in common is their feature extraction method. Computer vision techniques from a wide range of origins can be applied to this task. Feature extraction is probably also the most important part of the scientific endeavor and thus the most visible one with the most development. The best feature extraction method has yet to be determined and in many cases seems to depend on the application.

Feature extraction for visual speech often comprises two steps: locating the face and mouth and then extracting the actual features used for classification. Since the nature of the features that can be extracted depends heavily on the representation of the face after localization, these two problems have to be seen as one approach. That is why we have taken tracking and feature extraction together in this section.

2.5.1 Classification of feature extraction methods

Feature extraction needs to be performed for two reasons:
1. to reduce the dimensionality of the classification task,
2. to capture the relevant information about the process being modeled.

We will use the categorization introduced by a paper on active shape models by Luettin et al. [17] Two main approaches for extracting speech information from image sequences are the image-based approach and the model-based approach. Combinations of these are also widely used.

In the image-based approach, the image intensities are preprocessed and then used as the feature vector. Preprocessing normally consist of filtering and dimension reduction. The advantage of this approach is that no data is thrown away. Disadvantage is that the data is not normalized and the high dimensionality and high redundancy of the feature vector. The data driven approach does not use any previously defined model of the lip area and attempts to capture information based on the data alone. An example is optical flow analysis, which only uses the motion of an image sequence. Also mentioned is the processing of raw data which preserves a lot of details about the speaker like skin textures, but this only makes sense for speaker recognition, not lip reading.

In the model-based approach, a model of the visible speech articulators, mainly the lip contours, is built and its configuration is described by a small set of parameters. The advantage of the model-based approach is that important features are represented in a low dimensional space and are invariant. A disadvantage is that a particular model may not consider all relevant speech information. The main difficulty in the model based approach is to build a model which represents the lip shape efficiently and which is able to locate and track the lip contours of different speakers and under different illumination conditions. The model-based approach seeks to first make a model of the features of the face that could contain speech, allowing for a compact notation. It is difficult to come up with a generic lip model though. The feature set then consists of a set of parameters for such a model.

2.5.2 Optical flow analysis

Optical flow is a data driven approach to video tracking and feature extraction. It is used widely in video compression standards. A common definition of optical flow is "the distribution of apparent velocities of movement of brightness patterns in an image". From the differences between subsequent images, a guess is made about the movement between images, resulting in a grid of motion vectors. Video compression can then be accomplished by storing just the starting image and the motion vectors.

A motion field of this kind contains one motion vector for each block of pixels. Optical flow can also be used as a source for speech features. After localization, only the motion field around the mouth is considered.

There are many algorithms for optical flow recovery differentiated by the assumptions taken to alleviate the mouth cavity problem (sometimes the mouth is open, sometimes it is closed). However, the most used and accurate is the algorithm developed by Lucas and Kanade. [18]

Optical flow has been applied successfully to visual speech recognition. The optical flow features directly recover the motion information apparent around the speaker's mouth. The idea is to split the region of interest into a number of interesting zones, then compute and statistically describe the optical flow in each zone. It was shown that the motion vectors are better at describing the mouth movement than the delta

and acceleration of static features. However, it can be very slow, especially when the number of chosen motion vectors is very high.

2.5.3 Lip geometry estimation

Lip Geometry Estimation is a combination of the image-based and the model-based approach. LGE was designed by Jacek Wojdeł to obtain robust lip features [19]. One advantage is that no geometrical lip model needs to be defined in advance. In this approach, the geometry of the mouth is represented by an estimate of some of its statistic properties, making it insensitive to noise and personal characteristics of the speaker.

With Lip Geometry Estimation, two types of features are computed; geometric features and features based on area. Geometric feature extraction starts by identifying the region on interest by color filtering the image to locate the lip pixels. Hue, hue-value, grayscale, pseudo-hue histogram, and simple artificial neural networks based on different color spaces were tried. The best results were obtained with RGB based neural networks. More statistical methods were used to further reduce the artifacts of the filtered image (outliers).

After that, the center of gravity of the lip pixels is used as the center point around which the image is transformed into polar coordinates. The resulting intensity function has two interesting properties; its conditional mean (thickness of the lips) and variance (distance from the center of the mouth) for specific angles, thus describing the shape of the lips:

$$J(\alpha, r) = I\left(x_{center} + r\cos(\alpha), y_{center} + r\sin(\alpha)\right) \tag{2.5}$$

The formula to calculate the mean from equation *(2.5)*:

$$M(\alpha) = \frac{\int_r J(\alpha, r)\, r dr}{\int_r J(\alpha, r)\, dr} \tag{2.6}$$

The formula to compute the variance from equation *(2.5)*:

$$\sigma^2(\alpha) = \frac{\int_r J(\alpha, r)\left(r - M(\alpha)\right)^2 dr}{\int_r J(\alpha, r)\, dr} \tag{2.7}$$

Sampling this function in 18 predefined directions provides the geometrical features. The feature set can be extended by adding area features to obtain better recognition results. These features are based on the visibility of the teeth and tongue in a video frame. To determine the area occupied by the teeth, a color intensity filter is simply used on the region of interest (bright pixels are probably the teeth). The tongue is about the same color as the lips, but using an intensity filter to find the mouth cavity area (which is dark) and comparing it to the full area of the lips, the area occupied by the tongue can still be computed. Besides the areas, also the position of their centers of gravity relative to the center of the mouth are added to the feature vector, which ultimately forms the input to an artificial neural network or Hidden Markov Model. Including these intensity features increased performance from 60% to almost 80% for simple recognition tasks.

Although this method is supposedly speaker-independent, recognition experiments done using LGE decreased in performance when recordings of multiple people were used for training. Reasons for this could be that the method may not be robust to lighting conditions, occlusions and variations skin tone and presence of facial hair. We also do not see how it could handle lip shapes that also vary per person.

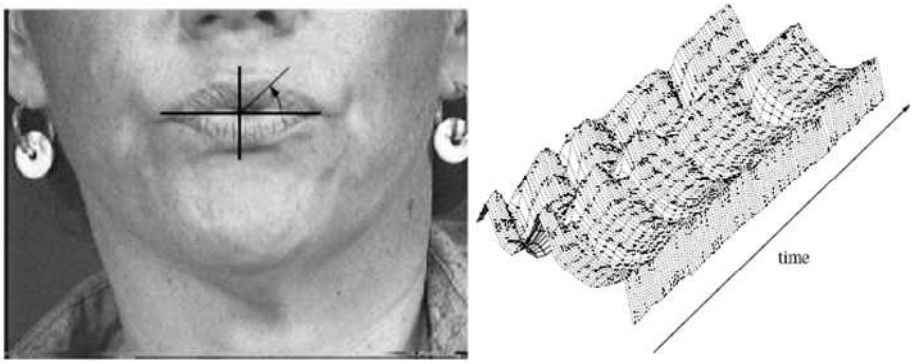


Figure 2.6: Lip geometry estimation applied on a video frame [3]

2.5.4 Active Shape Models

This approach is model-based but the training phase can be considered image-based. Active Shape Models are statistical models that approximate the shape of an object. It is used in computer vision to locate objects in new images. This is done by iteratively reforming the model, but with the constraint to vary only in ways seen in a training set of labeled examples. The algorithm makes use of the appearance of an image by assuming the points lie on edges.

Going from an initial estimate for the pose and shape parameters (e.g. the mean shape), a variant on the Expectation Maximisation algorithm is applied. The shape is iteratively updated as follows:

- Look along the normals through each model point to find the best local match for the model of the image appearance at the landmark (e.g. strongest nearby edge)
- Update the pose and shape parameters to best fit the model instance to the found points
- Repeat until convergence

This method has been applied successfully to lip reading. Luettin et al. [17] describe a model-based speech reading system where a model of the lips is constructed from a training set. The model is used to subsequently locate, track and parameterize lip contours in image sequences. These are the lip shape features that are handed to a HMM modeling visual speech.

Active shape models are flexible models which represent the boundary or other significant location of an object by a set of labeled points. ASMs use a priori knowledge about shape deformation from the statistics of a training set which was labeled by hand. PCA is used to map the main modes of shape variation into a linear subspace. Any shape can then be approximated by a linear combination of the "mean" shape and the first few modes of variation.

The parameters describing the shape of the lips are extracted at each time frame and used as feature vectors. They are invariant to scale, rotation, translation and illumination (parameters associated with these are not used) and can directly be used by the recognition network. Much speech information is contained in the dynamics of the lip movement rather than the actual shape. Therefore the delta shape parameters are added as features. Also the delta scaling parameter was added, even though scaling varies per speaker. Visual speech is modeled by representing each utterance as a sequence of visual speech vectors. Their emission probabilities are modeled by continuous Gaussian distributions and temporal changes are modeled by HMM.

They achieved a 85% recognition rate by using whole-word HMM with 5 or 6 states with the TULIPS1 corpus (see Table 2.1), which contains just the first four digits, but of which it is known that untrained humans performed at a word recognition rate of 90%, while trained lip readers perform at 95%. The nicest part is that these features are speaker-independent.

2.5.5 Introducing Active Appearance Models

The Active Appearance Model (AAM) is a generalisation of the Active Shape Model that uses all texture information of an image, instead of just the edges. The ASM essentially seeks to minimize the distance between model points and the corresponding points found in the image, whereas the AAM seeks to minimize the difference between the synthesized model image and the target image.

Because of this, AAMs have the following advantages over Active Shape Models:

- It is more robust because it explicitly minimizes texture errors.
- It takes advantage of all the grey-level information available across an object, making it more reliable.
- A convincing model can be built with a relatively small number of landmarks. Any extra shape variation is expressed in additional modes of the texture model. The ASM needs points around boundaries so as to define suitable directions for search.

Both methods have the drawback that an amount of labeled training examples is required to build a good model. They have the advantage that they are well suited to track objects through image sequences, using the previous frame as initiation of the next frame. This way only a few iterations are required to lock on.

We ended up using this method for mouth tracking because of its speed, robustness and convenience. More will be explained about Active Appearance Models in chapter 6.

2.5.6 Conclusion

After researching these methods (and seeing some of them put to action), there were some criteria that made up our mind about which technique to use for our goals. Feature extraction can be a real time bottleneck in an automatic lip reader that is meant to run live. Especially in a data driven (and thus computationally expensive) approach like optical flow this proved to be true. The more motion vectors were used, the worse the performance became. For the other method that works without a predefined model, LFG, the performance, although better than for OF, is still not that good because still whole images are processed each time step.

For the model-based point tracking methods, ASMs and AAMs, the difference was that for AAM fewer landmark points were required to build a model. Because of the considerable work required to get reliable image labeling, this is an advantage. Furthermore, there was an implementation readily available [20], which would save implementation time. Point tracking proved to be faster than the other methods.

Other ways we explored to perform data parameterization are optical flow and lip geometry estimation. Neither were as fast as AAMs, and because they were not based on point tracking resulted in many more model parameters (which would require more training data to train properly).

The reasons why we choose to use Active Appearance Models as our method for point tracking are the promised speed and robustness of the tracking. There was also an implementation readily available to prove this point. It was faster than other investigated methods for data parameterization.

3 Basics of speech recognition

How does one build an automatic speech recognizer? While this may not have been trivial question in the past, today a standard approach exists. The purpose of this chapter is to give a basic introduction to the theory and techniques that are usually applied to speech recognition problems. For automatic lip reading, this basic approach is the same. Specific aspects of visual speech recognition will be discussed in chapter 4.

In section 3.1 the problem of language modeling, which is fundamental to speech recognition, is introduced. In section 3.2 we discuss the application of Hidden Markov Models (HMM) to speech recognition. Training (3.3) and recognition (3.4) using these Markov models are discussed after that. In section 3.5 we talk about acoustic modeling in some more detail. In section 3.6 we introduce the tool we used to implement our automatic lip reader: the Hidden Markov Model Toolkit (HTK).

3.1 Statistical speech recognition

In the past, there have been different approaches to speech recognition. Artificial intelligence techniques such as expert systems, pattern matching and neural networks have been applied with mixed results. To date, the most successful approach appears to be the statistical one. Statistical speech recognition is based on the fact that words, or sub-word units like phonemes, are mutually dependent. The probability that certain words are observed given the previous words, can be captured in a language model.

According to this approach, the problem of speech recognition can be solved if the most likely sequence of (sub) words W given an observation sequence O is found, which is given by equation (3.1).

$$\hat{W} = \arg\max_{W} P(W \mid O) \qquad (3.1)$$

In this equation, P(W|O) is the probability of a word sequence w given observation sequence O. The word in W for which this probability is maximized is $\hat{W}$, the most likely utterance. This is however not directly computable, but using the Bayes rule this formula can be rewritten as equation (3.2).

$$\hat{W} = \arg\max_{W} P(W) P(O \mid W) \qquad (3.2)$$

Here, P(W) is the a priori probability that the word string W is uttered, called the language model and P(O|W) is the probability that when a (sub)word string W is uttered evidence O will be observed, called the likelihood. P(O), which would appear in the denominator after application of the Bayes rule, can be disregarded because of the properties of the maximum operation.

The language model P(W) should provide a way to calculate the probability that the (sub)word sequence W will be uttered in the language as a whole. For a single word w_i the probability depends on all previously recognized words. The bigram grammar

provides a simple and effective method to calculate this probability using the following equation:

$$P(W) = P(w_1)P(w_2 \mid w_1)...P(w_n \mid w_{n-1}) \qquad (3.3)$$

The probabilities $P(w_i \mid w_{i-1})$ can easily be estimated by counting the occurrence of each word pair in a representative and preferably huge (text-based) data corpus. Bigrams assume the current word only depends on the previous word. A better approximation of the language could be obtained by extending this to n-gram grammars that depend on n-1 previous words (trigrams or more), but apart from needing an even larger data corpus to train it properly, the most widely used algorithm for recognition, Viterbi, discussed in section 3.4, is unable to work with temporal dependencies exceeding bigrams.

A language model like n-grams is only required for continuous speech recognition that can recognize natural sentences. If the data consists of separate words or is bound by a fixed grammar, a simpler language model can be used.

In case of acoustic speech recognition, the likelihood P(O|W) is often called the acoustic model. It determines what sounds or observations will be produced when a given string of words is uttered. One way to model this is by using Hidden Markov Models.

3.2 Hidden Markov Models

The Hidden Markov Model (HMM) is a powerful mathematical tool to model time series. It is a finite state machine in which the system being modeled is assumed to be a Markov process. The state sequence however cannot be directly observed. Apart from state transition probabilities, an HMM also has emission probabilities, which means that for the same state sequence, numerous observation sequences are possible. The HMM can model the varying duration that is common for speech units well.

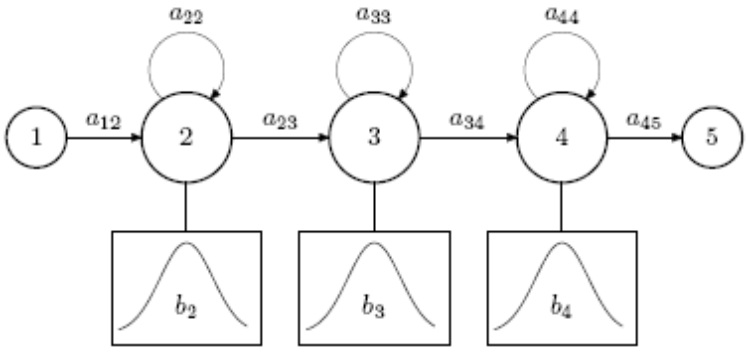


Figure 3.1: A standard three-state Hidden Markov Model, with non-emitting states 1 and 5, and emitting states 2, 3, and 4. Emitting states are associated with an output distribution b_i [21]

A Hidden Markov Model (see Figure 3.1) consists of a number of states, a number of state transition probabilities stored in a transition probability matrix A, and output

distributions $b_t(o_t)$ for each emitting state. For an HMM to learn all of these probabilities, is not a trivial task: it can only be estimated. An algorithm that does just that will be explained in the next section.

It should be clear how an HMM can generate an observation sequence O by traversing states, but in speech recognition, we already know the observation sequence and are only interested in the underlying process that generated those observations, i.e. the (sub)word sequence. Because we are using a Hidden Markov Model the state sequence is not directly visible to us though. In fact, there can be many solutions to the problem we want to solve. The only approach we can take here is trying to find the most likely underlying state sequence given the observations made.

Although the state transition probabilities and emission probabilities of an HMM can be estimated by an algorithm, the topology of an HMM can only be designed by hand. The architecture of the HMM to use depends greatly on the primitives of speech one wants to recognize, and the available training data. If words are used as speech units and each word is represented by one HMM, within-word co-articulation effects are well modeled, but training data will be required for every word, and retraining is required for every word added to the dictionary. Usually sub-word units like phonemes are used as units of speech so data can be shared among words.

Each unit of speech will be modeled by its own HMM. For phonemes, usually triphones are used to model the relation to surrounding phonemes, which should partially account for between-phone co-articulation. An HMM should not have too many states; adding more states means introducing more parameters and thus more degrees of freedom. Variations in sub-word units can be modeled more accurately but this also requires more training data to avoid undertraining. If a sub-word unit only occupies a limited amount of time frames the HMM should not have more states than that, or there should be short-cuts. [5]

3.3 Training: Baum-Welch re-estimation algorithm

One algorithm that can be used to train a Hidden Markov Model, although it is not guaranteed to give the best possible solution) is the forward-backward or Baum-Welch algorithm. The forward-backward algorithm will let us train the transition probabilities a_{ij} and emission probabilities $b_t(o_t)$ of an HMM.

For a transparent Markov Model for which every state emits a fixed symbol, we could find the transition probabilities by counting the times a transition occurs. For HMMs, the approach is to iteratively estimate these counts. It starts with an estimate for transition and emission probabilities, and refines the estimated probabilities by computing the forward probability for an observation and dividing the probability mass among the different paths that lead to this observation, with the previously estimated transition probabilities as weighing factors. The forward probability is the probability of being in state i after seeing the first t observations. It can be computed by the algorithm displayed in Figure 3.2, which is closely related to the Viterbi algorithm.

Similarly, the backward probability is the probability of seeing the observations from time t + 1 to the end, given that we are in state j at time t.

```
function FORWARD(observations, state graph) returns forward-probability

num-states <- NUM-OF-STATES(state-graph)
num-obs <- length(observations)
Create probability matrix forward[num-states+ 2, num-obs + 2]
forward[0, 0] <- 1.0
for each time step t from 0 to num-obs do
       for each state s from 0 to num-states do
              for each transition s' from s specified by state-graph
                     forward[s', t + 1] <- forward[s', t + 1]  + forward[s, t] *
                     a[s, s'] * b[s', o_t]
return the sum of the probabilities in the final column of forward
```

Figure 3.2: Forward algorithm for computing likelihood of observation sequence given a word model. a[s, s'] is the transition probability from current state s to next state s', and b[s', o_t] is the observation likelihood of s' given o_t [22]

On the basis of the forward and backward probabilities, the frequency of the transition-emission pair values is determined and divided by the probability of the entire sequence. This amounts to calculating the expected count of the particular transition-emission pair. Each time a particular transition is found, the value of the quotient of the transition divided by the probability of the entire string increases, and this value can then be made the new value of the transition.

Transition probabilities can be estimated by equation (3.4), and the observation probabilities by equation (3.5).

$$\hat{a}_{ij} = \frac{\text{expected number of transitions from state } i \text{ to state } j}{\text{expected number of transitions from state } i} \qquad (3.4)$$

$$\hat{b}_j(v_k) = \frac{\text{expected number of times in state } j \text{ and observing symbol } v_k}{\text{expected number of times in state } j} \qquad (3.5)$$

So, the algorithm provides ways to re-estimate these probabilities from an observation sequence O, assuming that we already have a previous estimate of a and b. The entire embedded training procedure for HMM chooses a first estimate and calculate a and b until convergence.

Pseudo-code of the algorithm to guess a state sequence given an observation sequence is given in Figure 3.2.

```
num-states <- NUM-OF-STATES(state-graph)
num-obs <- length(observations)
Create probability matrix forward-backward[num-states + 2, num-obs + 2]

FORWARD-BACKWARD (guessed initial state position, 0)

function FORWARD-BACKWARD (state s, time step t) returns forward probability

if t > num-obs then
        return 1
if forward-backward[s, t] > 0 then
        return forward-backward[s, t]
forward-backward[s, t] <- 0
for each transition s' from s specified by state-graph do
        forward-backward[s, t] = forward-backward[s, t] +
        FORWARD-BACKWARD(s', t + 1) *
        computed a[s, s'] given observation element at t
return forward-backward[s, t]
```

Figure 3.3: Simple pseudo-code representation of the forward-backward algorithm. The requirements are a state-graph, an observation sequence and guesses for the transition probabilities a[s, s'] and initial state position.

3.4 Recognition: Viterbi algorithm

Once there is a fully trained HMM at one's disposal, a common used algorithm for speech recognition is the Viterbi algorithm. The Viterbi algorithm can be used to find the most likely path through a Hidden Markov Model, as well as find the probability of the observation sequence given this most likely path. To decode (and in case of continuous speech find the word boundaries in) an observation sequence, first a large HMM is constructed that combines all words in the dictionary according to the grammar if there is one. Each cell viterbi[t, i] of the matrix contains the probability of the best path which accounts for the first t observations and ends in state I of the HMM. This is the most probable path out of all possible sequences of states of length t – 1:

$$viterbi[t, j] = \max_i \left(viterbi[t-1, i] a_{ij} \right) b_j(o_t) \qquad (3.6)$$

In order to compute viterbi[t, i], the Viterbi algorithm assumes the dynamic programming invariant or Markov property. This is the simplifying (but incorrect) assumption that if the ultimate best path for the entire observation sequence happens to go through a state q_i, that this best path must include the best path up to and including state q. This doesn't mean that the best path at any given time t is the best path for the whole sequence: a path can look bad at the beginning but turn out to be the best path. Because of this assumption the Viterbi algorithm breaks down for certain kinds of grammars, including trigram grammars.

Once the viterbi matrix and an accompanying matrix of back pointers have been constructed, the algorithm continues by looping through all time frames and states, calculating the most probable path for each "next" state. When the final time frame is reached, this leads to an optimal solution we can backtrack to find the path that led there. The probabilities are usually so small that the logarithmic scale is used to

represent them. This way, multiplication operations can be replaced by simple additions as well.

```
function VITERBI(observations of length T, state-graph) returns best-path

num-states <- NUM-OF-STATES(state-graph)
Create a path probability matrix viterbi[num-states+2, T+2]
viterbi[0,0] <- 1.0
for each time step t from 0 of T do
       for each state s from 0 to num-states do
              for each transition s' from s specified by state-graph
                     new-score <- viterbi[s,t] * a[s,s'] * b_s'(o_t)
                     if ((viterbi[s',t+1]=0||(new-score > viterbi[s',t+1]))
                     then
                            viterbi[s',t+1] <- new-score
                            back-pointer[s',t+1] <- s

Backtrace from highest probability state in the final column of viterbi[] and
return path.
```

Figure 3.4: Viterbi algorithm for finding optimal sequence of states in continuous speech recognition, simplified by using phones as inputs. Given an observation sequence of phones and a weighted automaton (state graph), the algorithm returns the path through the automaton which has minimum probability and accepts the observation sequence. A[s,s'] is the transition probability from current state s to next state s' and $b_{s'}(o_t)$ is the observation likelihood of s' given o_t. [22]

The Viterbi algorithm is an algorithm that runs in exponential time and can thus take long to get to a result. Also, the memory requirement can be large if the HMM is large (for word-level recognition for example, all words of the dictionary have to be combined into one big HMM before performing the search). Pruning of the search tree can be performed to make it more efficient.

3.5 Going into more detail: Features

To train and run a speech recognizer, data first needs to be poured in a numeric format that can be classified. Data parameterization or feature extraction has two functions: firstly and most importantly, it reduces the amount of data that needs to be processed; secondly, it makes the data more meaningful to the recognizer, and thus training easier. For an acoustic speech recognizer, recordings of sound are cut into samples according to a chosen time frame. Using the Fourier transform of the signal, the sound is split into more and less characteristic components that together make up a feature vector for every time frame, which is handed over to the HMM for recognition.

