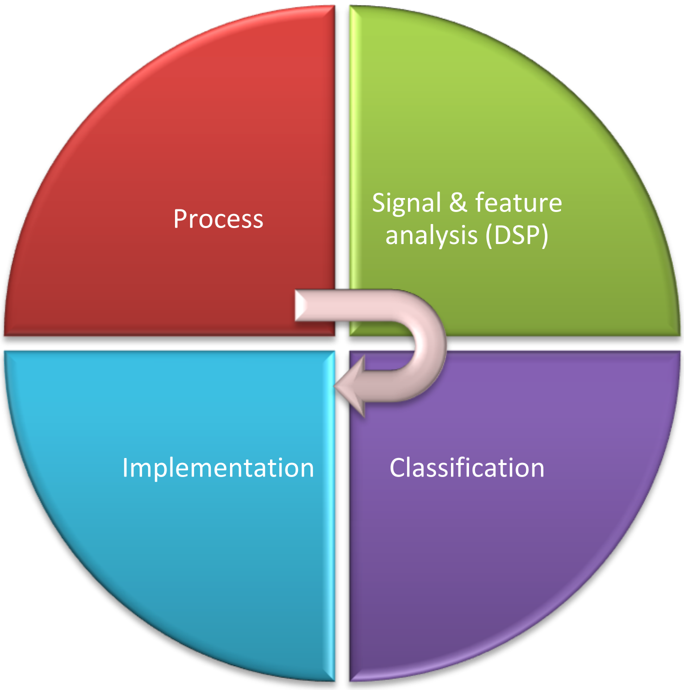
DTU Oktober 2007

Speaker identification

Master thesis



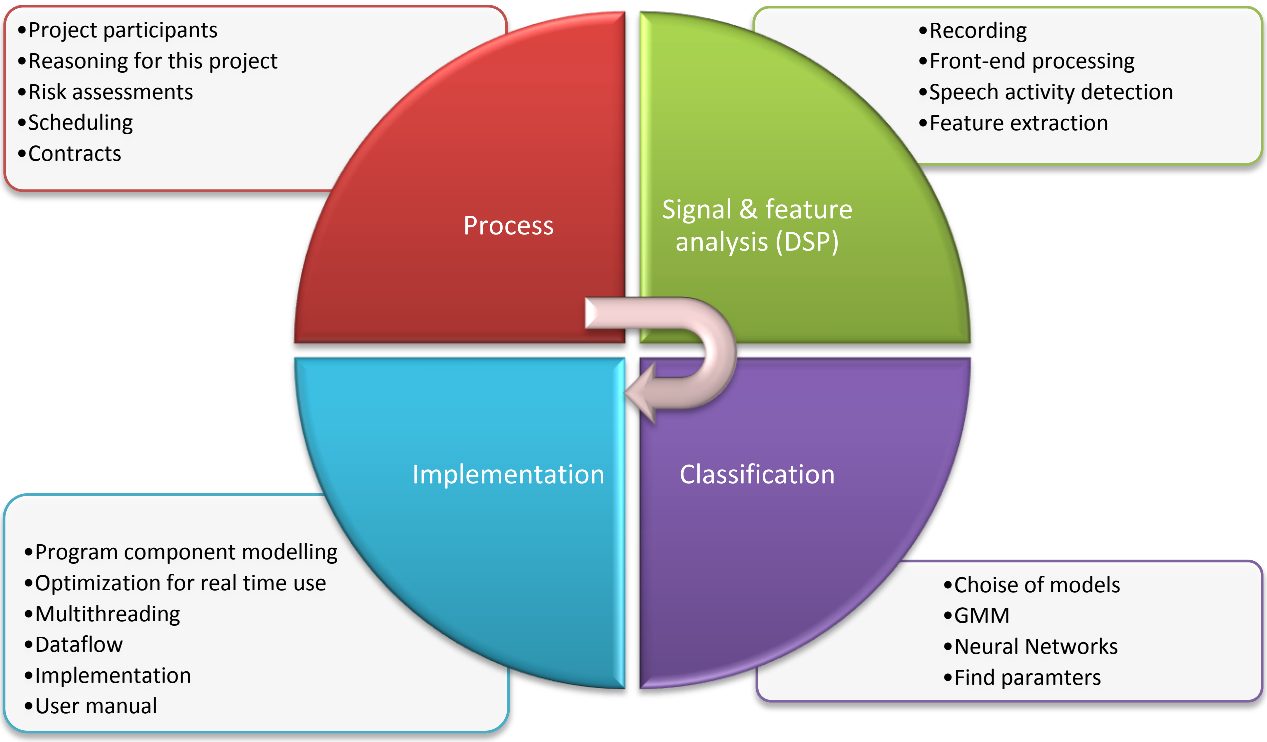
Supervisor
Niels-Ole Christensen

Authors

Dan Bakmand-Mikalski — Unsigned copy
____

Anders Havnsø Rasmussen — Unsigned copy
____

The reports



Process

- Project participants
- Reasoning for this project
- Risk assessments
- Scheduling
- Contracts

Signal & feature analysis (DSP)

- Recording
- Front-end processing
- Speech activity detection
- Feature extraction

Classification

- Choise of models
- GMM
- Neural Networks
- Find paramters

Implementation

- Program component modelling
- Optimization for real time use
- Multithreading
- Dataflow
- Implementation
- User manual

Other documents included

Appendix A	•Contract
Appendix B	•Milestones
Appendix C	•Schedule
Appendix D1	•DC removal by running average filtering
Appendix D2	•DC removal extended results
Appendix E	•Example on how RMS based voice detection adapts to changing SNR
Appendix F	•Brief walkthrough of PCA

1) Preface

The scope of this master thesis is to discover and analyze problems involved in speaker identification and develop a robust and scalable solution to overcome these problems. The thesis is carried out at the institute of Informatics and Mathematical Modelling (IMM) at the Technical University of Denmark. Supervisor of this project is Lector Niels-Ole Christensen working at IMM.

The end-result is a Win32 based .NET 2.0 application for real-time speaker identification.

The master thesis contains four different reports each listed below:

- Process
- Signal & feature analysis
- Classification
- Implementation

The *Process* report documents teamwork, risk assessments, time schedules and developing strategies.

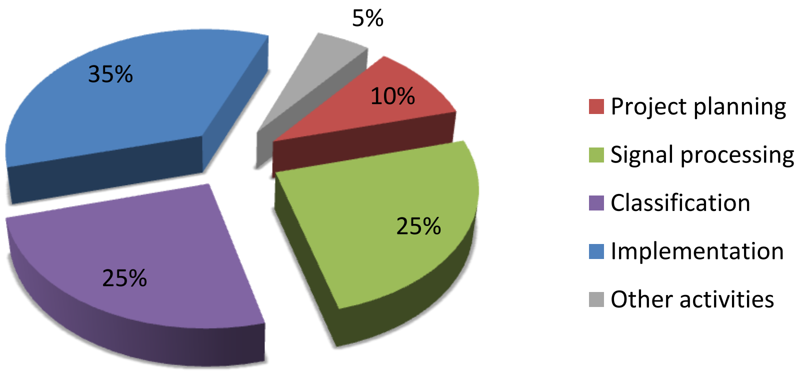
Signal & feature analysis covers documentation about front-end signal processing, voice activity detection and feature extraction.

The *Classification* report describes pattern recognition systems, implementation and performance of GMM and NN.

The *Implementation* report focuses on the implementation structure of the above features and classification systems in a real time Win32 application.

The report can be read in random order but it outlines the project phase better if read chronologically.

The figure below shows the overall work distribution of this project.



2) What is speaker identification?

Speaker identification is a branch of the biometrics tree. Here we show how it relates to this project.