For the probability density functions of the HMM's states Gaussians are often used. For every feature eventually one Gaussian is trained per state, leading to the mean and variance of the normal distribution to be model parameters. If a Gaussian is expected not to approximate the actual probability density function, which is mostly the case, a mixture of Gaussians is used to obtain a better approximation. Added together, a multitude of normal distributions can approximate any probability function, as long as one allows enough of them to be used, and the number of Gaussians required is limited because a time frame is also limited. The model

parameters will be a number of mean and variance values in this case, one for each Gaussian. This is how most acoustic speech recognizers operate.

In the case where the signal to be recognized is not just one waveform that can be transformed, the features will have to come from a different source. Care has to be taken that the numeric format chosen for the features can accurately be modeled by an HMM. Though it is possible to have discrete values, like Booleans, continuous values are better suited to be modeled with Gaussians. Requirements for the feature set depend on what one wants to recognize. In case of video, the features should preferably be invariant to lighting conditions, scaling and rotation for example. Other requirements can be speaker independency or speed of the feature extraction algorithm. Features can be object model parameters, distances between certain points or the derivative or acceleration values thereof.

3.6 Hidden Markov Model Toolkit

The Hidden Markov Model Toolkit (HTK) is often used for automatic speech recognizer implementation, because it contains generally applicable tools for HMM-based speech processing that are optimized for speed. The tools can be run from the command line and each program has many options for customization. It is also well documented: the manual [2] contains theory, examples, tutorials and detailed descriptions of all available tools. There are tools for each step of building a speech recognizer: data preparation, training, testing and analysis.

For data preparation, some available tools are *HCopy* that can conveniently extract features from audio files (e.g. Mel Frequency Cepstral Coefficients), *HList*, that can be used to inspect for example binary feature files, and *HLed*, that can format the labels of the data.

For training, one usually starts with defining the HMM topology by writing a prototype definition. The tools *HInit* and *HRest* can be used to make the initial guess. *HERest* is used to perform embedded training using the Baum-Welch algorithm. The philosophy of system construction in HTK is that HMM should be refined incrementally. The usual process is to modify a set of HMM in stages using HMM definition editor *HHEd* and then re-estimate the parameters of the modified set using *HERest* after each stage. Also *HVite* can be used here to adapt the HMM to a speaker.

HTK has one recognition tool called *HVite*, which implements the Viterbi algorithm. It has many options, including the option to run the algorithm with live (audio) input to enable on-line recognition, and pruning search trees to speed up recognition. *HVite* requires a word network describing the allowable word sequences, possibly generated by using the *HParse* tool on a grammar in Extended Backus Naur Form (EBNF), a dictionary defining the word pronunciations and a set of HMM. The tool *HDMan* can be used to merge dictionaries.

Analysis can be done using the tool *HResults* which compares the recognition results to the labels and counts the substitution, deletion and insertion errors among other things. Also useful are the speaker-by-speaker breakdowns, confusion matrices and time-aligned transcriptions.

With all these tools at our disposal, developing an automatic lip reader should come down to processing our data, computing our features and then pouring everything into the format required by HTK. The theory seen in this chapter will be applied in

the design of our lip reader. Visual and acoustic speech recognition can both be accomplished by Hidden Markov Models, so we can use the HTK toolkit for our implementation.

4 Visual speech recognition

Training a visual speech recognizer can be done in much the same way an acoustic speech recognizer can be trained. There are however some factors that make it a fundamentally harder task. That is why we will first go over the similarities and differences between acoustic and visual speech recognition. In this chapter we will first discuss the primitives of lip reading (section 4.1) and then explore the theoretical performance boundaries of automatic lip reading (section 4.2). At the end of this chapter, we will also have reached the end of the development phase. In section 4.3 we will present our design choices and model for the rest of this thesis.

4.1 Visemes

In this section we will discuss the "viseme" (introduced in section 2.1). In speech recognition systems, HMMs can be used to model sub-word units. This way a recognizer does not have to be retrained when items (with their representations) are added to the dictionary. For acoustic speech recognition, phonemes are used to represent words. They will be discussed in section 4.1.1. For visual speech, not as many sub-word units can be distinguished as there are phonemes. In order to still be able to use the phoneme representations of available dictionaries (which are essential to train a recognizer), a mapping from phonemes to visemes can be applied. There is however no general agreement to which mapping this should be. In section 4.1.2 we present the results of a study of which viseme classes can be distinguished by humans, and in section 4.1.3 the final mapping we applied for our automatic lip reader.

4.1.1 Phoneme set

For the Dutch language, around 40 phonemes are distinguished, one for each consonant or vowel. Table 4.1 shows the consonants and Table 4.2 shows the vowels acknowledged by the dictionary of the Polyphone speech corpus [10]. There exist different types of notation for these phones. IPA and SAMPA are internationally recognized with the big advantage of SAMPA being that the transcriptions are in ASCII format. However, HTK doesn't allow all the symbols of SAMPA to be used. That is why we had to embrace an alternative notation (found in the "HTK" column of the table). Through this text, we will try to be consistent and use the SAMPA notation.

It has to be noted that not all language research agrees on the same phoneme set. Not all phonemes included here are native to the Dutch language for example. Foreign phonemes are usually only used in "loan" words. Sounds that have their origin in French (g, Z, E:, 9:, O: in SAMPA notation) have become part of the language, but are still underrepresented compared to native Dutch sounds. This makes it hard for a speech recognizer to learn the statistical models (HMM) for those phonemes reliably.

We used the natural sentences that originated from the Polyphone transcriptions as prompts for recording New DUTAVSC. It therefore makes sense to use the same dictionary and phoneme set as the one available.

Table 4.1: Phoneme set: consonants

IPA	SAMPA	HTK	Example	Phonetic transcription
p	p	p	pak	p a k
b	b	b	bak	b a k
t	t	t	tak	t a k
d	d	d	dak	d a k
k	k	k	kap	k a p
g	g	gg	goal	gg oo l
f	f	f	fel	f e l
v	v	v	vel	v e l
s	s	s	sein	s ei n
z	z	z	zijn	z ei n
x	x	x	toch	t o x
ɣ	G	g	goed	g u t
ɦ	h	h	hand	h a n t
ʒ	Z	zj	bagage	b a g aa zj at
ʃ	S	sh	sjaal	sh aa l
m	m	m	met	m e t
n	n	n	net	n e t
ŋ	N	nn	bang	b a nn
l	l	l	land	l a n t
R	R	r	rand	r a n t
ʋ	w	w	wit	w i t
j	j	j	ja	j aa

4.1.2 Human viseme classification

Before trying to establish what a good mapping between phonemes and visemes would be, it might be a good idea to explore first which classes humans can distinguish. If a trained human lip reader cannot see the difference between certain spoken phonemes, there is no reason to believe that a computer could. Humans will probably always be the reference point when it comes to language processing tasks.

Most viseme classifications are deduced from linguistics theory. Here we will discuss a paper that bases classification on empirical evidence. Van Son et al. [23] describe an experiment where they try to determine three things that could all be of use to our project to some extent:

1. A general viseme classification for Dutch vowels and consonants (which could help us find a viseme set),
2. The relation between this classification and acoustic speech cues (which could help us decide which features would be valuable), and
3. The effect of lip reading expertise on viseme categorization (which could give us a reference point for performance at similar tasks).

Table 4.2: Phoneme set: vowels

IPA	SAMPA	HTK	Example	Phonetic transcription
I	I	i	pit	p i t
ɛ	E	e	pet	p e t
ɑ	A	a	pat	p a t
ɔ	O	o	pot	p o t
ʏ	Y	y	put	p y t
ə	@	at	de	d at
i	i	ie	vier	v ie r
y	y	yy	vuur	v yy r
u	u	u	voer	v u r
a:	a:	aa	vaar	v aa r
e:	e:	ee	veer	v ee r
ø:	2:	eu	deur	d eu r
o:	o:	oo	door	d oo r
ɛi	Ei	ei	fijn	f ei n
œ	9y	ui	huis	h ui s
ʌu	Au	ou	goud	x ou t
ɛ:	E:	eh	crème	k r eh m
œ:	9:	euh	freule	f r euh l at
ɔ:	O:	oh	roze	r oh z at

The experiment was done by presenting soundless video of syllables built from the phoneme of interest to subjects with different levels of lip reading expertise. The confusion between the perceived and actually uttered phonemes taken over all subjects provided the eventual classification, which can be seen in Table 4.3 and Table 4.4. An explanation will be given below.

Table 4.3: Viseme classification: consonants

1	p, b, m	bilabial consonants
2	f, v, w	labiodental consonants
3	s, z, S	nonlabial front fricatives
4a	t, d, n, j, l	other nonlabial front consonants
4b	k, R, x, N, h	other nonlabial back consonants

For consonants, mistakes were almost exclusively made within the three sets of bilabial consonants (p, b m), labiodental consonants (f, v, w) and nonlabial consonants (t, d, s, z, n, l, j, k, r, x, N, h). These sets can be distinguished most evidently by visibility of lip articulation. Within the non-labial set, the separation of front-articulated consonants (t, d) and back-articulated consonants (R, h) can be explained by the actual place of articulation, and identification of the fricatives (s, z, S) can be based on the (limited) degree of opening of the oral cavity. Table 4.3 gives the final consonant classification, where subsets 4a and 4b are only recognized by the better phoneme identifiers. The number of consonants that can be distinguished for Dutch are only 4 compared to 6 to 7 in English.

Table 4.4: Viseme classification: vowels

1a	i, I, e:, E	close and half-close front vowels (unrounded)
1b	Ei, a:, A	half-open and open vowels (unrounded)
2	u, y, 9:, O	short back vowels (rounded)
3	2:, o:	long back vowels (rounded)
4	Au, 9y	closing and rounding diphthongs

Four sets of visually similar vowels can be recognized, namely unrounded vowels (I, i, e:, E, Ei, a:, A), short rounded vowels (u, y, Y, O), long rounded vowels (2:, o:) and closing and rounding diphthongs (Au, 9y). Mistakes also occur between sets though. Lip rounding appears to be the most important feature in distinguishing these sets. Within the rounded vowels, vowel duration plays an important role, and for the diphthongs (which end in a rounded position) both lip rounding and lip opening play a role. In the final vowel classification in table Table 4.4, subsets 1a and 1b were not observed, but their existence was suggested by another study.

Initially, only 8 classes are distinguished in this paper, but at the final conclusion two of these are split resulting in 4 visemes for consonants and 4 visemes for vowels. The better lip readers find one more consonant and one more vowel viseme. If a classification for which the between-class correlation is weaker is allowed, there can be 10 classes in total.

Possibilities for features we saw in this paper are lip articulation, place of articulation, degree of lip opening, lip rounding and vowel duration. Using these features, the classification seen in this section can be made.

The analysis of the viseme classification abilities of people with different levels of lip reading ability, shows that pure viseme classification is not influenced by lip reading expertise, which is in agreement with earlier experiments. The skilled everyday lip readers (hearing impaired persons who were considered to be experienced and skilled lip readers) performed slightly better at viseme recognition than the other groups however. So, if lip reading skills and viseme classification performance are independent, we should also evaluate the classification abilities of our features and the recognizer Hidden Markov Model (the “expertise”) separately.

4.1.3 Viseme set

The choice of which viseme set to use for a mapping from phonemes to visemes, is essential for the final performance of a visual speech recognizer. There cannot be too few or too many classes, because that will cause the viseme models to be badly trained due to poorly separated data classes. This is more important than the decrease in word distinguishability that the use of a small viseme set implies.

In the last section we have seen a human viseme classification. However, this classification is not sacred, since some of their conclusions were not even established by the authors. Besides, for our purpose, we are not just looking for a classification, but a set. Certain phonemes (such as “@”) are missing from the final classification.

By most linguistics 10 to 14 visemes are distinguished. The viseme set used by Wojdeł (see section 2.2) is the one given in Table 4.5, extended by adding classes for “h” and “Ei”, which were not included in the original set. This leads to a viseme set of 16 classes, which can be found in Table 4.6.

Table 4.5: Viseme set according to [24]

Viseme	Phoneme class	Viseme	Phoneme class
1	f v w	8	l e:
2	s z	9	E E:
3	S Z	10	A
4	p b m	11	@
5	g k x n N r j	12	i
6	t d	13	O Y y u 2: o: 9 9: O:
7	l	14	a:

Looking at the source of Table 4.5, we are not too confident about the viseme set established there. The researchers are computer scientists, not linguists, and they do not say what they based this viseme set on. Furthermore they missed two phonemes. However, which viseme set to use is never a clear issue. We decided to use the same set as Wojdel so as to ensure that our obtained could be comparable to his.

Table 4.6: Chosen viseme set: mapping onto 16 classes

	Viseme	Phoneme set SAMPA	Phoneme set HTK
1	at	@	at
2	ie	I	ie
3	a	A	a
4	aa	a:	aa
5	iee	I e:	i ee
6	eeh	E E:	e eh
7	oyu	O Y y u 2: o: 9 9: O:	o y yy u eu oo ui euh oh ou
8	ei	Ei	ei
9	fvw	f v w	f v w
10	sz	s z	s z
11	shzj	S Z	sh zj
12	pbm	p b m	p b m
13	gkx	G k x n N r j g	g k x n nn r j gg
14	td	t d	t d
15	h	h	h
16	l	l	l

Now that we have discussed visemes, it has to be noted that using them may not always be the logical choice. When the dictionary of the recognition task is small, and no words are ever expected to be added to the dictionary, one might decide to use word-level HMMs, which make both phoneme and viseme representations obsolete. This approach can be quite successful as we saw in section 2.5.4.

It could also be decided to not apply any mapping and train a lip reader on phonemes. It would be interesting to see the results of this, although we would not expect these to be very good. To compare viseme sets for lip reading applications, the surest way is to train a new recognizer for every candidate, and compare the results of these. Looking at the many possibilities this would however be very time consuming. A faster way might be to base the choice of which viseme set to use on

how well they can be classified using a chosen feature set. In section 7.3 we tried to do this, although not for the purpose of choosing the viseme set.

4.2 Performance boundaries

In this section we show the differences between acoustic and visual speech recognition and how they theoretically influence the performance. In section 4.2.1 we discuss the performance of visual features, in section 4.2.2, we discuss the performance of a recognizer based on HMMs and visemes.

4.2.1 Comparison between acoustic and visual speech

The main difference between acoustic and visual speech recognition is the type of input. As should be evident, the use of video imposes extra requirements on a visual speech recognizer that an acoustic speech recognizer does not have to deal with.

First of all, the information over the visual channel is not that rich compared to audio. When they speak, humans use sound as the primary means of communication, the movement of the lips can be seen as just a byproduct. Furthermore, a lot of it happens inside of the mouth and throat, and is not directly visible. Human lip readers can compensate for this because they actually understand language and know which context applies, but an automatic lip reader depends primarily on the features.

For audio, features can be extracted by splitting the sound into time frames, applying a Fourier transform, and using the Mel Frequency Cepstral Coefficients (MFCC). For video, first the face has to be located in the image, then features important to lip reading have to be extracted. However, there are a lot of different ways to accomplish this, some of which were discussed in section 1.1. It is not agreed upon what the best method is. We do know that any such approach should:

- Capture the most essential speech characteristics,
- Be fast, because video processing can take a lot of time,
- Preferably be person-independent.

4.2.2 Implications of using visemes

If a many-to-one mapping from phonemes to visemes is applied, it will be clear that this has consequences for recognition.

Because of the one-to-many mapping, we see that some dictionary entries end up with the same viseme (mostly consonants) repeated within a word (e.g. "ro**mp**", "ha**ngk**ast"). The question is whether it is alright for the representations to have double entries. They may be perceived as one, in which case, one of these double entries should be removed. How they are perceived could also depend on their syllabic structure. In Dutch, a syllable has the form C*VC*, where "C" denotes a consonant, "V" denotes a vowel, and "*" means there can be any number of them, including zero. It might be worthwhile to find out whether these double viseme entries are combined in case they appear within the same "C*" part of a syllable.

Dictionary analysis performed by Wojdeł on the Polyphone dictionary [3] shows that a mapping from phonemes to visemes can lead up to 10% less word separability. However, the viseme set used to compute this was not specified.

One last thing to consider is the propagation of misclassifications. If visemes are misclassified on the lowest level of the recognizer, the available word list N should be searched for the most probably candidate that matches the recognized phoneme/viseme sequence best. Only after candidate matching we obtain the resulting word that best matches the observation. When there are multiple candidates in the dictionary that share the same representation, correcting the misclassifications in this way can end up at some other word, yielding a false positive. Since this affects the number of insertion errors, this has a negative effect on the word recognition rate.

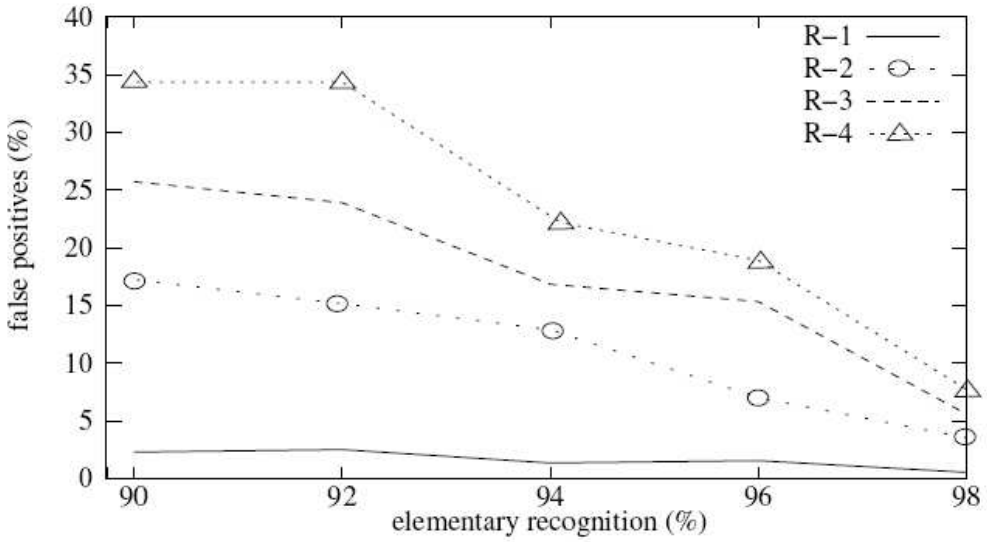


Figure 4.1: False positives as a function of misclassification of basic recognition entities [3]. R-1 is the full phonetic representation, R-2 is the viseme representation with syllable boundaries, vowel duration and stress point, R-3 is the viseme representation with syllable boundaries and vowel duration, R-4 is the viseme-only representation. For the Polyphone corpus, the percentage of words that are distinguishable assuming the most common option will be chosen is 99%, 93%, 92% and 91% for each representation respectively.

As shown by Figure 4.1 , even with a viseme recognition rate as high as 90%, already almost 35% of the words will be recognized incorrectly. For the phoneme representation where 99% of the words have a unique representation (note that for written language it would be 100%, the 1% error for phonemes represents the portion of words that sound the same but are spelled differently), the number of false positives would remain under 3%. And in a real-life system, the low-level misclassification rates will be much higher than 90%. That said, it might be overoptimistic to expect the same kind of performance from a viseme-based speech recognizer as from a phoneme-based recognizer.

4.3 Model

According to [3], building a typical automatic lip reader involves 4 steps. After discussing the remaining theoretical issues of visual speech recognition, the rest of the chapters will each explain one of the following steps:

1 Data aqcuisition (chapter 5)
2 Lip tracking (chapter 6)
3 Feature extraction (chapter 7)
4 Recognition (chapter 8)

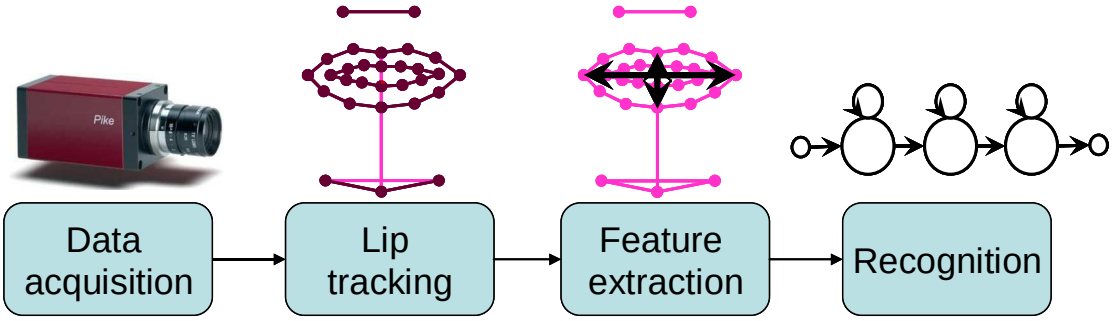


Figure 4.2: Visual speech recognition overview

These steps are also illustrated in Figure 4.2, and this picture will return in each of the chapters 5 to 8, to illustrate what the reader can expect to read about in those chapters. The pictures represent our design choices: a high-speed camera for recording, Active Appearance Models for lip tracking, point-based distances for feature extraction and Hidden Markov Models for the recognition framework. At the same time it models how a visual speech recognizer performs recognition.

The conclusion drawn from our research on existing speech corpora was that to really handle the task properly, a new data corpus was required. The chosen method for feature extraction is Active Appearance Models. This will result in a set of landmark points, which may not be directly usable as features for training. For training the recognizer, we will apply Hidden Markov Models based on visual phonemes (visemes).

As a consequence of using a phoneme to viseme mapping all dictionary entries have to be rewritten. For our lip reader we use the set of 16 visemes as seen in Table 4.6. Each viseme is modeled with Gaussian mixtures continuous density left to right Hidden Markov Models with five states, of which three are emitting. The same model is used successfully to model phonemes.

5 Data Acquisition

Training a speech recognizer requires large quantities of speech data. Data acquisition has been a big part of this project, because we the only training data we had at our disposal initially was the relatively small DUTAVSC speech corpus, and to train a lip reader successfully from scratch requires a large corpus. Furthermore we wanted to honor the conclusions from the research discussed in section 1.1.

This chapter starts by explaining the things that were considered before starting the recording sessions. First decisions had to be made about the speech that was going to be recorded (section 5.1), then we needed to think about the recording setup (section 5.2). After that we started recording many people, of which the results are given in section 5.3. The resulting data corpus was named New DUTAVSC. In section 5.4, we show the result of multiple recording sessions we had with a single person (Single Person New DUTAVSC) An analysis of the recorded data corpus can be found in section 5.5, and the result is discussed in section 5.6.

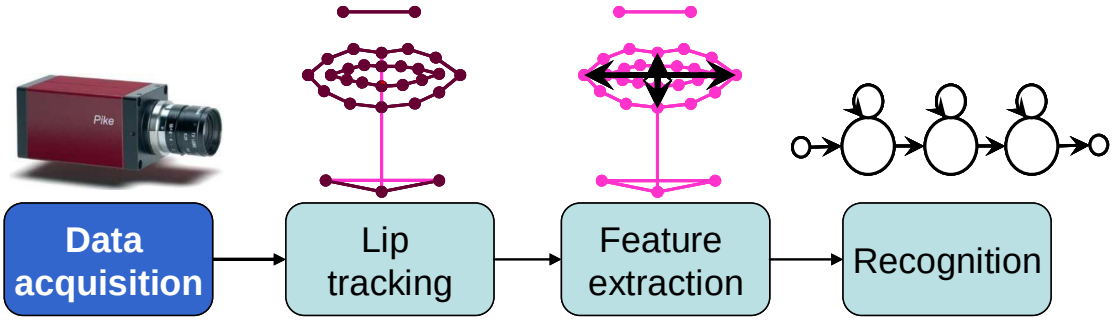


Figure 5.1: Visual speech recognition overview: data acquisition

5.1 Language coverage

The recording of New DUTAVSC had to be carefully prepared. Because of the scale of the new corpus we needed to make sure that all material would be gathered in a correct way. The new corpus would have to follow the quality requirements we had devised after the research done on other speech corpora; a high frame rate, capture of both frontal and profile view of the face and a rich utterance pool fit for continuous speech recognition experiments. Furthermore, we decided that it would be useful if people would be recorded at both normal and fast speech rate, and whispering (instead of low speech rate, as it appears to be more natural).