1.1 Biometrics

Biometrics is the technique of studying physical and behavioural characteristics of human beings as illustrated in Figure 1. This is often used to model the human traits in computer systems.

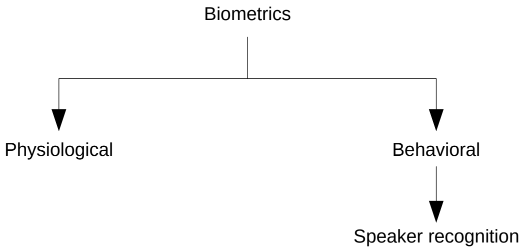


Figure 1 - Biometrics

Physiological

A physiological biometric is based on the shape of the human body. These systems uses prior knowledge of the human traits for classification. Face-, iris-, and DNA recognition systems can all be classified as physiological biometrics.

Behavioral

A behavioural biometric is related to the behaviour of individual humans. Systems that can be classified as a behavioral biometric are e.g. speech- and signature recognition systems.

For instance speech recognition systems use information on how individual speakers pronounce different words.

Speaker recognition is classified as a behavioural biometric. Features that uniquely represent the characteristics of individual speakers' voices are estimated and used for classification.

1.2 The behavioural pattern: Speaker recognition

Speaker identification is the task of recognizing speakers based on their voice. Speech recognition is on the other hand the task of recognizing what is being said.

Speaker recognition can be divided into speaker verification and speaker identification. These can furthermore be divided into text dependent and text independent systems as illustrated in Figure 2.

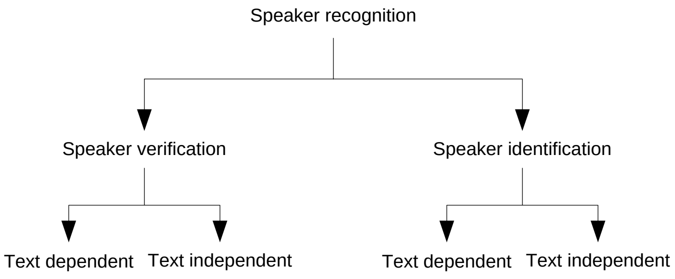


Figure 2 - Speaker recognition

Speaker identification

Speaker identification is used to decide whether an unknown speaker is a specific person or belongs to a given group of persons. This is done by comparing the speakers voice with a speaker database (1:N comparison). The database contains models of all known speakers. The unknown speaker is identified as the speaker from the database with the best match between the speech input and the database model.

This master thesis is about speaker identification.

Speaker verification

Speaker verification is another sub problem of speaker recognition. In speaker verification the system is verifying whether the speaker is whom he/she claims to be (1:1 comparison). One of the major challenges within these systems is trying to find a reliable threshold that can be used for decision making. A high threshold makes it difficult to get accepted by the system and may result in rejections of genuine persons. On the other hand a low threshold makes it easy to get accepted by the system with the risk of accepting imposters.

Speaker verification is often used in security access systems.

Text dependent vs. Text independent systems

As illustrated in Figure 2 speaker identification and verification systems can be divided into text dependent and text independent systems. Text dependent systems only make decisions on specific sentences. Text independent systems are more flexible and can make decisions on text independent sentences.

In this master thesis a text independent system is implemented.

Abstract

This master thesis focus on implementing a real time speaker identification system. Compared to other projects on this field the authors have focused on a more product orientated approach by giving the real time implementation pride of place.

The real time implementation not only motive the authors but also introduce several interesting problem areas. There is a clear distinction between using speech signals recorded under perfect conditions and signals recorded in areas containing ambient noise when implementing a speaker recognition system.

Front-end signal processing have been used to remove the DC value, noise and silence from the signals. This area have been a major challenge and an important factor in achieving high recognition rates. Ignoring these factors not only decreases the recognition rate but also increase the time used for classification as e.g. silence will be classified.

Using front-end processing have lead to better conditions for the feature extraction methods. Mel Frequency Cepstral Coefficients (MFCC) is the most commonly used feature in speaker identification systems and have showed to model the human voice more closely than any other method. Features used in this master thesis are MFCC, dMFCC in time and the pitch period. These features have through test showed to be robust and ideal for the real time speaker identification system.

Gaussian Mixture Models and Neural Networks is used as classification systems. It turns out that both classification systems generate high recognition rates based on speech signals recorded under perfect circumstances. No major differences in recognition rates, training time or the time used per classification between these systems have been noticed.

Due to the fact that the classification systems performs almost equally both have been implemented in the final application. The final application have been implemented in C# and resulted in recognition rates of 95 to 100 percent on signal recorded under perfect conditions. Using speech signals containing ambient noise the recognition rate decreases but still the classifications systems performs with a recognition rate above 90 percent.

Master thesis project formulation

Analysis of speaker dependent features. Examination and implementation of a relevant speech classification problem using these features.

Conclusion

This master thesis have focused on implementing a real time speaker identification system for the Win32 platform.

This thesis have been organised in four different reports describing *Proces*, *Signal & feature analysis, Classification* and *Implementation*.

The *Proces* report described time schedules, risk assessment and developing strategies. These were seen as guidelines to keep the project within the proposed time frame. It has been of most importance to keep to schedule as a course in “Network and Integer Programming” have been attended parallel with this project period (a possible delay factor). The most significant event is the time used on implementation far succeeded the planned time.

The *Signal & feature analysis* report have been a major part of this master thesis. Different front-end signal processing methods was used like DC-component removal, speech enhancement and noise removal. These together ensured that the input signals were enhanced for voice activity detection. Finally speech features are analyzed and extracted.

Different features were analysed, implemented and tested in Matlab. It turned out that features derived by a cepstral analysis provides specific information about individual humans. Mel Frequency Cepstral Coefficients (MFCC), delta MFCC, delta-delta MFCC, cepstral liftering and the pitch period was features/methods resulting in the best test results using PCA.

In the *classification* report two different classification systems were implemented (Gaussian Mixture models (GMM) and Neural Networks (NN)). The systems are based on different approaches but turned out to perform almost equally. The ELSDSR database that provides speech sentences recorded under conditions with minimal ambient noise were use to test the classification systems. The systems were tested on 2-10 speakers (5 male and 5 female) to give an insight in how to configure the systems based on the number of speaker models contained in a speaker identification system.

The GMM achieved a 100 percent recognition rate on 2 – 8 speakers. Using 10 speaker models the classification system performed with a 99.23 percent recognition rate. The NN did almost achieve identical classification rates with less time used per classification. The best configuration of the NN resulted in a recognition rate of 100 percent for 2 - 10 speakers.

Delta MFCC were used to optimize the GMM. Using delta coefficients didn’t result in an increase in recognition rate but reduced the ms of speech needed per classification.

The *Implementation* report demonstrates the real time application programmed in C#. The application is able to demonstrate and plot different signal processing methods like noise removal, coefficient calculation and filtering on different input signals. The application not only uses the ELSDSR database for test but is able to record real time signals and use these for training and test. Using real time signals results in lower recognition rates with a very poor microphone. Using 2 – 6 speakers the GMM and NN still classify the data well above 90 percent.

Explanation of words

LPCC	Linear Prediction Cepstral Coefficients
MFCC	Mel Frequency Cepstral Coefficients
LPC	Linear Prediction Coefficients
GMM	Gaussian Mixture Model
NN	Neural Network
VQ	Vector Quantization
ELSDSR	English Language Speech Database for Speaker Recognition
FT	Fourier Transform
DFT	Discrete Fourier Transform
DCT	Discrete Cosine Transform
STFT	Short Time Fourier Transform
VAD	Voice Activity Detection
RMS	Root Mean Square
PSD	Power Spectral Density

Literature

Title	Year	Authors	ISBN
Signal Processing First	2003	James H. McClellan Ronald W. Schafer Mark A. Yoder	0-13-120265-0
Neural Networks for Pattern Recognition	1995	Christopher M. Bishop	0-19-853864-2
Programming Microsoft Windows with C#	2002	Charles Petzold	0-7356-1370-2

Report:
Process

Figure list:

1) Introduction to Process

This report of the project has three major focus areas.