5.1.1 Utterance types

Here we will discuss the different types of utterances that were included in recording sessions and why. A large portion of it is based on the original DUTAVSC corpus. Digit sequences were included because it is a relatively easy task for a speech recognizer to accomplish and useful to see how well it performs at this basic task.

Spelling random words is different from reading a letter sequence, because firstly the length is unknown, making the task more difficult, and secondly the word itself is an existing one, which could provide some context information. Of course in practice mostly unfamiliar words like strange names would be spelled, in which case the context information would not be of much use.

Lists of random words were included to provide phoneme or viseme transitions that are rarely seen in actual sentences.

Bank application sentences were included to make the recognizer perform recognition tasks a little more difficult than digit sequences but still bound by a grammar. However, the original DUTAVSC grammar as can be seen in Figure 2.4 allows for grammatical errors confusing the speaker. Sentences have been corrected before they were presented to the speakers, but for future recordings an improved version of this grammar should be used to generate recording prompts. One such grammar is given in Figure 5.2. However, for recognition purposes the grammar should cover all recorded material, even if it does contain errors.

```
$number10 = twee | drie | vier | vijf | zes | zeven | acht | negen;
$number20 = tien | elf | twaalf | dertien | veertien | vijftien | zestien | zeventien |
achttien | negentien;
$number100 = [(1 | $number10) en] (twintig | dertig | veertig | vijftig | zestig |
zeventig | tachtig | negentig);
$number =
       [$number10] honderd [en] ($number100 | $number20 | $number10 | 1) |
       [$number10] honderd |
       $number100 |
       $number20 |
       $number10 ;
$digit = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9;
$amount = $number (euro | euro's) | een euro | 1 euro;
$greeting = goedemorgen | goedemiddag | goedenavond;
$please = alstublieft | alsjeblieft;
$want = wil | wilde | wou;
$type = [prive] [bank] rekening;
$account =
       [mijn] $type [nummer] $digit $digit $digit $digit $digit $digit $digit $digit |
       (mijn | m'n | een) $type;
$action =
       $amount van $account [naar $account] overmaken |
       $amount op $account storten |
       $amount storten op $account |
       $amount opnemen van $account |
       $amount van $account opnemen|
       een [nieuwe] $type openen |
       $account sluiten;
([$greeting] ik $want [graag] $action [$please] |
       [$greeting] ik ($want | zou) $action graag |
       [$greeting] ik zou graag $action [$please])
```

Figure 5.2: Adaptation of the grammar given in Figure 2.4 that will only allow for grammatically correct telebanking application sentences. HTK expects EBNF and requires the dollar sign $ to indicate non-final symbols

Generating prompts from a grammar formatted like this can be done using the following HTK commands:

```
> HParse bank_grammar.txt bank_wdnet.slf
> HSGen -l -n 50 bank_wdnet.slf dict.txt (> prompts.txt)
```

Where "bank_grammar.txt" contains the grammar, "bank_wdnet.slf" is a file that will contain the word network in standard lattice format, "50" is the number of prompts that will be generated and "dict.txt" contains a dictionary with a list of at least all the words that occur in the grammar. Viseme transcriptions are not yet required. ">

prompts.txt" can optionally be added to save the generated prompts to a file "prompts.txt".

Random sentences taken from Polyphone are especially useful for continuous speech recognition experiments. Because of the rich variety of sentence structures it is not really possible to constraint them by a grammar, unless it is by N-grams. N-grams are usually trained on large (textual) data corpora to get an accurate language model. It mainly counts how many times a given word is preceded by another word or combination of words. The resulting probabilities P (n| n-1, …, n-N) provide some context for a speech recognizer. Use of the Viterbi algorithm for alignment is dependent on the dynamic programming invariant, making it bigrams the only model we can use.

Common expressions like greetings and thanks were included because they would be useful for (dialogue) applications that require social interaction. Again it is hard to find a fitting grammar for this type of data, but many common expressions could almost be considered "words" (apart from the word combinations that already have become words, like "goedemiddag").

Finally, we included some open questions in hopes of recording some spontaneous speech. Most of them were formulated as questionnaire questions. The downside is that the answers had to be labeled by hand. Another practical disadcantage is that in our case the speakers had time to think before answering the question because they were recording themselves, thus eliminating part of their spontaneous reaction. To prevent one-word answers like just "ja" or "nee" next time, questions should be asked in such a way that people are provoked to give medium-long answers.

5.1.2 Speech types

We asked speakers to perform normal speech, fast speech and whispering. Recognizing speech spoken at the usual speed should not be a big problem for an average speech recognizer. Fast speech might be. Words are not articulated as clearly; subsequent words may be seemingly merged together, entire visemes or phonemes may be omitted. For automatic lip reading purposes, all previous video material was recorded at an insufficient rate, so recognizing fast speech was not even possible before.

Something sign language interpreters sometimes apply to facilitate lip reading, is *lip speaking*. This is done by emphasizing useful clues during speech. Obviously, normal people are not skilled at that, so to obtain a similar effect, we resorted to whispering. Whispering is something a lip reader might actually perform better at than an audio based recognizer. When people whisper, all sound they produce is unvoiced. Males and females even sound the same when they whisper. On the other hand, the mouth movements they make tend to be very articulated to compensate for the lack of sound.

Some side notes have to be made for recording this type of speech, though. First of all, people do not whisper a lot in daily life. Most subjects had to be reminded how to do it by the operator. Without instructions, some people were just speaking softly into the microphone with minimal mouth articulation, which wasn't what we had hoped for.

5.2 Recording setup

To record a large data corpus required the recording setup to be carefully thought through, because it was to remain in use over a longer period of time. For this corpus, we wanted the recordings to be dual-view and taken at high speed. First, we tried a setup with only one camera and a mirror placed at 45 degrees [25], but this proved troublesome because of distortions in the mirror image (it would appear further away than the original for instance) and it disallowed for high-resolution recordings. That is why we eventually went for a setup with two cameras, one in front and one at the side of the speaker. In the next sections we will give a description of that setup.

Figure 5.3: Photograph of the recording setup with recording subject and operator

5.2.1 Environment

The environmental conditions of the recordings determine the illumination and background of the scene. We used (blue) monochrome background panels so that speakers could be given custom backgrounds using "chroma keying" also used for weather forecasts on television.

To create the right environmental conditions we needed to have complete control over the room's lighting and noise level. Although the MMI department has a sound studio, the lighting conditions there could not be regulated making it infeasible to set up our lab there. The room we used to record was not isolated and we had no control over the light coming front the window and air-conditioning. That is why we first had to darken the room by covering the windows with packing foam panels and garbage bags, and introduce our own controllable light sources. We put 2 x 2 500 W

construction site lamps on stands and placed them at both sites of the speaker such that the scene would have uniform lighting (see Figure 5.5). Furthermore, the air-conditioning slits under the window were blocked with packing foam.

5.2.2 Equipment

We used two high-speed cameras and two directional microphones. We decided to go for 100 Hz video recording at half PAL resolution. The audio signal was sampled at 48 kHz on 16 bits. Just a computer with great RAM capacity was not acceptable for our experiments due to lack of speed. Instead the computer was one with a stripe driver setup (4 hard drives packed together in a RAID 0 configuration) so the uncompressed raw data of multiple recordings could be temporarily stored on it.

Figure 5.4: AVT Pike F-032C high-speed camera

We used two Pike F-032C cameras built by AVT (see Figure 5.4). The cameras are capable of recording at 200 Hz in black and white, 139 Hz when using the chroma sub-sampling ratio 4:1:1 and 105 Hz when using the chroma sub-sampling ratio 4:2:2 while capturing at the maximum resolution of 640 x 480. By setting a smaller Region Of Interest (ROI) the frame rate can be increased. In order to increase the Field Of View (FOV), we recorded in full VGA resolution at 100 Hz. We used the fire wire card bus' clock for synchronization into a 125 µs range.

For recording the audio signal we used NT2-A Studio Condensators. We recorded a stereo signal using a sample rate of 48 kHz and a sample size of 16 bits. The data was stored in PCM audio format. After the equipment was ordered, it took some time to arrive. Accomplishing the synchronization of the two cameras and linking it to the recording software also took time.

5.2.3 Laboratory setup

Two cameras were placed at eye height (120 cm from the floor) at a distance of 162 cm from the subject, one directly in front and one directly from the side. The subject was seated in a chair that couldn't turn or be adjusted in height, in a tradeoff between comfort and keeping subjects from moving. In cases where subjects were too short to be in view of the cameras (mostly females), we improvised by having them sit on packs of paper since adjusting the camera for each new subject would be time consuming and prone to errors.

We aimed to capture the lower half of the face, because we were recording at a high resolution and to lip reading the mouth and chin area is more relevant than the eyes. We also wanted to guarantee some anonymity of the speaker this way, but in practice the busy schedule didn't allow for camera adjustments between recording sessions, and there was a risk that the nostrils would fall out of the frame, which we need for computing later on. That is why we ended up capturing roughly the lower half of the face; sometimes the eyes were in the picture as well.

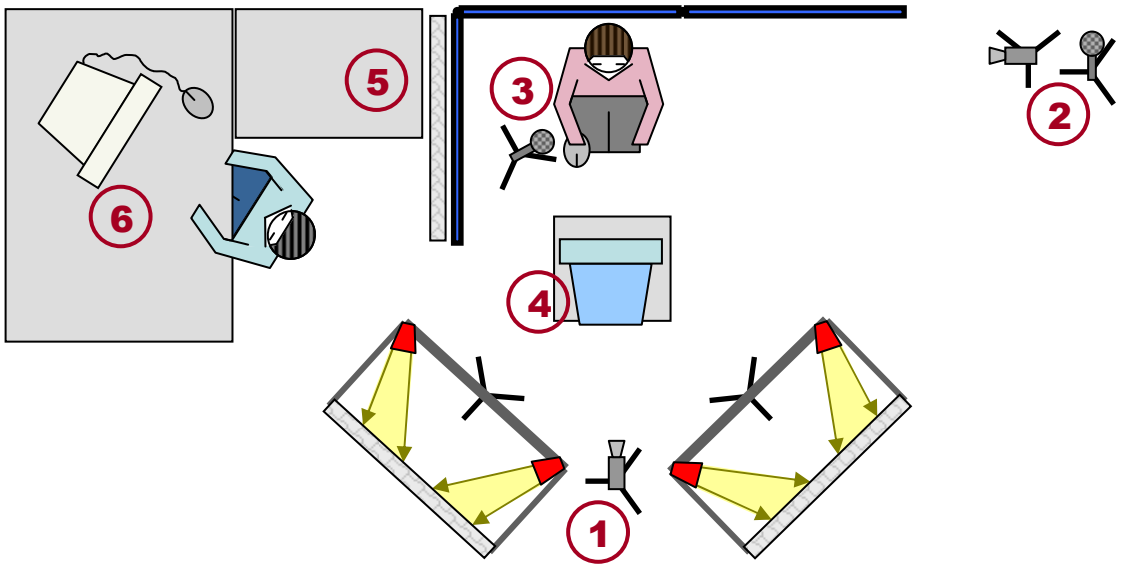


Figure 5.5: Map of the recording setup: 1) frontal camera (height 1.20 m); 2) side camera (height 1.20 m) and environment microphone (height 1.50 m); 3) recording subject and speech microphone (height 80 cm); 4) monitor showing prompts (36 x 26 cm); 5) computer running operating and recording software and microphone controller; 6) operator checking speaker performance on second monitor.

A monitor for the subject to read off was placed in front of the subject out of the line of sight of the cameras. One microphone was placed close to the subject; the other was placed some distance away to record the background noise. Figure 5.5 shows the precise setup of the laboratory and Figure 5.3 shows a photograph.

5.2.4 Operator

The operator was sitting behind the background panel of the side view camera, and was looking at the same prompts as the subject on a second monitor, allowing them to have the subject retake items in case of error or disturbing background noise (usually from outside the room). The operator also explained the courses of action to the subject and had them sign a consent document in advance. In appendices A and B respectively we present the written instructions and consent document as presented to the subjects.

5.2.5 Operating software

The tool we used to prompt the subjects for utterances was also used to control the video and audio devises. The subject was given a wireless mouse to operate the software with. The left mouse button would start and stop a recording; the right mouse button would take them to the next item. The screen would display the item to be uttered together with some instructions about the speaking style, e.g. normal speech rate, fast speech rate, or whispering. To make reading easier on the speaker, prompts were presented in black sans-serif letters on a white background (see Figure 5.6). Each recording session was preceded by a short trial to familiarize the subject with the software controls.

Figure 5.6: Screenshot of the prompter/recording software

Using a high-speed camera increases the storage needs for the recordings. It is almost impossible to record everything and cut the clips afterwards at the required lengths, mainly because the (large amount of) video data is temporarily captured in the RAM of the computer and needs to be written to the hard disk at set times. Giving the speaker control over the recordings allowed all video and audio for every utterance to be neatly synchronized and saved in a new folder, with the labels already known because they were used to prompt the speaker (although the speaker could have made mistakes). The tool was also used to keep track of the user's data, recording takes and recording sessions.

While it was convenient to us that the subject was taking their own recordings, we also had to carefully instruct them not to start a recording too late or end it too early, which people would inevitably start to do especially when they were asked for fast speech rate. We also instructed them to close their mouths between utterances, but of course this still wasn't something we had complete control over.

5.3 Recording of New DUTAVSC

In this section we will discuss the results of the recording of the New DUTAVSC corpus, using the setup previously discussed. In section 5.3.1, we will describe the planning of a recording session, of which we gathered one or two per recording subject. The variety of people that participated will be illustrated in section 5.3.2.

5.3.1 Recording session composition

When compiling an entirely new data corpus, there is a lot of freedom to decide what exactly to record. However, good language coverage is required. We decided to present the user with random sentences drawn from a pool and random digit and letter sequences. We wanted a recording session to last about 20 minutes, during which the items given in Table 5.1 needed to be recorded.

Table 5.1: Original build-up of a recording session for New DUTAVSC

Number	Speech type	Utterance type
3	normal	Random digit sequences of length 8
3	fast rate	"
3	whispering	"
3	normal	Spelling a random word of variable length
3	whispering	"
3	normal	Lists of random words of length 8
3	fast rate	"
3	whispering	"
5	normal	Fixed grammar bank application sentences
5	fast rate	"
5	whispering	"
5	normal	Random sentences taken from Polyphone
5	fast rate	"
5	whispering	"
5	normal	Every day use common expressions
5	normal	Short answers to random open questions

For this data corpus, we recorded a large variety of people. The original DUTAVSC corpus consisted of only 8 people. For this our new data corpus, we recruited as many people as we could find (though most were from inside the university). We set a goal of around 50 people, but in the end, we recorded a total of 70 different people, for some of them multiple sessions were recorded.

Seven of the subjects recorded multiple sessions. This was very welcome because one recording session was only good for about 10 minutes of material. While the data is rich because of the frame rate, the number of utterances per session was only 64. We recorded 79 sessions this way.

This results in a total of 64 utterances per recording session. One session lead to approximately 10 minutes of recorded material. Most categories also appeared in the original DUTAVSC data corpus. With all of this in mind, we made recordings of 70 people. A total of 79 sessions were recorded

5.3.2 Demography

We asked all participants of recording sessions for some basic information on gender, age, level of education, occupation, whether they were native Dutch speakers, and which province they originated from to find out about possible dialects (see appendix B).

A data corpus should consist of both male and female speech material, ideally with a ratio of 50%-50%. For some reason, the faculty of EEMCS at this university is low on female students and staff. To approach the ideal ratio, we had to recruit at the administration department, where, funny enough, most employees are female.

Our 70 recording subjects were all adults aged 19 to 64. Most subjects were students in the age range from 19 to 28, mostly male, originating from around the country. The next group was that of staff members in the age range from 25 to 50, most of them female. Their level of education varies, and most of them are originally from the province of Zuid-Holland (where Delft is also located). The professors and PhD students we recorded were from all over the country also, only one of them female, and obviously all of them with a Master's degree.

Of the 70 people recorded, 2 were not native Dutch speakers. Also, because the question about the province of origin could be interpreted as "where do you currently live", we fear our information is not always complete. In total there were 49 males and 21 females recorded. 41 students, 8 PhD students, 6 professors and 15 staff members. This is illustrated in Figure 5.7.

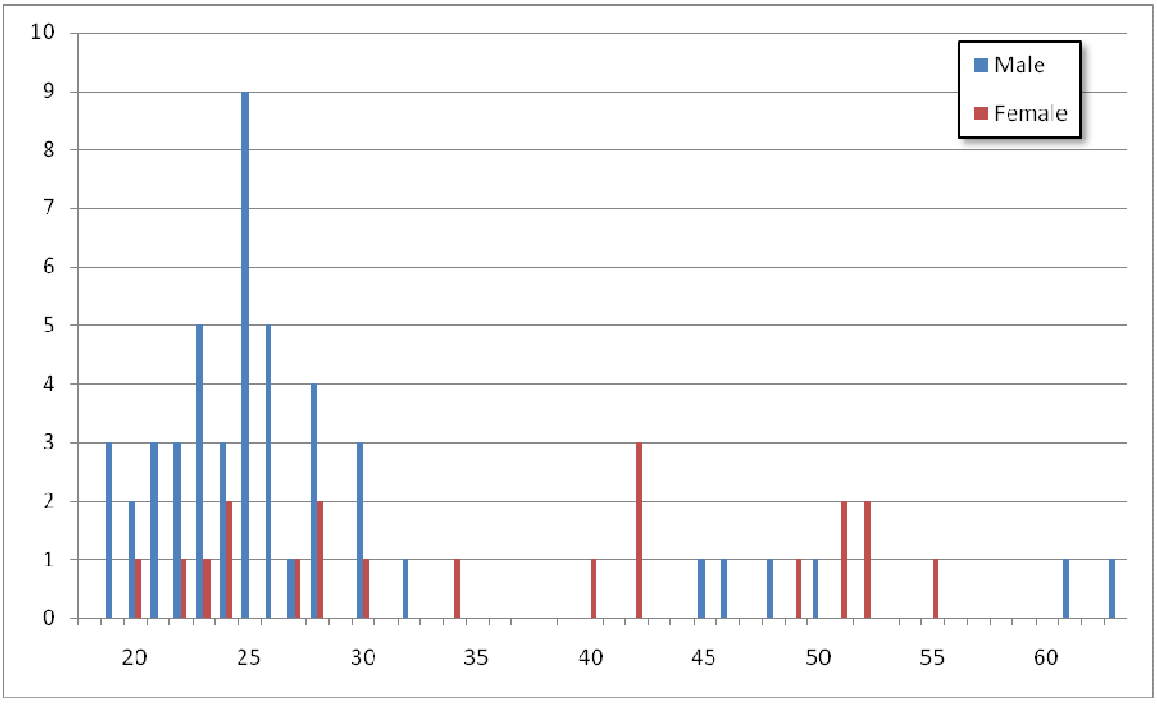


Figure 5.7: Age distribution and gender of all 70 recording subjects

5.4 Recording of Single Person New DUTAVSC

In this section we will give the results of the recording of Single Person New DUTAVSC, which was decided upon when our feature set turned out not to be person-independent.

We asked one person (female, 52 years of age, secretary) to help us gather more data so we could train a speaker dependent recognizer. This person recorded 1 session of 64, 5 sessions of 125 utterances and 5 sessions of 155 utterances. For

convenience, we will denote this as a separate one-person data corpus, and call it Single Person New DUTAVSC for the remainder of this thesis.

64 utterances for one person is not a lot of material to train on. We decided to record 10 more sessions composed a little differently for one willing subject. We came up with the session planning given in Table 5.2.

Table 5.2: Original build-up of a recording session for Single Person New DUTAVSC

Number	Speech type	Utterance type
45	normal	Sentences taken from Polyphone
10	"	Random digit strings of length 8
10	"	Random letter strings of length 8
30	"	Isolated digits
30	"	Isolated letters

This schedule accounted for a total of 125 utterances per session. For the second day of recordings and last 5 sessions we changed the numbers around to match the differences in dictionary size (10 for digits, 26 for letters), for a total of 155 utterances per session, as illustrated in table Table 5.3.

Table 5.3: Final build-up of a recording session for Single Person New DUTAVSC

Number	Speech type	Utterance type
45	normal	Sentences taken from Polyphone
10	"	Random digit strings of length 8
30	"	Random letter strings of length 8
20	"	Isolated digits
50	"	Isolated letters

The idea behind all of this was that to train for simple tasks like digit and letter recognition we wanted to make sure we gathered enough data. We only wanted to record normal speech rate this time, because for the previous recordings, roughly 2/3 of the data was at either fast speech rate or whispered, so deciding not to use them for training would not be an option, while for some applications this might be the logical choice. For all of this goes "less is more", less categories for the same data means more data per category. Bank application sentences, word lists, common expressions and open questions were left out due to their not so directly obvious use.

A problem we had seen before concerning the randomization of the prompts was that it happened quite a lot that the same prompt would come up twice in the same session, even twice in a row. This time, we wanted every single Polyphone sentence to be recorded, so we just iterated through them all alphabetically.

The motivation to include isolated digits and letters now, is that the model would not have to deal so much with co-articulation effects this way, making it easier to train. We instructed the subject more explicitly to start at a neutral (closed) mouth position and to take time before and after speaking, leading to nicer recordings from a lip reading perspective.

In the previous spelling assignments, for words that had the letter "IJ" in them it had been split up into "I" and "J", causing IJ as letter to never appear. This time we included IJ as letter instead of Y (which is "foreign" in Dutch anyway, people tend to call it either "Griekse IJ" or "Y-grèc", which is French).

Because of the many short recordings, a recording session took just a little longer than originally. Our subject insisted that doing five sessions in a row with just one short break was perfectly doable.

5.5 Processing the recordings

Before using data to train a speech recognizer, it is important to validate it. While Hidden Markov Model based speech recognition can deal with a certain amount of noise, it still has to be possible to extract something useful from it. Errors could have been introduced in several ways we all had to check.

Firstly, the speaker could have made mistakes. Both speaking errors and errors handling the recording controls (mouse buttons) occur frequently. This we checked by auditory validation described in the next section. Secondly, the hardware was responsible for quite a lot of mistakes as well. However, this is probably unavoidable while recording at this frame rate. Lastly we performed some visual validation on the data.

5.5.1 Auditory validation

The first thing we checked about the recordings was the audio, because it was the easiest and quickest to access. Often, the operator had already asked the speaker to retake something during the recording when a mistake was made, but there were always things slipping through. The main purpose was to make sure the transcriptions for the recordings were correct, as they did not always match the prompts.

We checked for the following anomalies:
- Words skipped by the speaker (deletion) ,
- Words inserted by the speaker (insertion),
- Words distorted by the speaker,
- Words missing due to premature stop of the recording,
- Words incomplete due to premature stop of the recording,
- Recordings for which the speech style is different from the instruction,
- Background noise.

Insertion and deletion mainly occurred because the speakers were not always familiar with - or used to - the expressions appearing on the prompter. Sometimes the prompts contained grammatical or spelling mistakes. Word distortions were mainly expected for recordings at fast speech rate, but also occurred naturally depending on the speaker. Bad timing sometimes caused the speaker to cut off words at the beginning or ending of a sentence.

Also, even though the speaker could retake something, the original take was not overwritten and still needed to be checked. Sometimes, noise was introduced by the environment. For the open questions, the answers had to be transcribed.