- The authors and handling of their strengths and weaknesses.
- Development strategies and risk handling.
- Scheduling

The goal of this project report is to achieve a safe and feasible foundation for the further progress of the project. This is accomplished by deploying a project framework of project strategies based on the authors strengths and scheduling. The result is believed to be project framework, specifically tailored for the individual parts of this project.

2) Project participants

To give the reader a sense of the main participants involved in this project, they are listed here.

2.1 Authors

2.1.1 Anders Havnsø Rasmussen

I am 28 years old and live in Stenløse. Originally I am educated at the IT-department at Tellabs Denmark A/S.

Due to a large interest within the IT area I decided to apply for admission as a computer scientist at the Business College of Ballerup. As a student at the Business College I made the acquaintance of Dan Bakmand-Mikalski.

When I graduated I decided to apply for admission at the Engineering College of Copenhagen. This education gave me a solid experience within signal processing, system development and programming. Dan Bakmand-Mikalski was one of the students that I made all major projects with.

In 2005 I applied for admission at the Technically University of Denmark, where I at the moment am doing my master thesis at the department of Informatics and Mathematically Modeling.

My spare time I spend on my family, friends and on different IT areas.

Figure 1 - Anders "the avenger" Havnsø Rasmussen

2.1.2 Dan Bakmand-Mikalski

I am 29 years old and live in Copenhagen. I often spend my spare time in Hedehusende at the house of my girlfriend.

I've always had a certain interest within the IT area. This was the main reason that I decided to apply for admission as a computer scientist at the Business College of Ballerup in 1999. Due to the enormous interest in different IT educations in the late 90'ties I was not admitted before 2001. In the meantime I decided to study and work as a social and health visitor.

The computer scientist education was a worth-while experience and together with a job as webmaster at BT I got a huge interest within the IT area.

In 2003 I decided to apply for admission at the Engineering College of Copenhagen, where areas like signal processing and programming where my main interests.

In 2005 I applied for admission at the Technically University of Denmark. The main focus at the Technically University of Denmark has been signal processing, image vision, A.I. and programming.

My spare time I spend on family, friends, Jujitsu and my be loving computer ☺

Figure 2 - Dan "the dork" Bakmand - Mikalski

2.2 The Authors as a group

This section describes the inter-relations of the authors.

2.2.1 Teamwork history

Both authors have previously completed the Computer scientist (Datamatiker) study together, without working together as a team though.

Some months later, both authors attended the IT-diploma program at the Engineering College of Copenhagen (Ingeniørhøjskolen i København). Because of the merits obtained from the Computer Science program, the authors joined a special course program with 11 other Computer Scientists. As we attended various courses with other IT-diploma students, it was beneficial for the Authors to work together as a team in relation to coordination of project work etc.

After graduating as IT-diploma engineers the Authors agreed to seek new challenges at Technical University of Denmark (DTU). Because of the sometimes overwhelming amount of project work, it was again extremely beneficial to work together as a team which, to some extent, eases the coordination and raises the gain from courses. At most courses, the team has been extended by 1 other fellow student if permissible.

2.2.1.1 Socially

The group has been extremely successful with regards to social working environment. Often new teams have to invest a lot of resources in building a common understanding of the participants' roles, strengths, weaknesses etc. This can of cause lead to conflicts and time waste. By working together in an already proven team, these risks are avoided. The Authors are positive that this has increased both academic gains and working morale. The occasional team-extension with other fellow students has been a welcome catalyst to avoid convergence, increase dynamics and generally evolve this team.

2.2.1.2 Expertise

The team has a reasonable balance in expertise regarding this project. As a whole, the team has strengths particularly in the areas of software analysis/design, programming skills, digital signal processing and various classification methods with focus on neural networks.

The major weakness of the team regarding this project is the lack of training in formulating mathematical methods and proofs. This is an offspring from the authors' line of education, where mathematics hasn't been such a large part of the obtained courses, as is the case with the "pure" DTU-student.

These strengths and weaknesses combine to a team with a good knowledge base for this project and at the same time present the team with new exiting challenges.

2.2.1.3 Roles

The co-author Dan Bakmand-Mikalski is mainly product oriented and to lesser extends analysis oriented, whereas co-author Anders Havnsø Rasmussen is more evenly balanced between product and analysis.

- **Benefits**
  The team have a clear benefit related to product oriented projects. As speaker identification from a practical approach is such a project involving several practical aspects such as making it work in real life situations this is considered a benefit. In general one can say that product oriented approaches promotes development, test and prototyping rather than in-depth analysis of problem domain.

- **Drawbacks**
  The major drawback and possible risk to the project is the authors combined weight on product orientation. This has to be taken into consideration and actions must be taken in both risk assessments and development strategies to avoid an unbalance between this project's analysis oriented goals and the authors overall product oriented tendencies.

2.3 Project supervisor

The supervisor of this project is Niels-Ole Christensen. He is currently operation as Lector on the IMM institute on DTU.

Both authors have previously attended a course in Neural Networks taught by Niels-Ole Christensen. The focus of that course was examination and understanding of different types of networks.

The course was concluded by a "large" project. The authors made a project in License plate recognition.

The authors are regarding Niels-Ole Christensen as being mainly product oriented, which means, that he has a large focus on converting theory into praxis. This is also evident from the form of this master thesis.

3) Reasoning for this project

3.1 The authors' reasons for choosing this project

The idea of this project is based on the teams desire to work with machine-learning and classification concepts combined with digital signal processing. Neural networks in particular are a huge area of interest of the authors.

We believe that this project of *Speaker identification* both utilizes our strengths and is combinable with our main interest areas while still maintaining a fair amount of challenges.

At an earlier state, other projects had been suggested, of which *Query by Humming*, was the runner up. It was abandoned due to input from our project supervisor and Professor Lars-Kai Hansen. Both stated that the scope of *Query by Humming* was too big for the scheduled project time-slot.

The authors expect this project to present difficult challenges and are aware that this project involves so many different aspects that it is a very time consuming project which both results in major risks.

The main aspects mentioned just above include:

- Complex signal processing to match real-life data.
- Analysis of speaker dependent features.
- Classification using Neural Networks and an alternative method e.g. GMM.
- A running prototype program demonstrating DSP and speaker identification at real-time.

4) Risk assessments

The risk assessment has been divided into separate parts to enhance overview. Furthermore the level of each risk is classified as Low, Medium or High. We need to take the Low level risks into account even though it is the Medium and High level risks that can have the most negative effect on this realization of this project.

4.1 Process

Risk type	Probability	Consequence	Handling
Illness	5 %	Low	If the period of the illness last more than a couple of days, the person will be assigned minor tasks to do at home.
Workgroup disagreements	10 %	Medium	Every idea that the participants of this project has for feature extraction, classification or implementation will be taken seriously and discussed e.g. with the supervisor.
Error in Time schedule estimates	15 %	High	Deadlines needs to be respected as this project is rather large in relation to the time period.
External activities takes too much time	30 %	High	As we need 10 ECTS points (course Networks and Inter Programming) besides this master thesis to graduate we need to respect all deadlines and if time problems occur downgrade the external course.
Goals not feasible	5 %	High	As speaker identification is a known area of speaker recognition we will analyze different methods to achieve out goals. If problems occur in the project period the supervisor will be contacted.
Unbalanced weighing between analysis and product phases	20 %	Medium	As mentioned earlier this project group is very product orientated. Even though it is important that we first of all analyze different speaker identification systems and features before we start implementing the program.