All the audio clips were lined up in a play list and observed. The results of the auditory validation were files for every recording session organized as seen in Table 5.4.

Table 5.4: Example of a verification report for a recording session

#	Utt #	Take #	Quality	Environment	Actual Label
1	1	1	g		
2	2	1	g		
3	3	1	g		
4	4	1	g		
5	5	1	g		
6	6	1	g		
7	7	1	g		
8	8	1	g		
9	9	1	g		
10	10	1	g	operator	<n> e...<o> <e> <m> <e> <n>
11	10	2	g		<n> <nul>
12	10	3	g		<n> <nul> <e>
13	10	4	g		
14	11	1	g		
15	12	1	g		
16	13	1	g		incomplete
17	13	2	g		
18	14	1	g		
19	15	1	g		
20	16	1	g		ja, één

5.5.2 Visual validation

Visual errors like the subject moving out of the frame were checked in the point tracking stage. At the time of the recordings, the operator could only see the first and last frame of a recording, giving some indication about the posture of the speaker, but not all the time. Many speakers tended to bend forward a bit as they sat down for some duration of time, forcing the operator to correct them. We do not exactly know for how many recordings this is the case, but sometimes the camera or chair was not properly adjusted to the speaker's height, causing the chin to go out of the picture while talking. Figure 5.8 shows a dual view example frame of New DUTAVSC.

5.5.3 Hardware issues

Some important errors were caused by the hardware. The audio signal was okay most of the time, although a little unclear during whispering. The recording of the video put far more strain on the systems, however, and this led to errors.

Figure 5.8: Sample front and profile view frames of New DUTAVSC

A lot of the data proved to be affected by a lagging hard disk and/or camera. Apparently, 100 Hz is hard to deal with for the devices we used. And of course there were two cameras recording at 100 Hz each at the same time. The result is data with temporal gaps in them: if frames were missing, a whole series of them would be missing.

The number of affected recordings is large, we estimate that one quarter of the complete data set is affected. However, for a large part the number of skipped frames is not that high. In spite of these errors we decided to use recordings for which the number of skipped frames was under 10 (0.10 seconds). Because recordings were made at the relatively high frame rate of 100 Hz, missing some frames in practical applications is not unthinkable, so we decided that under 10 missing frames with a recording of perhaps 2 seconds (200 frames) is acceptable. We estimate that about one third of all recordings with missing frames and thus one ninth of all data could not be used. For the single person recordings more accurate statistics are given in Table 5.5. 27% of all recordings had at least some missing frames, but in about 40% of the cases the number of missing frames is under 10. Session 0 (part of the original New DUTAVSC) was a very bad recording session in this sense.

5.6 Conclusion

After the recording of this data corpus was complete, we were able to concentrate on building the automatic lip reader. This would take the new speech corpus to the test. With respect to this data corpus, the main question we ended up with was, whether we had gathered enough data to train an automatic lip reader from scratch. And if so, whether this lip reader would be either person-dependent, or person-independent, like we had hoped.

Table 2.1 can now be extended with the information of our new corpora, given in Table 5.6.

Table 5.5: Percentage of missing frames, and percentage of that for which the number of missing frames is under 10 for frontal and profile frames. Single Person New DUTAVSC sessions were used.

Session	missing frames	< 10 (front)	< 10 (side)
0	38.1%	8.3%	12.5%
1	12.0%	33.3%	33.3%
2	18.4%	52.2%	47.8%
3	20.8%	26.9%	23.1%
4	22.4%	46.4%	46.4%
5	31.2%	53.8%	56.4%
6	37.4%	43.1%	43.1%
7	29.7%	45.7%	45.7%
8	34.2%	52.8%	52.8%
9	29.0%	46.7%	44.4%
10	25.8%	30.0%	30.0%
Mean	***27.1%***	***39.9%***	***39.6%***

Table 5.6: Extension of Table 2.1 showing the data for New DUTAVSC

Corpus	Language	Sessions	Number of speakers	Audio Quality	Video Quality	Language Quality	Stated purpose
New DUTAVSC	Dutch	79	70: 49 male, 21 female	48 kHz on 16 bits	384x288, 8bit, 100 fps, lower half of the face	Connected digits, spelling, application and natural sentences, normal, fast and whispered, common expressions and open questions	Multi-purpose: word-level, sentence-level and continuous speech recognition, restricted or unrestricted by grammar
Single-person New DITAVSC	Dutch	11	1 (female)	48 kHz on 16 bits	384x288, 8bit, 100 fps, lower half of the face	Single and connected digits and letters, natural sentences	Small vocabulary isolated/connected words recognition

6 Lip tracking

The first step we took to accomplish data parameterization on the recorded video material was that of finding the face and mouth. While there also exist other approaches, as seen in section 1.1, ours was to track certain landmark points. For this we used Active Appearance Models (AAM), which will first be discussed in section 6.1, and the AAM Annotation Lab, discussed in section 6.2.

The application to lip reading starts by defining the lip model (section 6.3), and training it (section 6.4). In section 6.5 we finally evaluate the results of point tracking using Active Appearance Models. The resulting face points will be used for feature extraction in the next chapter.

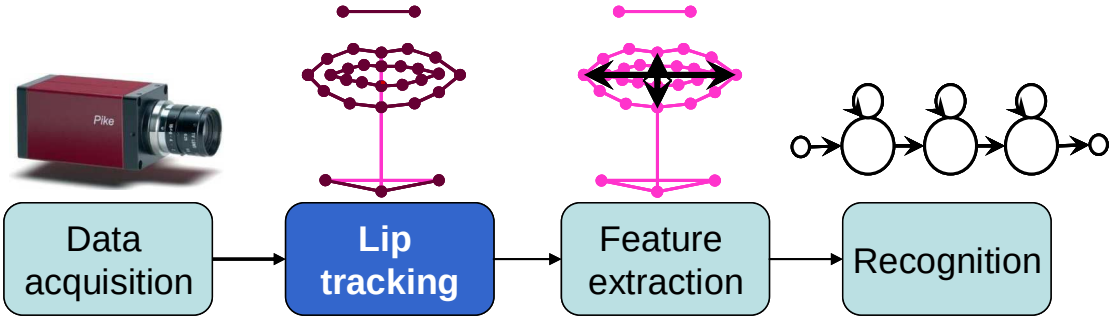


Figure 6.1: Visual speech recognition overview: lip tracking

6.1 Active Appearance Models

Active Appearance Models (AAM, introduced in section 2.5.5) are a convenient tool for certain computer vision tasks. Using a model of the shape and appearance of an object, similar objects can be found in images. How the model is allowed to transform depends on the set of annotated images it was trained on. Active Appearance Models were first introduced in a paper by Cootes et al. [26].

Approach

An Active Appearance Model combines statistical shape and grey-level appearances of certain objects to be identified in an image. The application of Active Appearance Models comprises of two steps. The first step involves an offline training phase that will estimate the model parameters. The second step involves searching new images for a fit of the model based on an initial estimate.

To train an AAM, the training supervisor first needs to present a training set labeled with landmark points that appear in all images. These are usually generated manually and takes a lot of time. A "bootstrap" approach to training can be to use the current model to help label new images, which are then added to the model. Incrementally building a model in this way is repeated until the AAM finds the points of new examples sufficiently accurately every time, which means it requires no more training.

AAM search

After an AAM has been trained for an object, it can be used to search new images for that object. The search starts from the mean model and iteratively modifies the

model parameters inside the learning range while minimizing the difference in appearance between the real image and the image synthesized based on the new model. The required number of parameters is computed in both cases by using Principal Component Analysis (PCA). In order to match to an image, we measure the current residuals and use the model to predict changes to the current parameters, resulting in a better fit. A good overall match is obtained in a few iterations, even from poor starting estimates.

The use of a face/mouth detection/tracking algorithm (Viola/Jones face detection) as an initial guess was found to greatly accelerate the search for the shape parameters during AAM based processing. This enhancement enabled a real-time implementation of the algorithm. The face is located in an Active Shape Model search, and the shape parameters are extracted. The face patch is then deformed to the average shape, and the grey-level parameters are extracted. The shape and grey-level parameters are used together for classification. Active Appearance Models combine both shape and texture parameters into one compact model.

Given a new image, the aim is to identify the object in a way that is invariant to confounding factors such as lighting, pose and expression. This is done using the Mahalanobis distance measure, which enhances the effect of inter-class variation, whilst suppressing the effect of between-class variation.

6.2 AAM Annotation Lab

The AAM Annotation Lab is the software tool we used to annotate video and perform point extraction, all using Active Appearance Models. It was developed by Alin Chiţu, who in turn used an external implementation by Cootes at al. [20]. A screenshot is displayed in Figure 6.2.

6.2.1 Functional Description

The software can be used both to train AAMs and apply AAM search. Frames can be annotated by hand (mark certain points on the face) and those annotated images used as training examples when training an Active Appearance Model. A saved model can then be loaded into the program and used to find the face points for new images.

Training a new model is done by loading a new image and adding model points organized in paths. In the case that the object is a face, different paths could be added around the contours of the face, eyes, nose and mouth for example. An addition by Alin Chiţu is the ability to put constraints on the placement of these points, as to ensure a more uniform annotation. Edges between points are forced to be parallel or perpendicular to each other. The AAM algorithm itself is not affected by these constraints. There are more options available to make annotating easier, for example the mean shape or previous annotation can be copied to the next image, and points that were found by a (partially) trained model can be moved around manually.

Training an AAM with this program is done by providing a folder with annotated training examples. The training time is dependent on the number of training samples provided. The model parameters are trained until at least 95% of the variance can be explained. The training time and size of the resulting model (stored in binary format) depend on the quantity and diversity of the training set.

Using this tool for point tracking is done by first loading a suitable model and image sequence, and perform a search. The AAM algorithm comes into play when points are tracked. For the first frame an initial guess is made by inserting the mean shape. Providing the initial guess can also be done manually by providing the annotation. Although this is not the procedure we want to follow - we want point extraction to be fully automatic - it does guarantee reliable results.

In the current version of the program, all frames for an (uninterrupted) recording are opened simultaneously before tracking is performed. During the search, the frames pop-up after each like a video sequence, allowing immediate visual validation. Most recordings take just a few seconds to be processed entirely.

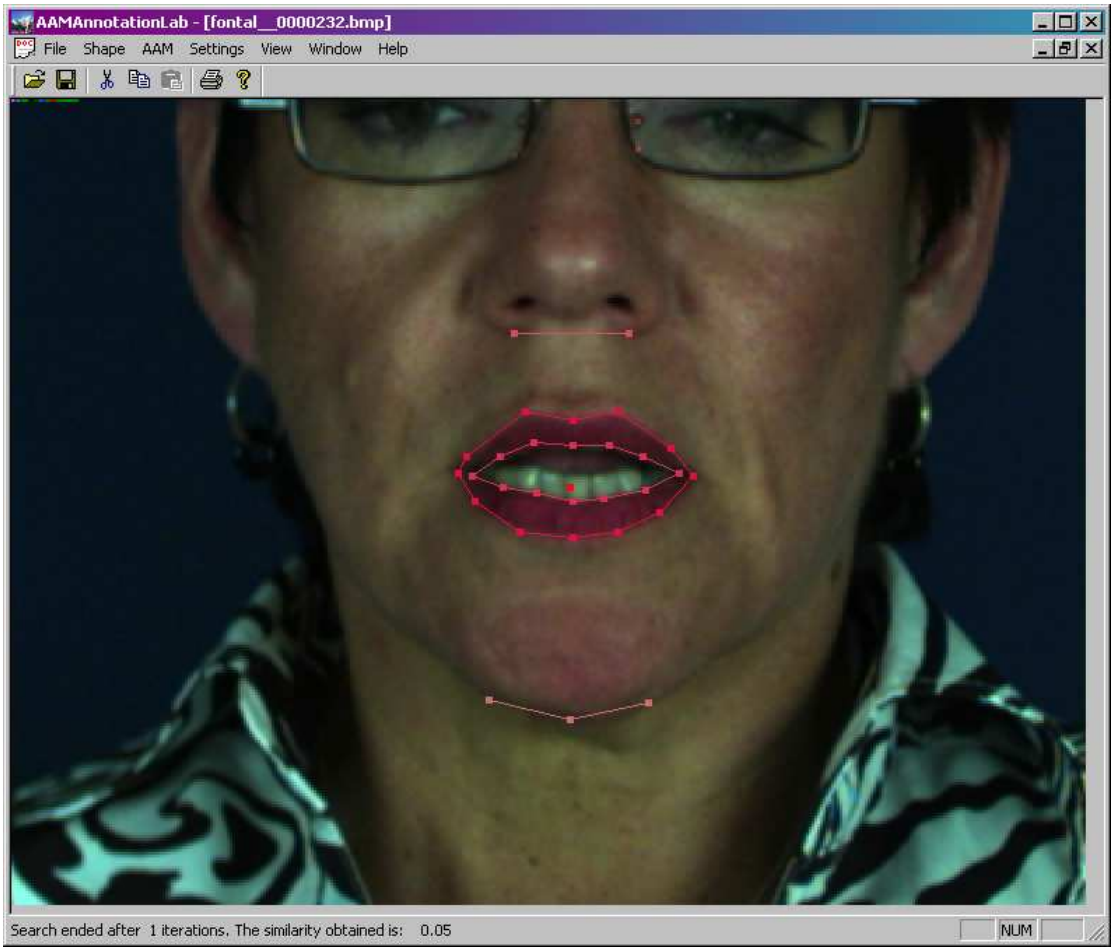


Figure 6.2: Screenshot of AAM Annotation Lab, with detected model points indicated on the loaded frame

6.2.2 File format

After an AAM is trained, it is stored in a binary format with the extension AMF.

The image annotations returned by the software are poured in a set of point coordinates in ASCII format. First the number of points is specified, then the relative coordinates of those points. Normalization of x and y coordinates is done according to the image resolution (384 x 288 in our case). The relations between the points are also given, showing which points form a path together. It also gives the name of the

image counterpart. Both annotating by hand and applying an AAM result in such a file. An example .ASF file is given in Figure 6.3.

```
######################################################################
#
#    AAM Shape File  -  written: Monday December 15 - 2008 [15:26]
#
######################################################################

#
# number of model points
#
25

#
# model points
#
# format: <path#> <type> <x rel.> <y rel.> <point#> <connects from> <connects to> <user1> <user2> <user3>
#
0       1       0.42607439      0.45010457      0       7       1       0.00    0.00    0.00
0       1       0.49949555      0.38849644      1       0       2       0.00    0.00    0.00
0       1       0.54768693      0.39242752      2       1       3       0.00    0.00    0.00
0       1       0.59027567      0.37919278      3       2       4       0.00    0.00    0.00
0       1       0.66781477      0.42058484      4       3       5       0.00    0.00    0.00
0       1       0.58141502      0.48384775      5       4       6       0.00    0.00    0.00
0       1       0.54841500      0.48982627      6       5       7       0.00    0.00    0.00
0       1       0.48850343      0.48843980      7       6       0       0.00    0.00    0.00
1       1       0.45399342      0.44162939      8       15      9       0.00    0.00    0.00
1       1       0.49576520      0.39856323      9       8       10      0.00    0.00    0.00
1       1       0.54763896      0.40857251      10      9       11      0.00    0.00    0.00
1       1       0.59916921      0.39788249      11      10      12      0.00    0.00    0.00
1       1       0.63395605      0.41811975      12      11      13      0.00    0.00    0.00
1       1       0.59354006      0.42725822      13      12      14      0.00    0.00    0.00
1       1       0.54982364      0.43767984      14      13      15      0.00    0.00    0.00
1       1       0.49112842      0.43150759      15      14      8       0.00    0.00    0.00
2       5       0.46685836      0.69888635      16      18      17      0.00    0.00    0.00
2       5       0.55169464      0.71679567      17      16      18      0.00    0.00    0.00
2       5       0.60614983      0.69688417      18      17      16      0.00    0.00    0.00
3       5       0.47582450      0.26400430      19      24      20      0.00    0.00    0.00
3       5       0.51690497      0.27177300      20      19      21      0.00    0.00    0.00
3       5       0.50063739      0.29660528      21      20      22      0.00    0.00    0.00
3       5       0.57881567      0.29554871      22      21      23      0.00    0.00    0.00
3       5       0.56178800      0.26712764      23      22      24      0.00    0.00    0.00
3       5       0.59318058      0.24862790      24      23      19      0.00    0.00    0.00

#
# host image
#
xfontal__0000020.bmp
```

Figure 6.3: Example of Active Appearance Model shape file containing model point co-ordinates

6.3 Defining the lip model

With Active Appearance Models it is possible to track any kinds of objects in images. In the original paper [26] they are applied to perform face tracking for identification purposes, and used on medical images. For the purpose of lip reading, we will use them to track the lower part of the face. More points than necessary would lead to more work for the annotators, while we still needed enough points to accommodate for the requirements of our features. The model should contain enough points to cope with all shape variations that occur on the object (e.g. lips). That is why our considerations for the chosen model points were based on the nature of the features we hoped to extract.

For every frame point its coordinates would be calculated. A logical choice would be to take certain distances and areas as features. The lips seem to provide the most visual cues in speech production, so first we included certain points on and around the lips. We wanted to add the area of the lips and mouth opening as features, so there would have to be enough points to approximate the shape of the lips closely. Lips are affected by many muscles and can thus take a great variety of different shapes (see Figure 6.4).

Having a reference point to enable us to determine the scaling and orientation of the face is useful. This made us decide to include fixed points around the nose.

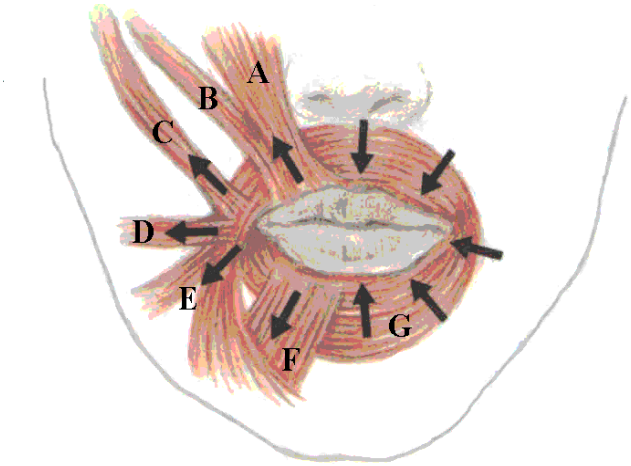


Figure 6.4: Facial muscles around the mouth and directions of muscle contraction. A. levator labii superioris. B) m. zygomaticus minor. C) m. zygomaticus major. D) m. risorius. E) m. depressor anguli oris. F) m. labii inferioris. G) m. orbicularis oris [27]

In the work of Jacek Wojdeł, we have seen that it had paid off to determine whether the tongue and teeth were visible using a color filter. While we would just be tracking the points instead of using color information, we figured that the visibility of the teeth would also be apparent from the distance from the nose to the chin, since the teeth are attached to the jaws. So, we decided to include some chin points in the model.

6.3.1 Terminology

Before defining the model, we had to be clear about some anatomical descriptions. For the annotators to come up with a uniform annotation of the images, the model points had to be carefully defined. Each point was given a definition to be used by annotators to manually produce (or correct) key point positions using the software.

First, we need to explain some anatomical terminology in the lip area. Figure 6.5 shows a nose and mouth seen from a low angle, with the lips and nostrils fully visible. The central vermillion tubercle is the "lump" most people have in the center of their upper lip. It often sticks out a bit and can be easily recognized. The philtrum is the narrow area between the nose and lips between two "lines". The nasal columnella is what separates the nostrils.

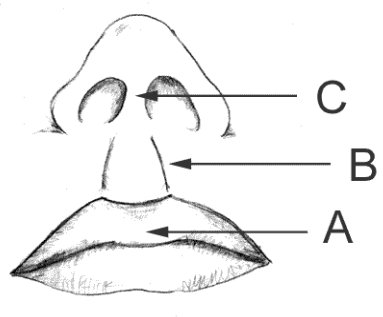


Figure 6.5: Lip anatomy. A) Central vermillion tubercle. B) Philtral column. C) Nasal columnella [28]

6.3.2 Original 25-point lip model

The first model we used for annotation was mainly based on intuition. Although most of the points could be placed unmistakably according to their definitions, not all of them could. The initial model had 24 points: 6 around the nose, 3 around the chin and 16 in the mouth area, following the point definitions in table Table 6.1.

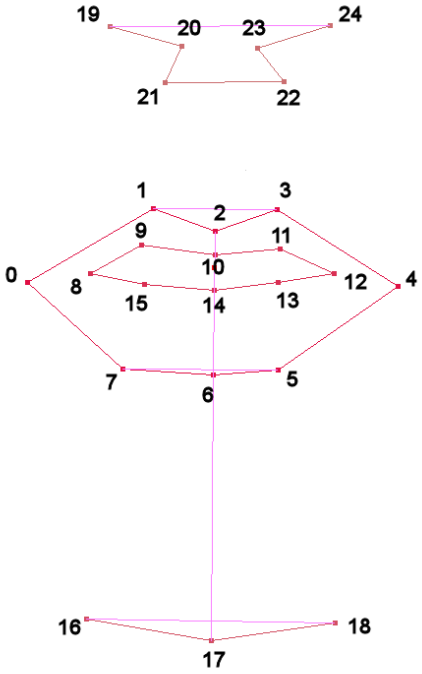


Figure 6.6: Initial model – points are numbered, red lines indicate polylines, purple lines indicate constraints

There were however some limitations to this model. It was not able to cope with all mouth shapes we could present, e.g. a widely opened mouth. Furthermore not all points had an unambiguous definition. Some definitions were just plain unclear or only valid when the mouth was open. Because we wanted to include area features, the interior shape of the mouth should cover the area of the mouth more exactly. A more rigorous definition was needed.

Table 6.1: Initial model point definitions

0, 4	Points at the corners of the lips
8, 12	Intersection points of upper and lower lip when mouth is opened
17	Lowest point at the center of the chin
16, 18	Points on the jaw flanking the chin (no exact definition)
0-7	Polygon describing the outer shape of the lips as well as possible
8-15	Polygon describing the shape of the mouth opening as well as possible
1, 3	Points where the "lines" of the philtrum meet the upper lip
2	Lowest point where the upper lip curls inward
10	Lowest point of central vermillion tubercle
6, 14	Low and high points on the lower lip on the imaginary line from 2 to 17
19	Upper left point of left nostril (from camera's point of view)
20	Lower right point of left nostril
23	Lower left point of right nostril
24	Upper right point of right nostril
21, 22	Base of the central columnella/starting point of nose wings

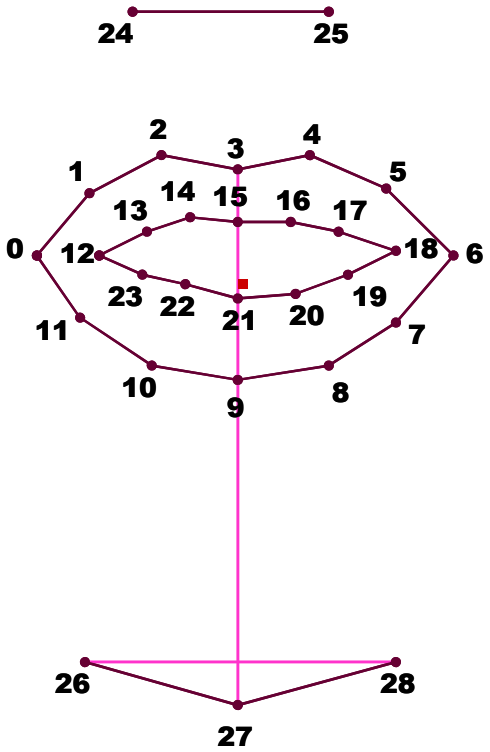


Figure 6.7: Final model – points are numbered, red lines indicate polylines, purple lines indicate constraints

6.3.3 Improved 29-point lip model

To allow the model to cope with more extreme mouth shapes, we chose to add extra points halfway between the point pairs (0, 1), (3, 4), (4, 5) and (7, 0), and similarly for the inner shape of the mouth. This way, the contours could be followed more accurately, which leads to a better approximation of the mouth area. Using this new model, the performance of the AAM is expected to increase and training made easier, because the elements of the mouth would be more separated.

The new model, as shown in Figure 6.7, consists of 29 points: 24 points in the mouth area; 12 as the outer contour and 12 as the inner contour of the lips, 2 points at the base of the nose and 3 points at the chin. The definitions of these points can be found in Table 6.2 and Table 6.3.

Table 6.2: Point definitions of the outer mouth contour

0	Leftmost point still on the lips (left mouth corner)
6	Rightmost point still on the lips (right mouth corner)
2, 3, 4	Points placed in accordance with the philtrum (infranasal depression), namely, 2 and 4 at the foot of the philtral column and 3 in the place the where the philtrum meets the upper lip in the center
8, 9, 10	Points on the lower lip corresponding to point 4, 3 and 2 respectively
1, 5, 7, 11	Points placed such that the lip area is approximted as closely as possible. Their positions are preferred to be at equal distances from their neighboring points

Table 6.3: Point definitions of the inner mouth contour

12	Leftmost point in the cavity of the mouth but not on the lips. However, in the case of a closed mouth this is not possible to observe, so it should be placed such that it best describes the mouth line, always to the left of points 13 and 23 however.
18	Rightmost point in the cavity of the mouth but not on the lips. Again not observable in the case of a closed mouth. Instead it should then be placed such that it best describes the mouth line. Always to the right of points 17 and 19 however.
15, 21	Points corresponding to points 3 and 9 and following the philtrum

6.4 Training the lip model

To train an Active Appearance Model, a number of labeled training examples needed to be presented to the algorithm. We had no choice but to annotate these by hand. One of the advantages of AAM is that one occurrence of a shape is enough to have it modeled, so the amount of training data does not have to be so large. To allow for a robust model, each allowed lip shape should appear in the training set at least once.

Approach

The way mostly used to perform training of an AAM is he bootstrap method, where labeling and training are repeated until the model shows appropriate behavior for all new images. We gave it our own twist by searching for the most extreme mouth positions first.

To cover as many lip positions as possible, we tried to have a "system" while nominating the frames that would be annotated by hand. We figured it is best to use frames from different utterances, to ensure a good coverage of the data. The following scheme was employed to try and train an AAM efficiently and effectively:

1. Choose a couple of utterances. In whispered ones there may be the most extreme mouth positions. An attempt should be made to cover closed mouths and lip smacking as well.
2. Take one frame every 30 frames or so and annotate them. If possible, select frames that show the most extreme mouth shapes.
3. Put all of the annotated frames into the same folder. This may be a problem because a lot of images share the same filename. We wrote the tool "Dir2Filename" (Java) to allow the name of the folder to be appended to the filenames inside that folder.
4. Build the AAM using the training folder.
5. Add training images/utterances until the AAM seems to do fine, test with some extreme mouth positions.

Some scripts were written to circumvent limitations of the program during training (although the other annotator used DOS commands to accomplish the same thing). All Java programs manipulate the subfolders of the folder they are placed in. They are called from batch files, but all class files specified in the batch files need to be in the same folder as well. We will now give their descriptions.

Dir2Filename adds the folder names to all files in that folder as a prefix, allowing to put images from different utterances in the same folder and using them in the same training set. The operation is performed on all child directories. Afterwards, *MatchFilename* is performed.

DirFromFilename checks if the folder name is prefixing the file names, and removes it if that is the case. This reverses Dir2Filename. The operation is performed on all child directories. Afterwards *MatchFilename* is performed.

MatchFilename changes the BMP filenames in ASF files to whatever the filename currently is. The operation is performed on all child directories.

SkippedFrames makes a global missing frame report out of the ones produced at synchronization made for each utterance. It omits all utterances that had no missing

frames. The operation is performed on all child directories. This didn't have anything to do with training AAM directly, but was pretty useful when evaluating the recordings.

6.5 Visual validation

Evaluating an Active Appearance Model is not easy. The hardest thing we ran into while training Active Appearance Models, was to know whether the training set provided was extensive enough. According to the paper by Cootes, the training set needs to be *representative* of the data that could occur, because it will determine which model deformations are "allowed". If certain extreme mouth positions are omitted in the training set, the algorithm will come up with an approximation within the model boundaries.

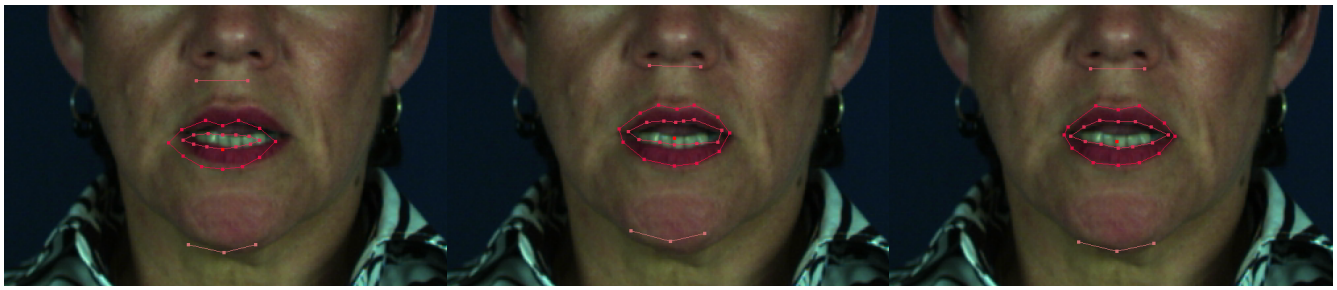


Figure 6.8: AAM search. In just 2 steps the final point coordinates are found. The image on the left shows the mean shape inserted to the frame, the image on the right shows the result after one search.

In the ideal case, we would work with just one model that works for all people in all lighting conditions. For Single Person New DUTAVSC there were 3 recording sessions, one in the original setup and two longer once. This amounts to a total of 1463 utterances having been recorded. For each session the subject was wearing different glasses. From session 1, 71 images were annotated; from session 2, 43 and from session 3, 23. The AAM trained on the first session images could be successfully applied to session 2 data, but not session 3 data. A more generic model trained on images from all sessions performed even worse for session 3, while a model trained on just sessions 2 and 3 worked perfectly for session 3.

This shows that a model made using one recording session cannot necessarily be used for subsequent recording sessions, even from the same person. This could be because AAMs depend on intensity information in the image, so a small change in lighting conditions could force the researcher to train a whole new model.

In [17] they used active shape models instead of AAMs to follow the lip contours. Using the model parameters as features they obtained decent results. We are using the *coordinates* of the points around the lips instead. This means we have to apply scaling and normalization afterwards. It could be beneficial to use the shape model parameters directly.

Another problem we encountered was that of proper initialization. The AAM algorithm is mainly good at tracking the lips (see Figure 6.8), not detecting them in the first place. For that, another algorithm is used, and we suspect that it *is* dependent on external factors like lighting conditions.

We trained a generic AAM using all the annotated frames for 8 different people. Once properly initialized, it generalized pretty well to the lip shape, for some random samples the initialization was perfect, even for one person who was not in the training set this was the case, and for a subject with a moustache. For 4 out of 9 people, initialization was no problem. We found out that some experiments with generic versus person dependent AMMs have already been done by others [29]. The results are that although constructing a generic shape model is relatively easy (measured in number of images required to train), fitting a generic AAM is far harder than fitting a person specific AAM because the effective dimensionality of the generic shape model is far higher than that of the person specific shape models.

An AAM sometimes places points outside of the image. This actually makes it robust against occlusion, but there appears to be no limit to the number of points that can be outside the frame, or the scaling and rotation factors for that matter. For faces, it might not be such a bad idea to set these limits.

AAMs had been trained for 8 different people when we decided to just use Single Person New DUTAVSC for training our automatic lip reader. We used different models for each session. At some point we decided to perform manual initialization before AAM search. The new configuration of AAM Annotation Lab at that point allowed us to actually see how the AAM performed from beginning to end of the recording. Only few mistakes were spotted. The mistakes that occurred were mainly after a missing frame gap, as discussed in section 1.1.1. This is understandable as the previous frame is always used for initialization of the next. We argue that once a model has been trained specifically for a session, the initialization should be okay.

7 Feature extraction

The feature extraction we performed is closely related to the model we used for the lips. So, first an Active Appearance Model was trained for the subject. Then, for every time frame of every recording, coordinates of the face points were computed. These coordinates were used to compute some features that seemed useful for lip reading. In this chapter, we will first discuss the chosen feature set (section 7.1), then discuss the algorithm implemented to perform the feature extraction (7.2). After that we evaluate the features based on criteria like robustness and correlation (7.3), and discuss results in section 7.4.

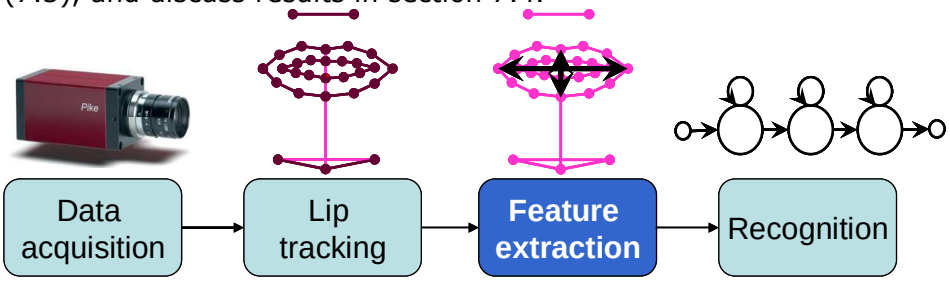


Figure 7.1: Visual speech recognition overview: feature extraction

7.1 Defining the features

First, we defined some basic visual features that could easily be derived from a point model and computed. They follow the point definitions given in Figure 6.7 of the previous chapter. From the landmarks detected on the speaker's face using AAM we computed some geometric features such as distances between key points and areas. The seven features we came up with this way are visualized in Figure 7.2. All distances are given in pixels.

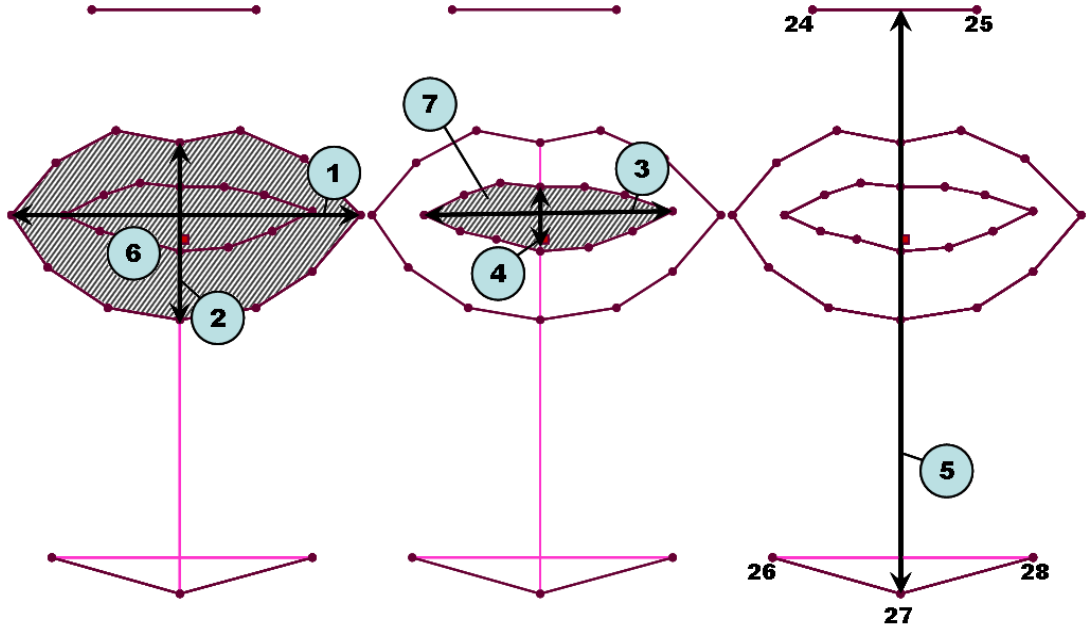


Figure 7.2: Visualization of the features: 1) Outer lip width, 2) Outer lip height, 3) Inner lip width, 4) Inner lip height; 5) Chin to nose distance, 6) Outer lip area, 7) Inner lip area.

7.1.1 Features computed from the outer lip shape

It was pretty straightforward to define and compute the features based on the global or outer mouth shape of the lips. We distinguish between mouth height (the length of the lips vertically), mouth width (the length of the lips horizontally) and mouth area (the area of the polygon approximating the lip shape). To compute the width, we took the points on the far sides of the lips, and for the height, we simply used the points on the middle line of the face, even though it may not result in the full maximum height being computed.

In terms of the numbered points defined in Figure 6.7 and Figure 7.3, we define mouth height as the distance between points 3 and 9, mouth width as the distance between points 0 and 6 and mouth area as the area inside of the outer mouth contour.

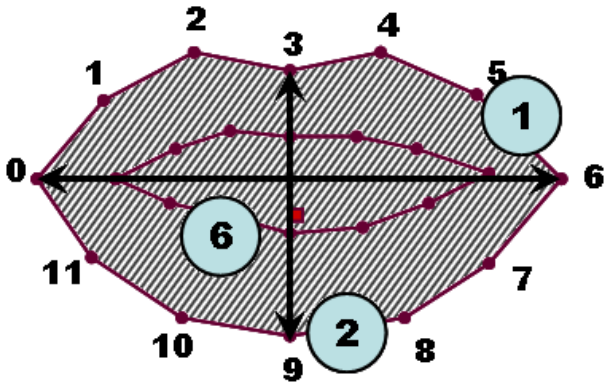


Figure 7.3: Visualization of the features computed from the outer mouth shape

7.1.2 Features computed from the inner lip shape

Defining the features computed from the mouth opening was a little less straightforward, because there isn't always a mouth opening. We decided to take this into account when defining the features. Nevertheless, the opening of the mouth is intuitively a speech feature important to sound production.

We discovered some problems when defining the height of the mouth opening, because not all people open their mouths symmetrically. Just taking the height of the middle points was therefore not accurate enough, so we took the largest observed opening height instead. Eventually, we defined aperture height as *the largest distance between the pairs of points (13, 23), (14, 22), (15, 21), (16, 20) and (17, 19)*. This is illustrated in Figure 7.4

Defining the width of the mouth opening required some thought, because the mouth might be closed or opened only partially. We decided that if out of the pairs of points (13, 23), (14, 22), (15, 21), (16, 20) and (17, 19) the distance was close to zero (some error is unavoidable with digital data) that part of the mouth would be considered closed, and disregarded in determining the width of the mouth opening. So, if the mouth is opened it would be the distance between the two mouth corners, if the mouth is closed it would be zero, and if partly opened it would be the minimum distance between two "closed" point pairs (or mouth corners). We defined aperture width as *the distance between the first point (or coinciding pair of points) to the last point (or coinciding pair of points) on the inner mouth contour, the points being counted in a left to right order*.

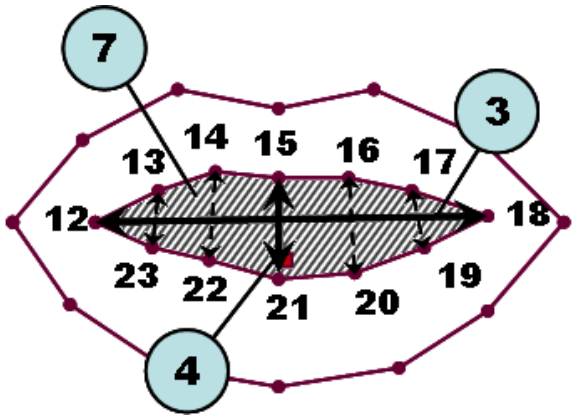


Figure 7.4: Visualization of the features computed from the inner mouth shape

The area of the mouth opening is computed in the same way as the area of the lips, with the note that there is a chance of the polygon being self-overlapping, especially in case of a (partly) closed mouth. In our implementation this could lead to negative values for the area, which we did not mind too much because the mouth would probably be closed in case of overlap anyway. We define aperture area as the area covered by the mouth aperture, namely the inner contour.

7.1.3 Features computed from nose and chin positions

From the two points on the nose and the three points on the chin we can compute the distance between the nose and chin. We chose to use the lowest point on the chin: the one in the center of the face. For the nose we chose the point exactly at the center of the two defined noise points. We initially thought that the distance between the nose and chin could give clues for onset/offset detection. But because a closed mouth does not necessarily mean that the jaws are closed, the mouth aperture width could be a better candidate. As a feature, it may still be valuable though: the visibility of the teeth – an important visual cue – depends on both the upper lip and the positioning of the lower jaw. This way we can compensate for the fact that our model is based on points and not on color values. We define nose to chin distance as *the distance between the line formed by points 24 and 25, and point 27* (see Figure 7.2).

7.1.4 Other possible features

Another feature we could have considered to add is for example, the distance between the center of gravity of the polygon approximating the lips, and the nose or chin. The distance between the two points of the nose is already used for scaling, as we will see later on. It is hard to come up with a point-based feature set that is speaker independent. It would be interesting to see which features are comparable between speakers. Intuition says some mouth features might be, like inner mouth area, but after scaling this might not be true anymore.

The displacement and acceleration of the specified features could be a dynamic addition to the feature set. The options for this are already built into the HTK tools. The delta and acceleration of point coordinates themselves could probably also be used as features, in which case the feature vector would have become a motion vector, as is the case with optical flow.

7.2 Feature extracting algorithm

Features had to be extracted from many different files containing model point coordinates. We designed an algorithm that could process these files automatically.

7.2.1 Formatting

HTK requires a feature vector to follow a certain format. We wrote a Java program to perform the feature extraction and format the data. The input of the algorithm is a hierarchy of folders containing frames in .BMP format and coordinates of the found model points for each frame in .ASF format (see section 6.2.2). Then, a recursive method goes through all the folders and computes the feature vector for all folders containing .ASF files.

Computing the distances was straightforward. The areas were a bit harder to compute, but we implemented an elegant method to compute the area of a polygon called the trapezium/trapezoid method. Here, the area below the bottom side of the polygon is subtracted from the area below the upper side, resulting in the area of the polygon itself. Because the Active Appearance Model sometimes results in a self-overlapping polygon (for the inner shape of the lips when the mouth is closed) the area can turn out smaller than it actually is using this method, even resulting in a negative value. However, in those cases the mouth cavity area is already around zero anyway, so this behavior doesn't seem harmful.

Another challenge to implement was the inner mouth width discussed in section 7.1.2. For its definition we said that it should be zero if the mouth is closed. So, for each pair of points lying above each other had to be checked whether their distance is zero (or negative in case of a self-intersecting polygon). Then, the width would be the distance between the last points that were placed on top of each other outward in from the corners of the mouth. This results in staircase-like graphs for closing/opening mouths as seen in section 7.3.

Table 7.1: Average mean and variance of "nose width" scaling factor for Single Person New DUTAVSC

session	mean mean	mean var
0	42.00	2.85
1	26.13	1.14
2	25.43	1.26
3	25.40	2.57
4	25.60	2.55
5	24.75	2.69
6	25.65	1.86
8	23.70	1.06

7.2.2 Normalization

The coordinates returned by the active appearance models had been transformed to the domain [0, 1) by dividing by the height and width of the frame. Before computing the distances, we reversed this operation. Then, we introduced another scaling factor to compensate for variation in distance between the recording subject and frontal view camera. All distances were normalized according to the most constant points of our model: the distance between the two points below the nose. Although a larger distance may have allowed for more accuracy, inspection shows a small variance in this particular value (see Table 7.1). The areas were normalized by

the square of the normalization factor. The only problem is that this scaling factor is not person-independent. We are not sure that there is any feature in the human face that is constant across individuals.

The algorithm is fast to process folders of annotated recordings, and would probably run even faster if the output was kept to a minimum during execution. It is also backwards compatible with the first AAM definition. The algorithm is initiated by running "LipFeatureExtracer.bat" provided that it and the required Java class files are in the same directory as the one of which the subfolders need to be processed. The description of the algorithm is given in Figure 7.5 and the full source code can be found in appendix C.

```
-> LipFeatureExtracter.bat
LipFeatureExtracter.class
DirFilter.class
ASFFilter.class

LipFeatureExtracter reads the coordinates from all asf files,
extracts features and writes them to HTK feature vector format.
A text file is also written for easy access.
With the exception of feature 5 and as long as path 0
is the outer mouth shape and path 1 is the inner mouth shape,
with a multiple of 4 points counted clockwise from the left-most point,
the features are independent of the model used.
The operation is performed on all child directories,
child directories of the child directories and so on.
```

Figure 7.5: Read-me of lip feature extraction algorithm

7.3 Visual validation of feature performance

It is hard to objectively measure the performance of a feature set. The most accurate way to validate the feature set is to measure the performance of a speech recognizer trained on that set and do so for all possible feature configurations. As this would take many human and computing resources, instead we decided to evaluate the performance visually by inspecting plots of the feature values.

The reason why a speech signal needs to be modeled by HMM in the first place is because the utterance of a phoneme/viseme never has the same length. Although to compare feature plots, they could be reformed using mathematics (with B-splines for example [30]), it would still be hard to say where a phoneme/viseme begins and ends in a recording.

7.3.1 Robustness

The first thing we wanted to validate is that the features are robust, i.e. show the same behavior for all realizations of the same viseme. Of course, it was not possible to inspect the total of the data, so we merely inspect some random samples taken from the recordings of single letters and digits. Most letters are composed of only one or two visemes. This made it easier to identify separate visemes. 13 out of 16

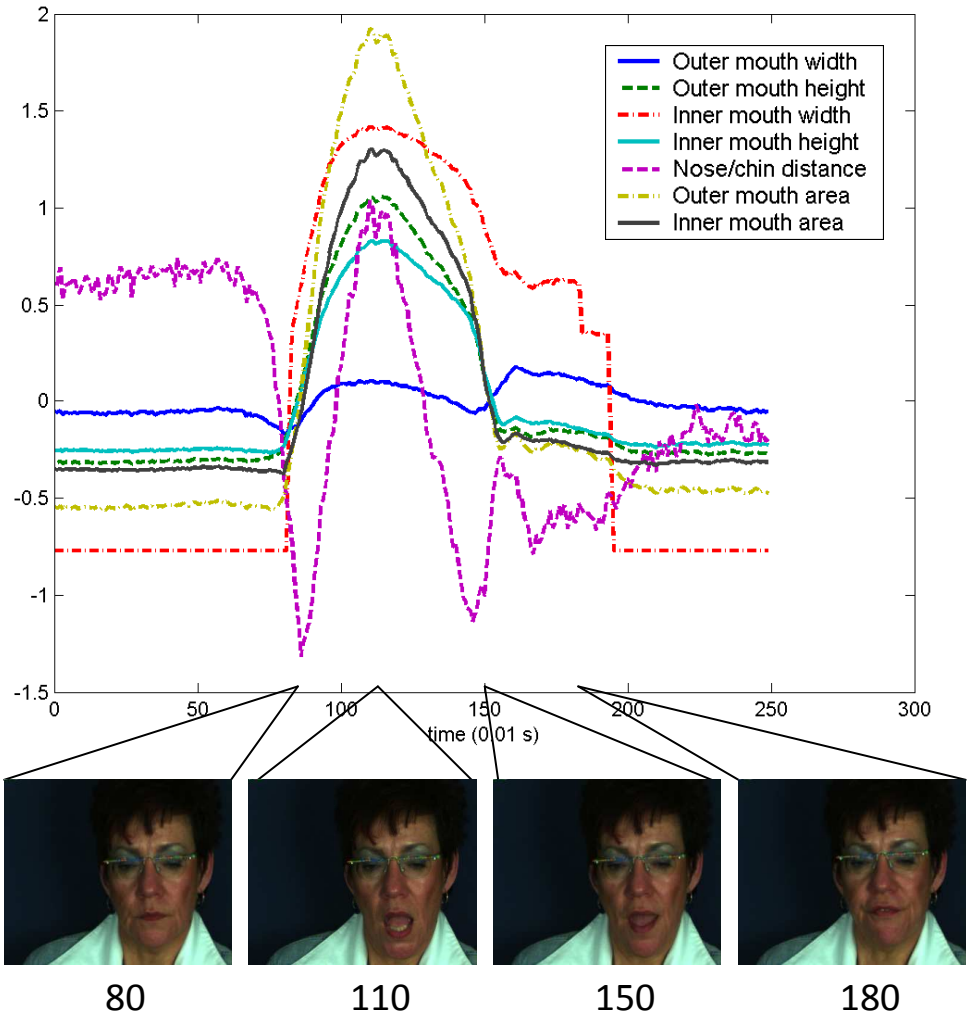


Figure 7.6: The seven features plotted for an instance of the letter F, containing the 2 visemes: eeh and fvw

visemes appear in the alphabet. 2 additional visemes could be inspected by looking at the digits as well. Figure 7.6 shows for a letter "F" (with the viseme representation "eeh fvw") how the feature values change per time frame (i.e. 100 frames per second). All graphs have been normalized around their mean value so they can be shown conveniently in one figure. As most features represent distances it should be easy to rationalize them.

Inner mouth width seems to provide a clue about the start and end points of the utterance: when the mouth is opened. Not all recordings were as nice as this example in terms of the mouth being closed before and after the utterance though. It is perhaps more natural for humans to start with their mouth in the position "@", as discussed in section 4.1.2. It could however be a nice cue to use in onset/offset detection.

It is also interesting to follow the nose/chin distance graph as this represents the openness of the jaw. The greater the nose-chin distance the more the jaw was opened. If the mouth is closed, it doesn't necessarily mean that the jaw is, as can be seen clearly from this plot.

The outer mouth width seems to be the last seemingly independently operating feature, because when we normalize the graphs not only by their mean, but also their variance, we get graphs like shown in Figure 7.7.

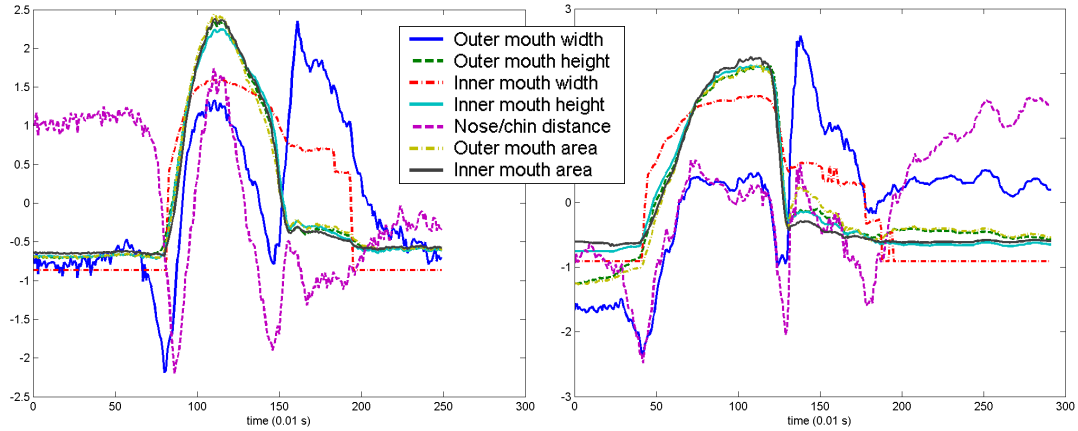
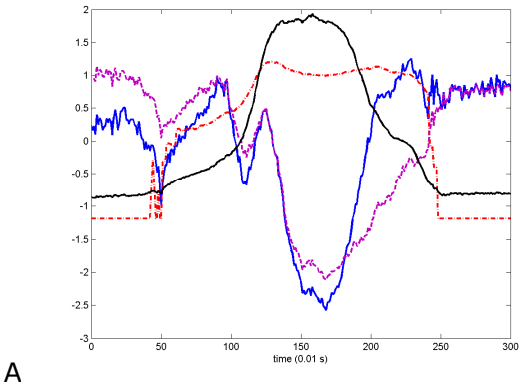


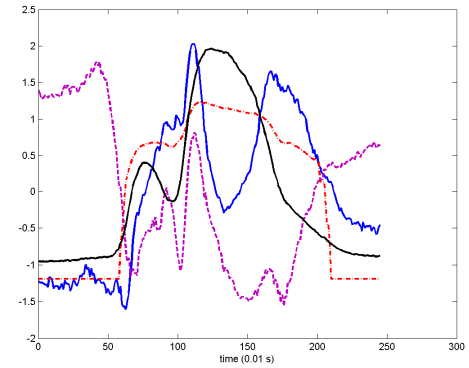
Figure 7.7: Feature values plotted for two utterances of the letter "F", of which the viseme representation is "eeh fvw", normalized according to mean and variance of the feature values over time

Figure 7.7 shows two instances of the letter F, normalized not only according to their mean, but also their variance. What is apparent here is that 4 out of 7 features appear to be so correlated that they end up almost exactly on top of each other. These are the inner lip area, outer lip area, outer lip height and inner lip height. This suggests that only one of these needs to be included in the final feature set. We assumed that inner mouth area would be a characteristic feature, since this is the opening through which the air has to go to produce sound, but perhaps these other features are just as good. We saw however that not for all recordings they end up exactly on top of each other, although they always act similarly. Sometimes it appears as if the variation is so perfectly balanced around zero, that the graph starts jagging when it is used as normalization. Sometimes they seem to have some delay, and at some points one feature is a little off for 10 frames or so.

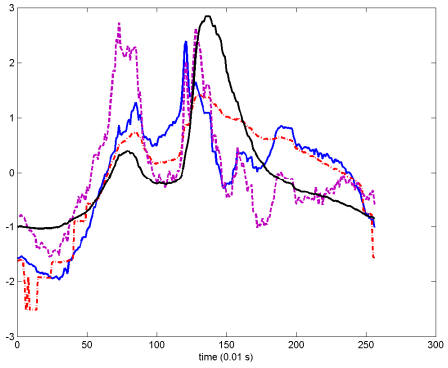
The two feature plots in Figure 7.7 show similar behavior. Looking at the inner lip width shows where the utterance starts and ends. Because of the graph normalization a graph also depends on the behavior during the rest of the recording time, so our focus should be on the general behavior within this time frame. The downside is that not all recordings start and end with a closed mouth, but whether this is the case is immediately evident from a plot. A look at the plots during the time that the mouth was opened shows that all features made similar movements, not regarding the exact scaling too closely. This suggests that the features are more or less robust for the letter "F". This appears to be the case for most letters we inspected.



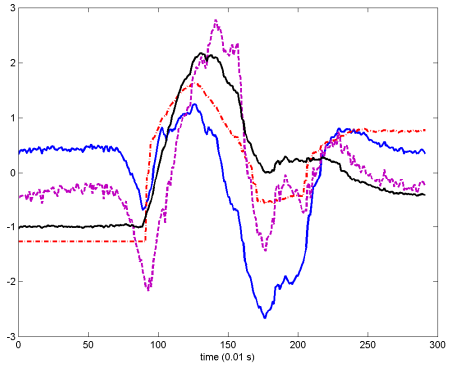
A



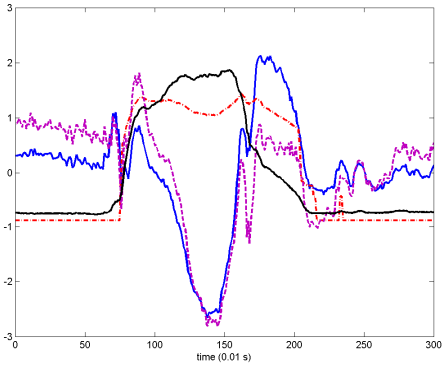
H



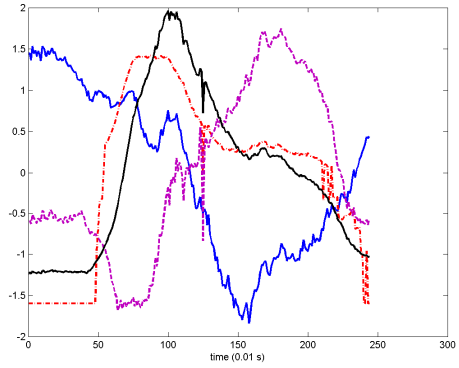
K



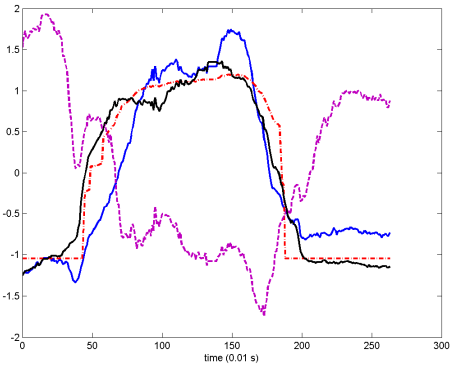
Q



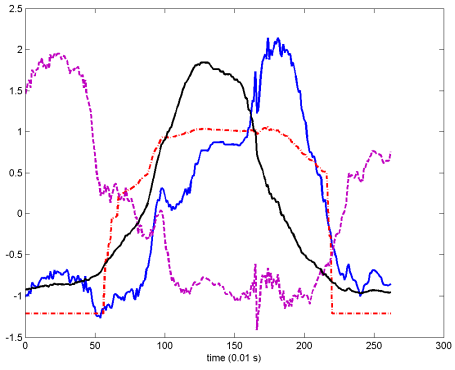
8



O



I



IJ

Figure 7.8: Feature values plotted for the letters and digits A (aa), H (h aa), K (gkx aa), Q (gkx oyu), I (ie), O (oyu), IJ (ei) and 8 (a gkx td).

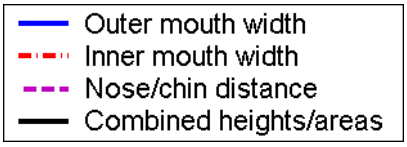


Figure 7.9: Legend for the plots in Figure 7.8.

7.3.2 Classification performance

After exploring how uniformly these features behave across instances, we wanted to know how well suited these feature are for viseme classification. We will do this by plotting the features for all the viseme classes that could be covered this way. There is one viseme that does not appear in the combined set of letters and digits, namely the "shzj". For the remaining 15 visemes we will try to find out if this feature set would be able to make a distinction between them. For the graphs in this section we will include an average of the features "outer mouth height", "inner mouth height", "outer mouth area" and "inner mouth" instead of including them all, to make the graphs easier to interpret (naturally after validating whether for that particular recording this generalization can be made).

Figure 7.8 and Figure 7.10 show the normalized plots of the resulting four features, for different letters of the alphabet, extended with two digits to cover "at" and "a". The viseme representations of the letters and digits can be found in Table 9.1 and Table 9.4 respectively. Although we have looked at other instances of these utterances, we chose to display the nicest ones we encountered, or for which the inner mouth width showed a nice cycle of opening at the beginning and closing at the end.

1. **aa**

First of all, we will compare the behavior of the features for all that contain the viseme "aa". Figure 7.8 shows feature plots for the letters A (aa), H (h aa) and K (gkx aa). In each of these cases, the combined height and area feature cluster shows a single bump, while the outer mouth width and nose to chin distance show a dip, indicating that the mouth becomes narrower and the jaw is opened. According to the inner mouth width, the mouth is opened up wider.

2. **h**

The viseme "h" is only represented by one example (H). For the combined feature, a second bump shows in front of the bump of the second viseme. Nose to chin distance seems to show some specific behavior as well.

3. **gkx**

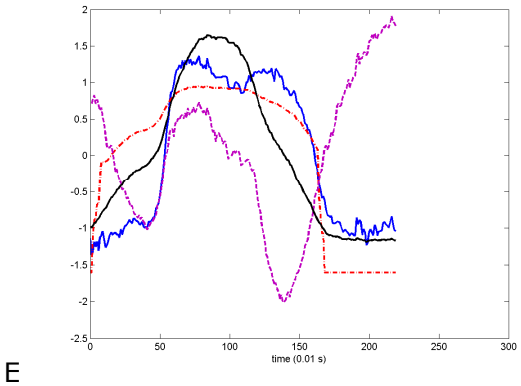
For the viseme "gkx" we have several examples. First of all there is the letter K, then there is Q (gkx oyu) and 8 (a gkx td). Their feature plots mainly show bumps for all features where "gkx" should be observed.

4. **a**

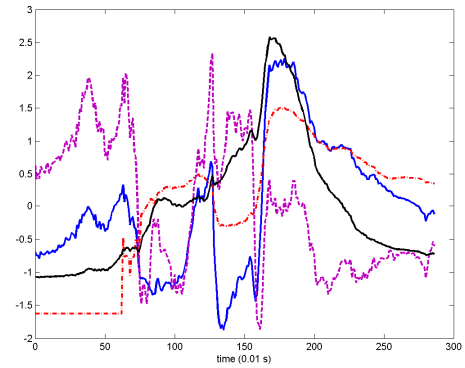
Viseme "a" only has the example 8. The behavior of the features seems to be almost exactly the same as for "aa". This might mean these viseme classes are not that well separable.

5. **oyu**

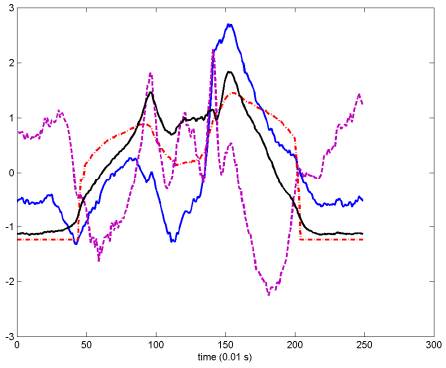
For "oyu" the examples are Q and O (oyu). The heigt/area feature and outer mouth width show a similar decrease. According to the nose to chin distance, the jaw opening becomes smaller for a moment as well.



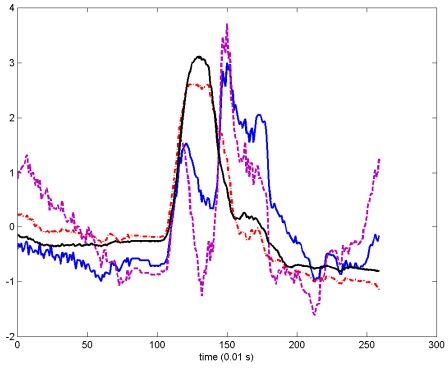
E



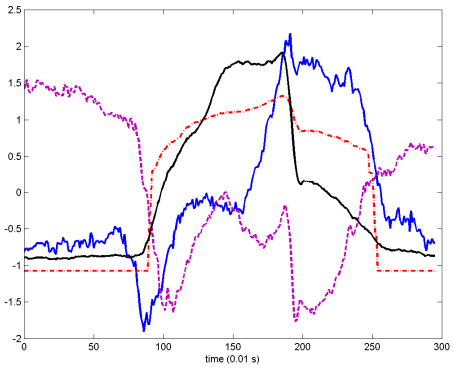
D



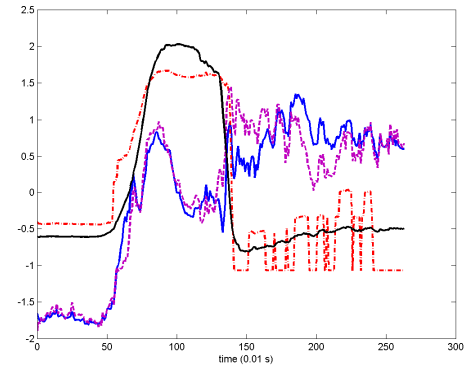
C



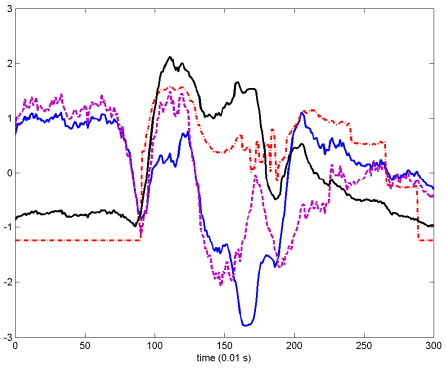
L



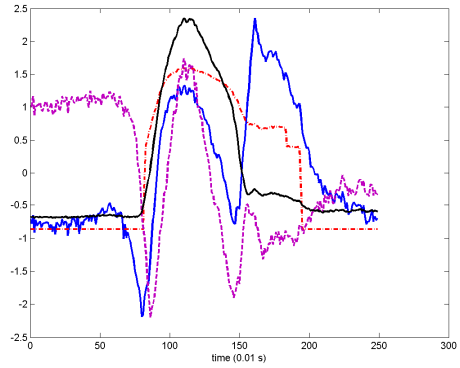
S



M



7



F

Figure 7.10: Feature values plotted for the letters and digits E (iee), D (td iee), C (sz iee), L (eeh l), S (eeh sz), M (eeh m), 7 (sz iee fvw at) and F (eeh fvw).

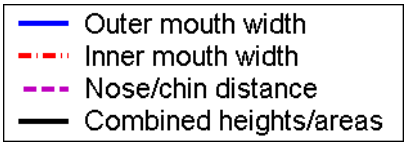


Figure 7.11: Legend for the plots in Figure 7.8.

6. ie

For "ie" (only example I (ie)) all features except for nose to chin distance show a bump, that seems to last the full time that the mouth is opened. The viseme length might be a nice additional feature to recognize this viseme.

7. ei

Of "ei" there is also one example: IJ (ei). It seems to look a bit like "aa", but the outer mouth width shows some different behavior, with an extra bump at the end.

8. iee

Switching over to Figure 7.10, we will discuss the viseme "iee" next. Examples shown are letter E (iee), D (td iee), C (sz iee) and 7 (sz iee fvw at). The features inner mouth width, outer mouth width and the composed feature all show a bump. The nose to chin distance however shows first a bump and then a dip, and all within the time span indicated by inner mouth width.

9. td

For viseme "td", one example is included (D). Before the formation of the "iee" begins, the features show a lot of complex behavior, with the most prominent being a sudden peak by the outer mouth width and nose to chin distance. Apparently the mouth becomes a little wider and the jaw is closed.

10. sz

For "sz", two instances are included: C and S (eeh sz). The features that seem to show the same behavior are the outer mouth width, which shows a bump, and inner mouth width, which is at a slightly lower level as the vowel for these examples.

11. eeh

For the viseme "eeh", there are 4 examples: S, L (eeh l), M (eeh pbm) and F (eeh fvw). In the clearest examples, it looks like a short version of "iee", with the difference that the nose to chin distance and outer mouth width both show a bump and a dip.

12. l

For "l" there is one example: L. At the beginning of the viseme, both outer mouth width and nose to chin distance start in a peak and decrease from there. The other features also show a decrease, but these are not preceded by a peak.

13. pbm

The example or viseme "pbm" is M. It can be nicely seen from the plot that when the "pbm" is formed, the mouth is closed (inner mouth width). The other features stay at a constant level in the mean time.

14. fvw

"fvw" appears in both F and 7. Furthermore, another instance of F was shown in figure Figure 7.7. The main characteristic seems to be a peak for the outer mouth width, and movement toward the neutral position for all features.

15. at

"at" only appears at the end of digit 7. We cannot be entirely sure however that the "n" at the end of "zeven" was not uttered. The length of the word should normally give a clue about that, but 7 already has at least 4 visemes and it is hard to say where the "fvw" turns into an "at" and optionally a "gkx". After what appears to be the "fvw" viseme, the graphs of all features gradually decrease, before at last the mouth is closed. In other work "at" has been assumed to be the neutral mouth position, and could perhaps be used as an alternative for onset/offset detection. After inspecting the feature plots for each viseme in Figure 7.8 and Figure 7.10, we can conclude that these features seem to be very capable of distinguishing between at least some of the visemes. Table 7.2 shows which features seem to be important for which viseme, together with the accompanying characteristic behavior.

Table 7.2: Feature behavior for different visemes: +) bump -) dip -+) increase +-) decrease

	aa	h	gkx	a	oyu	ie	ei	iee	td	sz	eeh	l	pbm	fvw	at
Outer mouth width	-		+	-	-	+	-+	+	+	+	+-	+-		+-	
Inner mouth width	+		+	+		+	+	+			+		-		
Nose/chin distance	-	+	+	-	-	-	-	+-	+		+-	+-			
Height/area features	+	+	+	+	-	+	+	+			+				

This evaluation was also useful because it helped us notice a labeling error with the letter "Q" as plain "gkx" instead of "gkx oyu". Wrongly labeled data can make it harder to train a recognizer and can introduce errors.

7.4 Conclusions

We found that some features are more useful than others. The first thing we observed was that for normalized graphs (centered around their means and divided by their standard deviations) four features show great overlap indicating high correlation. They are inner lip height, outer lip height, inner lip area and outer lip area. If they are the same it is not useful to include them all. This also shows that the point tracking and feature extraction used is robust.

Another thing we saw is that because of its definition the feature "inner lip width" is a useful indicator to see whether the mouth is closed or not. If the mouth is closed its value everywhere will just be zero. A horizontal line in the graph of any other feature would mean that there is a series of missing frames in the recording. This leaves us with two more features that can be used to classify the visemes: outer lip width and nose to chin distance. That makes a total of at least four distinct features.

One important challenge remains: these features are not person-independent. Even if another Active Appearance Model is trained for a new person and the points can be located successfully, people's facial features are not always the same. In fact, any computed distance between points could be different, even when they are normalized by a distance that is the same for all humans (the distance between "nose points" we used is probably not, though for recordings of the same person it complies). This renders a recognizer trained on data from one person unreliable for another. Perhaps some kind of speaker adaptation could be applied.

8 Implementation

In this chapter we will describe the final implementation of our visual speech recognizer. There are four main phases in the development of a speech recognizer: data preparation (section 8.1), training (section 8.2), testing (section 8.3) and analysis. For most of the tasks, HTK tools are readily available, and most who have trained a speech recognizer using HTK before will recognize the approach taken, as it is also described in the HTK manual [2]. This chapter should provide the reader with enough background information to train a speech recognizer of their own.

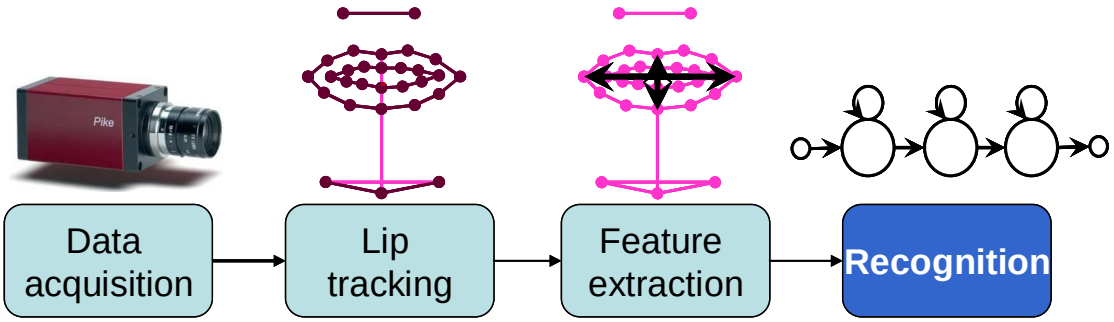


Figure 8.1: Visual speech recognition overview: recognition

What we implemented was a speaker dependent recognizer, using HMM with three emitting states for each viseme, using the viseme set presented in chapter 4. However, while the approach is about the same for all recognizers, it is not possible to quickly train a recognizer using only a portion of the data or changing a parameter, as some steps involve the training supervisor to do something, like designing and copying HMM, or selecting data. In theory, a shell could be used to perform all the steps, but this was not the focus of our work.

8.1 Data preparation

In the previous chapters we described how feature vectors were attained from the video material. The training data available for the lip reader were the features extracted from Single Person New DUTAVSC. Aside from the feature vectors however, the automatic lip reader would require a formatted dictionary, grammars and labels for each recording. We started out with the grammar, then the dictionary and finally the labels.

Within HTK, grammars are stored within a word net. There are two different ways of defining a grammar in HTK. The first is a restrictive one using Extended Bachus-Naur Normal Form, which is sufficient for simple recognition tasks, like digit or digit string recognition (it is important whether a fixed length of such a string is defined or not). An example of such a grammar is given in Figure 8.2. But also more complex tasks like the bank application seen before can be captured in a grammar, as presented in Figure 5.2.

```
$letter =
       A | B | C | D | E |
       F | G | H | I | J |
       K | L | M | N | O |
       P | Q | R | S | T |
       U | V | W | X |
       IJ | Z;

$digit = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9;

($digit $digit $digit $digit $digit $digit $digit $digit |
$letter $letter $letter $letter $letter $letter $letter $letter |
$digit |$letter)
```

Figure 8.2: Example grammar in EBNF as accepted by HTK. This grammar allows for single digits, single letters and letter or digit strings of length 8.

The other type of grammar that can be applied is a bigram language model. To train N-grams, however, one requires an extensive text corpus of natural language. What we initially did was using our sentence pool of Polyphone sentences to train a bigram, while a better approach would have been to take an existing bigram model and simply delete the words that did not appear in our dictionary.

This brings us to the dictionary. One was already available to us namely the same dictionary as used by Polyphone and previous work on speech recognition [5]. A dictionary as required by a speech recognizer consists of words and their phonetic transcriptions. We edited the dictionary and transcriptions by performing the phoneme to viseme mapping given in Table 4.6.

Through our recording software most of the data (except for the answers to open questions) had already been pre-labeled. Labels are used to train and evaluate the recognizer (by comparing the label to the recognizer output) and consist of (a series of) dictionary words indicating what was said on the recording. HTK requires all the labels for training to be listed in a master label file. Also, a mapping is performed from words to visemes, where silence between words is inserted. A label needs to match the utterance perfectly, including any mistakes. We decide to only use utterances that were fluent and without mistakes, because any irregularities that show in a sound recording would probably be amplified in the visual modality. There were probably more irregularities in the recordings than we could hear during auditory validation.

Furthermore, all the words presented needed to appear in the dictionary. HTK does not cope with special characters well. A HTK compatible dictionary can contain capitalized and non-capitalized words, numbers, dashes and single quotes, but that is about all. For English this would be fine, but in Dutch special characters change the meanings and pronunciations of words sometimes (e.g. "een" vs. "één"), plus a good spelling often depends on it. In the end, we just capitalized all words and word labels, wrong spelling or not. The other consequence of this restriction of HTK is that we were unable to use the standardized ASCII IPA notation for the dictionary. Instead we had to use phoneme and viseme representations HTK could process. We discussed this in chapter 4.

8.2 Training

Training a speech recognizer is an incremental process. As pointed out in chapter 3, the only (currently known) way to train a Hidden Markov Model is to use an approximate algorithm, namely the Baum-Welch re-estimation algorithm. In the end, for every speech primitive (in our case visemes extended with one or model silence/noise models) one Hidden Markov Model will be trained and together with its state output probability density functions and state transition probabilities stored in its own file.

A good initial estimate of a triphone HMM using Gaussian mixtures is a trained monophone HMM using a single Gaussian. To define the model topology, first prototype HMMs need to be composed. We chose a topology of five states per viseme, of which three emitting. Prototype HMMs contain the size of the feature vector that will be presented, the number of states and the initial state transition probabilities. Initial HMM of this kind are all identical, with identical initial values for mean and variance. This is how we constructed the prototype using HCompV (words between % are variables):

```
HCompV -T 1 -C ../configs/%config% -m -f 0.01 -S ../lists/%trainset% -
M ../models/hmm0 ../protos/%proto%
```

To obtain the final set of initialized monophone HMM, they need to be re-estimated three times by using the HTK tool HERest thrice. Calling HERest is done like this:

```
HERest -T 1 -C ../configs/%config% -S ../lists/%trainset% -
I ../../Data/%database%/Labels/%subcorpus%_MLF0_viseme_transcription.mlf -
H ../models/hmm%lasthmm%/%macros% -H ../models/hmm%lasthmm%/%hmmdefs% -
M ../models/hmm%newhmm% ../blocks/monovisemes0_%database%_%subcorpus%
```

The long and short pause silence models that are used to model noise among other things can be made more robust by adding extra state transitions. A short pause is allowed to be skipped while a long pause may be repeated to occupy a greater time span with impulsive noise. We added these state transitions and re-estimating twice more to make the set of HMMs incorporate these variations. The silence models are fixed using HHEd:

```
HHEd -A -D -H ../models/hmm%lasthmm%/%macros% -
H ../models/hmm%lasthmm%/%hmmdefs% -M ../models/hmm%newhmm%
sil.hed ../blocks/monovisemes1_%database%_%subcorpus%
```

Another thing that had to be accounted for, are words in the dictionary for which multiple pronunciations exist. Viseme transcriptions that were previously generated automatically by picking the first available pronunciation in the dictionary, can be realigned to more accurate versions using the model trained thus far. This is done using the HTK tool HVite, followed by re-estimating twice more. HVite is called as such:

```
HVite -a -b SENT-START -m -o SWT -y lab -T 1 -t 250.0 150.0 1000.0 -
C ../configs/%config% -H ../models/hmm%lasthmm%/%macros% -
H ../models/hmm%lasthmm%/%hmmdefs% -l '*' -
i ../workdata/%database%_%subcorpus%_trainset_MLF1_viseme_transcription_aligned.
mlf -I ../../Data/%database%/Labels/%subcorpus%_MLF_word_transcription.mlf -
S ../lists/%trainset% ../../Data/%database%/Dictionaries/%subcorpus%_viseme_dict
ionary.dic ../blocks/monovisemes1_%database%_%subcorpus%
```

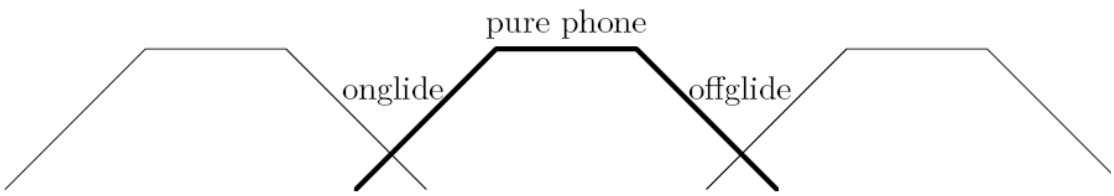


Figure 8.3: The three phases of a phoneme [21]

Now that monophone HMM had been trained for all the visemes, we could extend this to triphones that can model the dependency between phones more accurately. A phone is usually regarded as a having three phases; the onglide, pure phone and offglide (see Figure 8.3). The onglide and offglide often overlap with those of previous and sequential phones. With triphones instead of for each phone, one HMM will be needed for each phone with an onglide to the specific previous phone and offglide to the specific next phone. As one can imagine this causes an explosion in the number of HMM that needs to be estimated, making pruning and storing the HMM in a binary file important to consider for optimization.

Context-dependent triphones can be made by simply cloning monophones and then re-estimating using triphone transcriptions. Using the label editor HLEd, the monophone transcription can be made into a triphone transcription, disregarding the long pause sil at the beginning and ending of the utterance and short pause sp as the word boundary symbol.

Because this approach would require much more training data than monophones, there can be made use of parameter tying between HMMs. This way, transition matrices can be shared by multiple HMMs, increasing the amount of training data for each such matrix. When re-estimating tied parameters, the data which would have been used for each of the original untied parameters is pooled such that a much more reliable estimate can be obtained. Some triphones will occur only once or twice and so very poor estimates would be obtained if tying was not done. The most reliable state tying can be done according to a linguistic model by decision tree clustering using the HTK tool HHEd, but as we didn't have such a thing at our disposal, we used a data driven model. This model is also available through the tool HDMan as well as some scripts provided with HTK. Re-estimation again must be done twice to obtain the final result.

At the same time, the number of Gaussians mixtures could be increased to make the probability density function for each state to better fit the data. This process has to again be performed incrementally, because too many mixtures would cause the model to be overtrained and perform badly on new data [31]. And again, this increases the number of parameters so that the required training time is expected to be longer.

8.3 Evaluation

We trained four types of recognizers. They differ with respect to the inclusion of delta and acceleration factors of the features described in chapter 1, and whether

monophones or triphones were used. To obtain the best result flat training was applied, and for all 32 possible combinations of Gaussian mixtures the one that gave the best performance was chosen. Each recognizer was trained on 85% of the type of data we wanted to use the recognizer for, leaving a test set of 15%. While it is possible to train the recognizer on all data, because what we are actually training are the *viseme-level HMMs* after all, we thought this way more accurate results could be obtained.

To evaluate the performance of the completed recognizers, several things are required: test data that has not been used for training, the recognition network (generated by our grammar) and the dictionary. Recognition is done using the Viterbi algorithm implemented by the HTK tool HVite. It can process a whole list of test data files in sequence and several options concerning the word-cross probability and weighing factor of the grammar can be set additionally.

The final performance can be determined by comparing the output of HVite to the known labels using the tool HResults. HResults has many options to draw statistics from the results. In the standard case, it will give the percentage of sentences and words that were recognized correctly (the percent correct and percent accuracy, see section 2.1), the word accuracy rate and the number of errors of each of the types. If certain instances are allowed to be confused, like in our case at a given point some of the letters which had the same viseme transcription, there is an option to make them equivalent to HResults. It can also show a full confusion matrix so we can see all the substitution errors that were made for each word. Here is how HResults is called:

```
HResults %classescluster% -
I ../../Data/%database%/Labels/%subcorpus%_MLF_word_transcription.mlf ../blocks/
monovisemes1_%database%_%subcorpus% ../results/recout_%database%_%subcorpus%_%fe
aturestype%_%deltaacc%_1B_word.mlf
```

In the next chapter we will describe the recognition performance obtained by these recognizers.

9 Experiments and results

Once we finished the implementation of the automatic lip reader, we were able to design some experiments to test its performance. The data used was from the Single Person New DUTAVSC subcorpus (see section 5.3.2), which contains recordings of sentences, single digits, random digit sequences of length 8, single letters and random letter sequences of length 8, and all of a single person.

The first experiment involved digit recognition (9.1), which is a simple classification task that took us back to the basics. The second experiment involved letter recognition (9.2), which would be a spelling task. Then, we will compare the results of these experiments (9.4) and discuss the project-wide results (9.5).

9.1 Digit recognition

The first and simplest experiment we conducted involved digit recognition. There are 10 words and thus 10 classes the data can fall into. We also did this for series of digits using a fixed grammar. The "pure chance" performance level is considered to be 10% for digit recognition. We used a restrictive grammar that only allowed either a single digit or a string of 8 digits to be recognized. The data presented was also of this kind. The viseme representations of the words of this recognition task are shown in Table 9.1.

Table 9.1: Viseme representations of the 10 digits in Dutch

Digit	*Viseme representation*
1	*iee gkx*
2	*td fvw iee*
3	*td gkx ie*
4	*fvw ie gkx*
5	*fvw ei fvw*
6	*sz eeh sz*
7	*sz iee fvw at*
8	*a gkx td*
9	*gkx iee gkx at*
0	*gkx oyu l*

For a subset of the data corpus consisting of 31 utterances and 73 digits, the best result obtained was 67.7 % of the sentences correct and 78.1 % of the words, with an accuracy of 68.5 % considering the insertion errors. To get this result, the delta and acceleration coefficients were included, and monophones were used with 24 mixtures. The results for each type of recognizer are shown in Table 9.2.

Table 9.2: : Percent correct and Word Recognition Rate for recognition of 73 digits

	No delta/accelaration	Including delta/acceleration
Monophones	69.86 %; WRR=61.64 %	**78.08 %; WRR=68.49 %**
Triphones	65.75 %; WRR=60.27 %	72.60 %; WRR=50.68 %

Regarding the confusion matrix in Table 9.3, the following can be remarked. The digits to get confused most often are the "3" and "4", which share a viseme but are otherwise distinct. And "1" and "9", but this is no wonder as both letters contain "iee

gkx", and furthermore, the phonetic transcription for 9, "n ee g at" is perhaps inaccurate, as the "n" at the ending of the word "negen" may be clearly pronounced when speech is not uttered at a fast rate.

Table 9.3: Confusion matrix for digit recognition task of 73 digits

	0	1	2	3	4	5	6	7	8	9	total
0	**8**	0	0	0	0	0	0	0	0	0	8
1	0	**3**	0	1	0	0	0	0	0	2	6
2	0	0	**4**	0	0	0	0	0	0	0	4
3	0	0	0	**8**	3	0	1	1	0	2	15
4	0	0	1	1	**8**	0	0	0	0	0	10
5	0	0	0	0	0	**6**	0	0	0	0	6
6	0	1	0	0	0	0	**3**	0	0	1	5
7	0	0	0	0	0	0	0	**3**	0	0	3
8	0	0	0	0	0	0	0	0	**10**	0	10
9	0	2	0	0	0	0	0	0	0	**4**	6

9.2 Letter recognition

In spelling there are two letters that often seem to get confused by Dutch listeners, namely "M" and "N". Adding the visual modality however this task becomes much easier, because their viseme representations are different. In this example the value of lip reading in spelling becomes evident.

The next experiment we did was concerned with letter recognition. It is a slightly more difficult task than digit recognition, because the word representations of the letters are generally shorter than those of digits, namely 1 to 3 (but usually 2) phonemes/visemes. It could give some more insight on the recognition of pure visemes. The number of classes here is 26 (or 20 as we will see later) and the approach is the same as for digit recognition. We imposed a grammar on the recognition results that allowed for single letters, letter strings of length 8, and letters strings of arbitrary length (resulting from spelling of actual words).

There was however one thing we overlooked while performing this task: as seen in Table 9.4, for some letters of the Dutch alphabet the viseme representations are equal. Due to the mapping from phonemes to visemes the distinctiveness of the letters of the alphabet has become less. While for recognition tasks with longer utterances this would not be a problem, for really short words like word representations of letters, it is apparent. The HTK recognition tool HVite solves the problem of multiple dictionary entries for the same viseme representation by picking the first one. With 26 classes, of which there are 6 pairs of equal looking letters, this should lead to an error in 6 out of 26 cases (23%). Indeed, clustering these pairs together in one class gave an increase in performance of about 10%. Considering they were just 186 *random* letters, it is possible that the calculated 23% was only 10% in reality. The problem of the propagation of misclassifications was discussed in section 4.2.2.

Table 9.4: Viseme representations of the letters of the Dutch alphabet

Letter	*Viseme representation*
A	*aa*
B, P	*pbm iee*
C	*sz iee*
D, T	*td iee*
E	*iee*
F	*eeh fvw*
G, J	*gkx iee*
H	*h aa*
I	*ie*
K	*gkx aa*
L	*eeh l*
M	*eeh pbm*
N, R	*eeh gkx*
O, U	*oyu*
Q	*gkx oyu*
S	*eeh sz*
V, W	*fvw iee*
X	*iee gkx sz*
IJ	*ei*
Z	*sz eeh td*

In the end, there were 20 classes (of which 6 clusters) for the letter recognition task. This amounts to a 5% pure chance recognition rate. For a subset of 60 utterances and 186 words in total, the best results obtained were those of the recognizer with delta and acceleration, using triphones and 18 Gaussian mixtures. According to Table 9.5, 49.5% of all words are correctly recognized with a word recognition rate of -12.9%. Furthermore 31.7% of all sentences were recognized correctly.

Table 9.5: Percent correct and Word Recognition Rate for recognition of 186 letters

	No delta/accelaration	Including delta/acceleration
Monophones	40.32 %; WRR=-35.48 %	44.62 %; WRR=-25.27 %
Triphones	37.10 %; WRR=-44.62 %	**49.46 %; WRR=-12.90 %**

The confusion matrix given in Table 9.6 can give us some insight about the classification performance of the feature set, especially since the letter representations for 6 vowels (A, E, I, [O, U] and IJ) consist of just one viseme. Apparent is that one viseme that seems to get confused a lot is gkx, probably because it is formed in the back of the mouth and it is kind of a "garbage collection" viseme, containing a lot of different consonants. Digits and letters that contain this viseme are [G, J], [N, R], K, Q, X, 1, 3, 4, 8, 9, 0.

More interesting perhaps than which letters get confused are the letters that do not get confused. For F, H, I, [O,U] and [V, W], over 70% are correctly recognized. Because random letters were used, a lot of classes are actually underrepresented. IJ and [N,R] are never recognized correctly, despite being represented in the test set at least 5 times.

Table 9.6: Confusion matrix for letter recognition task

	A	[B, P]	C	[D, T]	E	F	[G, J]	H	I	IJ	K	L	M	N	[O, U]	Q	S	[V, W]	X	Z	total
A	**0**	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1
[B, P]	0	**13**	0	0	0	0	0	0	4	0	0	0	2	0	0	0	0	0	0	0	19
C	0	0	**3**	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1	0	6
[D, T]	0	0	2	**1**	0	0	0	0	0	0	0	2	0	1	0	0	0	0	2	1	9
E	0	0	1	1	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
F	0	0	0	0	0	**8**	0	0	0	0	0	1	0	0	0	0	0	0	0	2	11
[G, J]	0	0	0	2	2	0	**3**	0	3	0	0	0	0	0	0	0	1	0	3	0	14
H	0	0	0	0	0	0	0	**10**	1	0	0	1	0	0	0	0	0	0	0	0	12
I	0	0	0	0	0	0	0	0	**3**	0	0	0	0	0	0	0	0	0	1	0	4
IJ	0	0	0	0	0	0	0	1	0	**0**	0	2	0	0	0	0	0	0	2	0	5
K	0	0	0	0	0	0	0	2	1	0	**6**	0	0	0	0	0	0	0	0	1	10
L	0	0	1	0	0	0	0	0	0	0	0	**7**	0	0	0	0	0	0	0	0	8
M	0	0	0	0	1	0	0	1	0	0	0	1	**6**	0	0	0	0	0	0	0	9
[N, R]	0	1	0	0	0	0	0	1	1	0	0	5	0	**0**	0	0	0	0	2	0	10
[O, U]	0	0	0	0	0	1	0	0	0	0	0	0	0	0	**7**	0	0	0	0	0	8
Q	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	**3**	0	0	0	0	7
S	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	**4**	0	2	0	9
[V, W]	0	2	0	0	0	0	0	1	0	0	0	0	0	0	0	2	0	**12**	0	0	17
X	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	1	0	**3**	0	7
Z	0	0	1	0	0	2	0	0	0	0	0	0	1	0	0	0	0	0	0	**3**	7

That the restrictive grammar imposed on the results allows for letter strings of arbitrary length, could be responsible for part of the insertion errors. Although for spelling tasks it is only logical to have no length imposed on a word, in our case, we did not have any recordings of actual spelling, because they were only included in the original session setup, and because of the frame skipping issue discussed in section 1.1.1 not any of them were usable (see table Table 9.7). It is possible that the amount of visible articulation for visemes requires a more restrictive grammar to be used. We will have to see if using the more restrictive grammar variant will improve performance and word recognition rate especially.

9.3 Comparison

The results we obtained match the predictions about theoretical lip reader performance addressed in section 4.2. We cannot be sure whether this can be improved by increasing the amount of training data. The difference of instances per symbol to recognize could also have an influence, although the effect should not be as great because we trained our HMM at viseme level and not at word level. If we did the latter (and use larger HMM to model the classes) performance can be expected to increase, even when the visemes are the same. Another possible cause can be the length of the expression. Letters are of length 1-3, while digits are of length 2-4. A longer length gives the Viterbi algorithm more opportunity to make use of the context information.

Table 9.7: Number of usable recordings for first session of Single Person New DUTAVSC

Number	Speech type	Utterance type	<10 missing frames
3	normal	Random digit sequences of length 8	3
3	fast rate	"	**0**
3	whispering	"	**0**
3	normal	Spelling a random word of variable length	**0**
3	whispering	"	**0**
3	normal	Lists of random words of length 8	3
3	fast rate	"	1
3	whispering	"	2
5	normal	Fixed grammar bank application sentences	4
5	fast rate	"	5
5	whispering	"	5
5	normal	Random sentences taken from Polyphone	5
5	fast rate	"	4
5	whispering	"	4
5	normal	Every day use common expressions	5
5	normal	Short answers to random open questions	**0**

The difference in performance of digit and letter recognition may be due to a number of reasons. First of all, as discussed before the restrictiveness of the grammar for letter recognition will need to be looked into. After that there are still two possible reasons. Firstly the number of classes for digit recognition is 10, while for letter recognition it is 20. Because of this there are more opportunities for confusion, especially since the words for letters are shorter than those for digits (see Table 9.8). Secondly, there was more training data available for digits than for letters. Because there are twice as many letter classes, twice as much training data would be needed to get the same training result as for the digits.

Table 9.8: Frequency of word lengths for the viseme representations of digits and letters

# visemes	1	2	3	4	average
Digits	5	13	2	0	3,7
Letters	0	1	7	2	1,55

After that, whether it is because of lack of data or number of classes or number of training samples is easy to determine by just cutting back on those training samples in the case of digit recognition and comparing the results. For now we can only speculate on that however.

One important thing we omitted to do was evaluating the recognizers on data that had not been seen before. It is customary to split a data set into a 80% training set, 10% test set (used for performing the training cycles) and 10% evaluation set. This evaluation set needs to be used to check whether the recognizer has not been overtrained on the test data.

For a better validation of the feature classification performance, it is important to evaluate the performance for *all* visemes, or even phonemes if possible. The digit recognition task covers only 12 out of 16 visemes (omitting "h, shzj, pbm, aa"), while the alphabet recognition task only covers 13 out of 16 (omitting "at, shzj, a"). Taken all of them together, only the viseme "shzj" is missing, but that one rarely appears in the language anyway.

9.4 Discussion

Nearing the end of the descriptional part of this thesis, we need to evaluate to what extent this work has contributed to the field of speech recognition. The experimental work includes the following:

- Recording a new, extensive, audio-visual speech data corpus, containing high-speed footage of both front and profile view of the speakers. 70 people were recorded, 90 sessions were taken. All data had to be checked and processed.
- A contribution has been made to the old DUTAVSC corpus, transcribing 3 out of 8 recording sessions.
- An attempt was made to integrate the source code of HVite into a C++ project, but this was stopped after the focus of our project shifted from making a real-time implementation to documenting the new data corpus.
- Using the method of Active Appearance Models to perform lip tracking. To train a model (manually) annotated video frames are needed. For 8 people models were trained. The capabilities of this method were investigated.
- Extracting features based on the lip model point coordinates, and evaluating the performance of these features.
- A lip reader was trained, and results of several simple experiments were evaluated.

The previous paragraph was mainly focused on why the letter recognizer performed poorly, while it is perhaps more interesting to think about how the performance of the recognizer that was getting good results, the digit recognizer, could be improved.

With respect to the HMM architecture the following can be said. All along, we have been talking about using visemes to perform automatic lip reading. Using visemes is a good idea if the objective is to make a versatile speech recognizer that uses viseme models and searches the dictionary for the viseme representation of certain words. For simple tasks, like digit recognition, an alternative approach could be to train word-level HMMs. Here a larger HMM architecture can be used than the typical three-state HMM that is used for phones, with, for example, five states. This way one could get around the viseme mapping. This might be a good option, because there is no general agreement on the visemes set to use.

Also, while we were working with visemes anyway, it would have been perfectly possible to train the recognizer on all the data. It is a generally good idea to add training data, and this way no new recordings are needed.

We think we can answer part of the research questions now:

1. ***Is it possible to build an automatic lip reader comparable to or even better than a human lip reader?***

   We don't know. In the literature we have seen one small speech corpus (TULIPS1) which provides the recognition performance for humans without lip reading

experience and humans with lip reading experience. Using this data corpus to test a recognizer could answer the question.

2. ***Is it possible to build an automatic lip reader that performs as well as an acoustic speech recognizer?***

Most probably not. Using a phoneme to viseme mapping some speech information is lost.

3. ***In which way should we integrate the results of automatic lip reading and acoustic speech recognition?***

This lies outside the scope of this thesis.

4. ***Can we make an automatic lip reader that performs real-time?***

Not yet. Even though Active Appearance Model search is very fast, it still takes longer to process than the length of the utterance. Now that we use a frame rate that is five times higher the recognizer has to be four times as fast to reach real time.

5. ***What are the quantitative and qualitative requirements of the data we use to train an automatic lip reader?***

We believe we need high-speed recordings of a decent resolution in the mouth area.

6. ***Can the methodology to train an acoustic speech recognizer be directly applied to train an automatic lip reader?***

We have done so and the results seem decent enough.

7. ***Which feature extraction method should be chosen as the standard for automatic lip reading in general?***

We haven't encountered a single method yet that is robust, fast and speaker-independent.

10 Conclusions and recommendations

In this final chapter we will discuss and evaluate this project and give recommendations for further research. For each of the project goals defined in our problem definition, we will first discuss the results and draw our conclusions (section 10.1). In section 10.2 we will discuss some ideas that this work has brought forth, but of which the realization fell outside the scope of this thesis.

10.1 Conclusions

1. Exploring the potential of a lip reading system based on Hidden Markov Models

After going through some literature, especially the previous PhD work of Jacek Wojdeł and the manual of the Hidden Markov Model Toolkit, we saw that Hidden Markov Models are very flexible and can be used to model any kind of speech. After a phoneme to viseme mapping is applied and features have been extracted, an automatic lip reader can be treated as any other speech recognizer. There are however some limiting factors that are discussed in chapter 4.

2. Exploring the possibilities for real-time visual speech recognition

The bottleneck in most visual speech recognition systems is the feature extraction. Active Appearance Models has the property that they can perform face tracking very fast. Since we are working with high-speed recordings, to reach a real-time performance it would have to be four times as fast as for regular video though. Once all elements of visual speech recognition have been lined up to process live input, we expect satisfactory results. We have not yet measured the speed of the algorithms used. This way the Real Time Factor could be calculated.

3. Evaluating feature extraction methods discussed in literature according to the criteria of performance, speed and speaker independence

Although we wanted to come up with a fast *and* speaker-invariant way to extract features, this proved to be harder than expected. First of all, the initialization method of the Active Appearance Models is sensitive to lighting conditions, making it necessary to train a new model for every recording session, for which hand-annotated material is required. This makes any kind of live application impossible. Furthermore, the AAM search returns *point coordinates*. While we managed to compensate for any rotation, translation and scaling, the resulting features are speaker-*dependent* because of facial differences between people. A way to deal with this could be to include an adaptation phase in applications.

4. Obtaining a visual speech corpus that is sufficient in size and quality to train and evaluate an automatic lip reader from scratch

The purpose of this project was to develop a visual speech recognizer for Dutch. The success of this depended greatly on the available amount of training data. Experiments with the small audio-visual data corpus DUTCVSC pointed out the need for a new and larger data corpus. Our efforts produced high-speed video material of 70 different speakers recorded from the front and side. Because of expected technical difficulties with a speaker-independent system, we made some additional recordings of one person uttering just letters, digits and natural sentences, and used these to train the automatic lip reader. The new data corpus increased our expectations beyond the state of the art.

5. **Preparing and implementing the separate parts that make up an automatic lip reader and linking them together**

We identified four steps that were all prerequisites for building an automatic lip reading; data acquisition, lip tracking, feature extraction and training. The results for each of these steps had to be evaluated to ensure the quality of the final result. For the recordings the performance of the speakers and hardware were evaluated. For Active Appearance Models the accuracy was evaluated, for the features their classification performance was evaluated and the training was validated by examining the recognition results.

6. **Evaluating the results obtained from experiments conducted using the implemented lip reader**

After implementing the lip reader, we obtained a result of 78.1% correct and a word recognition rate of 68.5%, for the simplest task: digit recognition. For other tasks the recognition performance stayed behind. Further experimentation will have to shed some light on the exact reason behind this.

10.2 Future work

Looking back on this project, we have seen a number of topics pass the revue. Only the surface of what could be possible in automatic lip reading has been scratched however. In this section several ideas for future work will be addressed.

First of all, because of the limitations there are to automatic lip reading, we agree to the general idea that using lip reading in a stand-alone application would not lead to a satisfactory performance. Future research on visual speech recognition should therefore be focused on finding the right feature extraction method, model and training corpus. For real applications, combining modalities into an audio-visual recognizer has much more potential.

We still don't know exactly how much training data is required before a lip reader is sufficiently trained. After a certain amount of training data has been used the performance should stop to increase and eventually converge. At this time, new recordings are being processed to investigate when this point is reached. Another approach could be to decrease the amount of training data used gradually and display all results in a graph. The only downside to this approach is that training a speech recognizer is a time consuming process. If it turns out that any of the viseme models are undertrained, this would suggest that either more training data is required, or that the viseme classification needs to be revised.

A number of ideas regarding recognition experiments have been posed in this thesis. New experiments will have to be conducted to evaluate these ideas. Work is expected to continue in this respect. The visual speech recognizer performance will also have to be evaluated for other types of utterances, like natural sentences, for which bigrams would have to be used for the language model.

Bibliography

[1] "2006 FIFA World Cup Final," August 2, 2009; http://en.wikipedia.org/wiki/2006_FIFA_World_Cup_Final.

[2] S. Young, G. Evermann, M. Gales *et al.*, *The HTK Book (for HTK Version 3.4)*: Cambridge University Press, 2006.

[3] J. C. Wojdeł, "Automatic lipreading in the Dutch language," Delft University of Technology, 2003.

[4] T. Starner, and A. Pentland, "Real-time American Sign Language recognition from video using Hidden Markov Models," *Computational Imaging and Vision,* vol. 9, pp. 227-244, 1997.

[5] P. Wiggers, "Hidden Markov Models for Automatic Speech Recognition and their Multimodal Applications," Delft University of Technology, 2001.

[6] T. I. Boogaart, L. Bos, and L. Boves, "Use of the Dutch POLYPHONE corpus for application development." pp. 145-148.

[7] H. Yashwanth, H. Mahendrakar, and S. David, "Automatic speech recognition using audio visual cues." pp. 166-169.

[8] A. G. Chiţu, and L. J. M. Rothkrantz, "On Dual View Lipreading Using High Speed Camera," 2008.

[9] J. C. Wojdeł, P. Wiggers, and L. J. M. Rothkrantz, "An audio-visual corpus for multimodal speech recognition in Dutch language."

[10] M. Damhuis, T. Boogaart, C. Veld *et al.*, "Creation and analysis of the Dutch Polyphone corpus."

[11] "Persistence of vision," August 4, 2009; http://en.wikipedia.org/wiki/Persistence_of_vision.

[12] F. Wilson, and P. T. Descamps, "Should we accept anything less than TV quality: visual communication," *IEEE Conference Publications,* vol. 1996, no. CP428, pp. 606-611, 1996.

[13] M. d. Boo, "De automaat leest uw lippen (multimodale spraakherkenning)," *Delft Integraal*, Delft University of Technology, 2002.

[14] A. G. Chiţu, and L. J. M. Rothkrantz, "The Influence of Video Sampling Rate on Lipreading Performance." pp. 6-7452.

[15] K. Kumar, T. Chen, and R. Stern, "Profile view lip reading."

[16] T. Yoshinaga, S. Tamura, K. Iwano *et al.*, "Audio-visual speech recognition using lip movement extracted from side-face images."

[17] J. Luettin, N. A. Thacker, and S. W. Beet, "Visual speech recognition using active shape models and hidden Markov models." pp. 817-820 vol. 2.

[18] B. Lucas, and T. Kanade, "An iterative image registration technique with an application to stereo vision." pp. 674–679.

[19] J. C. Wojdeł, and L. J. M. Rothkrantz, "Using aerial and geometric features in automatic lip-reading."

[20] T. F. Cootes. "Modelling and Search Software," 2009; http://personalpages.manchester.ac.uk/staff/timothy.f.cootes/software/am_tools_doc/index.html.

[21] R. C. v. Dalen, "Lexical Stress in Speech Recognition," Delft University of Technology, 2005.

[22] D. Jurafsky, and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*: Prentice Hall PTR, 2000.

[23] N. v. Son, T. M. I. Huiskamp, A. J. Bosman *et al.*, "Viseme classifications of Dutch consonants and vowels," *The Journal of the Acoustical Society of America,* vol. 96, no. 3, pp. 1341-1355, 1994.

[24] M. Visser, M. Poel, and A. Nijholt, "Classifying Visemes for Automatic Lipreading," *Text, Speech and Dialogue*, pp. 843-843, 1999.
[25] A. G. Chiţu, and L. J. M. Rothkrantz, "Dutch Multimodal Corpus for Speech Recognition." p. 56.
[26] T. F. Cootes, G. J. Edwards, and C. J. Taylor, "Active appearance models," *Pattern Analysis and Machine Intelligence, IEEE Transactions on,* vol. 23, no. 6, pp. 681-685, 2001.
[27] M. Simunek, "Visualization of talking human head," 2009, http://www.cg.tuwien.ac.at/hostings/cescg/CESCG-2001/MSimunek/index.html, [2009, 2001].
[28] G. H. Landis, and C. Buckley, "Surgical Repair of the Cleft Lip and Palate," http://www.drlandis.com/pro/clsurg.htm, 2001].
[29] R. Gross, I. Matthews, and S. Baker, "Generic vs. person specific active appearance models," *Image and Vision Computing,* vol. 23, no. 12, pp. 1080-1093, 2005.
[30] P. Eilers, and B. Marx, "Flexible smoothing with B-splines and penalties," *Statistical Science,* vol. 11, no. 2, pp. 89-101, 1996.
[31] P. Wiggers, and L. J. M. Rothkrantz, "Integration of Speech Recognition and Automatic Lip-Reading," in Proceedings of the 5th International Conference on Text, Speech and Dialogue, 2002.

A Written instructions for New DUTAVSC recordings

Recordings instructions and progress

Thank you for accepting to join our experiment.

During this experiment you will be asked to utter several language items. The items to be uttered are going to be presented to you via a prompter like software. The program will display the next item to be recorded and instruct you on the modality to utter the current item. Please read these instructions carefully. When you are ready to start the recording please press the <LEFT MOUSE BUTTON>. When finished uttering please press AGAIN the <LEFT MOUSE BUTTON> to stop the recording. During the recording it is very important to follow the instructions given. For the success of the experiment and the research work that will use the resulted recordings it is very important to have a good posture on the whole period of recording. Hence, while uttering please keep your back, neck and head straight at all times and look straight into the camera. Try as much as possible not to move while recording. We expect that the total length of a recording to be maximum 10-15 seconds. If for some reasons the last recording was flawed you can retake the recording by pressing <LEFT MOUSE BUTTON> when ready. To go to the next item press <RIGHT MOUSE BUTTON>. Sometimes you will be asked to retake an item because the memory is full. When for some reasons you want to retake a previous item then using the <MOUSE WHEEL> is possible to go back.

Before starting the recordings please fill in the slots at the end of this document. After reading this document you can have a small trial, to accommodate with the software and the recording conditions. To start the recordings you need to register as a user. The experimenter will assist you in this process. He/She will then provide you with the print out of the items that are going to be presented to you. Please read the items. When you are ready you can start the experiment. The detailed expected timing of the experiment is given in Table 1.

Table A.1. The experiment timings

Timing	Description
00:00	Read the software manual.
-----	Start a trial session.
00:05	Register the user.
00:07	Print the pool of utterances for the user.
00:10	Ask the user to familiarize with the utterances.
00:15	Start the recordings
00:30	End the session.

Total time of the experiment is thus approximately 30 minutes

Please answer the following questions before going to the next step.

1. You were presented a **consent document** by the experimenter. Did you read and understand everything that was said in that document? Please say **yes** or **no** here: _____. Did you sign that document? Please say **yes** or **no** here:_____.
2. Did you read and understand the present document? Please say **yes** or **no** here:_____.

Before going further please take a trial with the software.

Thank you very much for agreeing to participate in our exercise. We hope that you will enjoy working with us, and hope you could come again for a follow up.
After the experiment is over the experimenter will give you some goodies to show our appreciation.

Please write your name here:_____ Date:_____

B Consent document for New DUTAVSC recordings

Consent document

Thank you very much for accepting to participate in our experiment.

This experiment consists of audio-video recordings of speakers uttering a set of items in Dutch. The data resulted is compiled in a database that will be used for scientific research purposes, namely training and testing different systems in the domain of audio-visual speech recognition, affective state recognition, speaker identification, etc. The resulted systems and analyses are going to be presented in scientific papers and public presentations, or used for demo during scientific events. We might also make the database available to other researchers. We might sometimes need to prove the database in some of the papers, presentations and/or demos, which means that some video or audio frames are going to be presented. During the experiment we will record your voice and the frontal and the side view of your head. Only the mouth area will be visible so the anonymity is guaranteed. We also guaranty that your name will never appear in public.

Please fill in this form and answer the questions at the end. We are very thankful for you help.

Name: _____
Sex: _____
Age: _____
Level of education : _____
Native language(s): _____
Country and province of origin: _____

Please answer the following questions:

Do you agree that the recordings that feature **you** to be used for scientific research as explained above? Please answer **yes** or **no** here: ____.

Do you agree that some images showing **your** mouth to be included in scientific papers, public presentations and/or demos? Please answer **yes** or **no** here: ____.

Do you agree that complete or partial audio clips that feature **your** voice to be included in public presentations and/or demos? Please answer **yes** or **no** here: ____.

Have you read and understood this document? Please answer **yes** or **no** here: ____.

Please sign the document and hand it to the operator. Thank you for your co-operation and we hope we can count on you in the future.

Date: _____ Signature: _____

C Lip feature extracting algorithm

```
//LipFeatureExtracter computes the feature vectors from a set of model point coordinates
class LipFeatureExtracter{

	public static String report = "";

	public static void main(String[] args){
		File current = new File (".");

		try {
			searchDir(current, "");

			PrintWriter out = new PrintWriter(
				new BufferedWriter(
					new FileWriter("nose_width.txt", false)));
			out.println(report);
			out.close();

		}
		catch(IOException e) {
			e.printStackTrace();
		}
	}
	//substract point2 from point1
	public static double xDistance(double[] point1, double[] point2){
		return point1[0] - point2[0];
	}
	//substract point2 from point1, screen coordinates!
	public static double yDistance(double[] point1, double[] point2){
		return point2[1] - point1[1];
	}
	//Pythagoras
	public static double distance(double[] point1, double[] point2){
		return Math.sqrt(Math.pow(point1[0] - point2[0],2) +
			    Math.pow(point1[1] - point2[1],2));
	}
	public static double[] middle(double[] point1, double[] point2){
		double[] array = new double[2];
		array[0] = Math.abs(point1[0] - point2[0]);
		array[1] = Math.abs(point1[1] - point2[1]);
		return array;
	}
	public static int nextIndex(int index, int size, int increment){
		int ans = index + increment;
		if (ans > size - 1)
			ans = 0;
		if (ans < 0)
			ans = size - 1;
		return ans;
	}
	//pre: point 0 and point length/2 are the mouth corners, only even # points allowed
	//post: returns minimum width of a polygon (inner mouth width)
	public static double polygonWidth(double[][] points, double epsilon){

		if (points.length % 2 != 0)
			return 0;
		int left = points.length/2, right = 0, i=1;
		double distance;
		while (left == points.length/2 && i < points.length/2){
			distance = distance(points[i], points[points.length-i]);
			if (distance > epsilon)
				left = i-1;
			i++;
		}
		i = points.length/2-1;
		while (right == 0 && i > 0){
			distance = distance(points[i], points[points.length-i]);
			if (distance > epsilon)
```

```
                        right = i+1;
                i--;
        }
        if (xDistance(points[left], points[right])>=0)
                return 0;
        return distance(points[left], points[right]);
}
//pre: point 0 and point length/2 are the mouth corners, only even # points allowed
//post: returns maximum height between point pairs (inner mouth height)
public static double polygonHeight(double[][] points){

        if (points.length % 2 != 0)
                return 0;
        double max = 0, testValue;
        for (int i = 1; i < points.length/2; i++){
                testValue = yDistance(points[i], points[points.length-i]);
                if (testValue > max)
                        max = testValue;
        }
        return max;
}
// pre: points form a closed path
// post: returns polygon area computed using trapezium method
public static double polygonArea(double[][] points){

        double minY = points[0][1];
        double[] minX = points[0], maxX = points[0];
        int index = 0, nextIndex = 0, incr = 1;
        for (int i = 0; i<points.length;i++){
                if (points[i][1] < minY)
                        minY = points[i][1];
                if (points[i][0] < minX[0]){
                        minX = points[i];
                        index = i;
                }
                if (points[i][0] > maxX[0])
                        maxX = points[i];
        }
        // could also just take the diff between area1 and area2,
        // but this way you can have a negative result for self-intersection
        nextIndex = nextIndex(index, points.length, 1);
        if (points[nextIndex(index, points.length, -1)][1] > points[nextIndex][1])
                incr = -1;

        double area1 = 0, area2 = 0;
        //A(trapezium) = h ((a+b)/2)
        while (points[index][0] < maxX[0]){
               nextIndex = nextIndex(index, points.length, incr);
               area1 = area1 + Math.abs(points[index][0] - points[nextIndex][0])*
               ((points[index][1] + points[nextIndex][1] - 2 * minY)/2.0);
               index = nextIndex;
        }
        while (points[index][0] > minX[0]){
               nextIndex = nextIndex(index, points.length, incr);
               area2 = area2 + Math.abs(points[index][0] - points[nextIndex][0])*
                       ((points[index][1] + points[nextIndex][1] - 2 * minY)/2.0);
                index = nextIndex;
        }
        //the area of self-intersection is subtracted instead of added:
        //A(trapezia top) - A(trapezia bottom)
        return area1 - area2;
}
//pre: features contains the features to be written
//post: the feature vector is written to a file
public static void writeFeatures(String dirname, float[][] features) throws IOException{
        if (features.length > 0){
                dirname = dirname.replace("_","");
                DataOutputStream dos = new DataOutputStream(
                        new FileOutputStream(dirname+".dat", false));
                //nSamples (4-byte int)
                dos.writeInt(features.length);
```

```
                //sampPeriod (4-byte int; 10 ms sample rate in 100 ns units)
                dos.writeInt(100000);
                //sampSize (2-byte int; #bytes per sample)
                dos.writeShort(4 * 5);
                //parmKind (2-byte int; code indicating sample kind)
                // + 4 * 8*8 + 8*8*8);
                //user defined + delta + acceleration USER_D_A
                dos.writeShort(9);

                PrintWriter out = new PrintWriter(
                        new BufferedWriter(
                                new FileWriter(dirname+".txt", false)));
                //check array contents
                for (int i = 0; i < features.length; i++){
                        out.print(i + "\t");
                        for (int j = 0; j < features[0].length; j++){
                                out.print(features[i][j] + "\t");
                                        //write to HTK file
                                dos.writeFloat(features[i][j]);
                        }
                        out.println();
                }
                out.close();
        }
}
//post: searches file system recursively for ASF files containing AAM coordinates
public static void searchDir(File f, String name) throws IOException{

        File[] dirs = f.listFiles(new DirFilter());
        for (int i = 0; i<dirs.length;i++){
                if (         dirs[i].getName().startsWith("T") |
                   dirs[i].getName().equals("Frontal")|
                   dirs[i].getName().equals("Side"))
                        searchDir(dirs[i],name);
                else if (name.equals(""))
                        searchDir(dirs[i], dirs[i].getName());
                else
                        searchDir(dirs[i], name + "_" + dirs[i].getName());
        }
        System.out.println("Processing: " + f.getName());

        File[] files = f.listFiles(new ASFFilter());
        //sort files by filename
        Arrays.sort(files);
        float[][] features = new float[files.length][7];
        double mean = 0, variance = 0;
        double[] scalings = new double[files.length];

        for (int i = 0; i < files.length; i++){

                String fullFilename = f.getCanonicalPath()+ "\\"+files[i].getName();

                BufferedReader in = new BufferedReader(
                        new FileReader(fullFilename));

                String line = in.readLine();
                int numCount = 0, x = -1, y=0;
                int[] path = new int[2];

                double[][] coordinates = new double[1][2];
                StreamTokenizer st = new StreamTokenizer(in);
                //handle #points
                while (st.ttype != st.TT_EOF && numCount<5){
                        st.nextToken();
                        if (st.ttype == st.TT_NUMBER){
                                coordinates = new double[(int)st.nval][2];
                                numCount++;
                        }
                }
                while (st.ttype != st.TT_EOF){ //read coordinates
                        st.nextToken();
```

```
                        if (st.ttype == st.TT_NUMBER){
                            //you encounter a new row
                            if (numCount%10 == 5){
                                x++;
                            //store # points for first 2 paths (lip contours)
                                if ((int)st.nval < 2)
                                  path[(int)st.nval] = path[(int)st.nval]+1;
                            }else if (x > -1 && x < coordinates.length){
                                if (numCount%10 == 7)
                                    coordinates[x][0] = st.nval * 384.0
                                else if (numCount%10 == 8)
                                    coordinates[x][1] = st.nval * 288.0;
                            }
                            numCount++;
                        }
                    }
                    in.close();

                    double[][] outer = new double[path[0]][2];
                    double[][] inner = new double[path[1]][2];
                    for (int j = 0; j < path[0]; j++){
                        outer[j] = coordinates[j];
                    }
                    for (int j = path[0]; j < path[0]+path[1]; j++){
                        inner[j-path[0]] = coordinates[j];
                    }

                    //scaling factor for camera distance normalization
                    double scaling; //nose points distance
                    if (coordinates.length == 25) //old model
                        scaling = distance(coordinates[19],coordinates[24]);
                    else //new model
                        scaling = distance(coordinates[24],coordinates[25]);
                    mean += scaling/(double)features.length;
                    scalings[i] = scaling;

                    //feature 1: outer lip width
                    features[i][0] = (float)(distance(outer[0], outer[outer.length/2])/scaling);
                    //feature 2: outer lip height
                    features[i][1] = (float)(distance(outer[outer.length/4],
                            outer[3*outer.length/4])/scaling);
                    //feature 3: inner lip width
                    //image 384x288, so epsilon = 1.92 & 1.44 px, .005 => .001
                    features[i][2] = (float)(polygonWidth(inner, 2.0)/scaling);
                    //feature 4: inner lip height
                    features[i][3] = (float)(polygonHeight(inner)/scaling);
                    //feature 5: nose/chin distance
                    if (coordinates.length == 25)
                        features[i][4] = (float)(distance(coordinates[17],
                        middle(coordinates[19],coordinates[24]))/scaling);
                    else //new model (29 points)
                        features[i][4] = (float)(distance(coordinates[27],
                        middle(coordinates[24],coordinates[25]))/scaling);
                    //feature 6: polygon area of outer lip shape
                    features[i][5] = (float)(polygonArea(outer)/(scaling*scaling));
                    //feature 7: polygon area of inner lip shape
                    features[i][6] = (float)(polygonArea(inner)/(scaling*scaling));
                }
                writeFeatures(name, features);
                //report about mean and var of the distance between nose points
                if (features.length > 0){
                    for (int i = 0; i < scalings.length; i++){
                       variance += ((scalings[i] - mean) * (scalings[i] - mean))
                       / (double) scalings.length;
                     }
                     report = report + name + " Mean: " + mean + " Variance: " + variance + "\n";
                }

            }
        }
```