4.2 Features and Classification results

Risk type	Probability	Consequence	Handling
No relevant and robust features can be found	10%	High	Examine results from external sources. Analyze the human speech mechanics. Analyze a wide range of features.
Classification yields non-deterministic results	5%	High	Examine a broad variety of classification and preprocessing methods based on different strategies.
Data for analysis inadequate	15%	High	Use the ELSDSR database for analysis purposes. It is strictly documented and made under controlled conditions.
Problem complexity yields infeasible long processing time.	20%	Medium	Create prototypes of central components as early as possible. Examine real-time perspectives of training optimized Neural Networks on large datasets. Enhance programming skills toward code optimization. Analyze Big O problem sizes of relevant components.

4.3 Software

Risk type	Probability	Consequence	Handling
Realization problems due to lack of skills	5 %	High	Problems doing with the realization of the project will be discussed with the supervisor, students at DTU etc.
Software stability issues	10 %	Medium	The ELSDSR can be used for test and a high performance external microphone will be purchased doing the project period.
Software complexity too large to handle by 2 Authors	10 %	Medium	It is important that limitations of the project are written down. This way we ensure that the goal is reachable. Furthermore the limitations have the effect that the complexity of the project won't increase during the project period.
Real-time not achievable	15 %	High	Other projects in speaker identification are analyzed before we start feature extraction, classification and implementation. This way we know exactly which methods works and which won't.

5) Development strategies

Due to the difference of the various tasks involved in this project it has been decided to utilize different development strategies for each main area of this project.

It has also been decided to divide this project into 4 reports each covering a different phase of the project. They are however linked together as they together forms this project. The reports are:

- Process
- Signal & feature analysis
- Classification
- Implementation

5.1 Process

In this phase we look into the background of this project, the participants, the risks and a set of milestones of the project.

Combining these with the strategies and the goals, we can create a schedule (as in Figure 3) to help keep focus on the most important parts of the project in case the planned schedule should slip (which it almost always do when software development is involved).

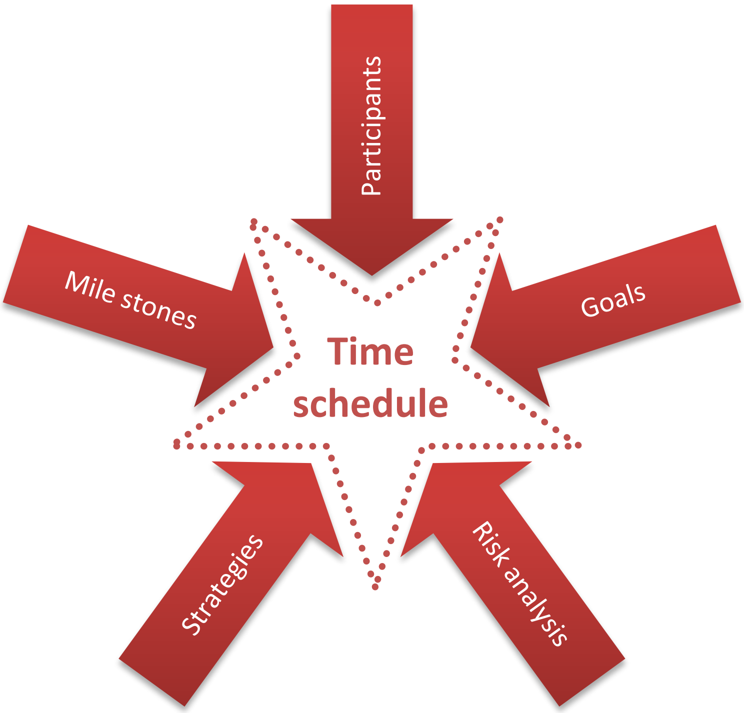
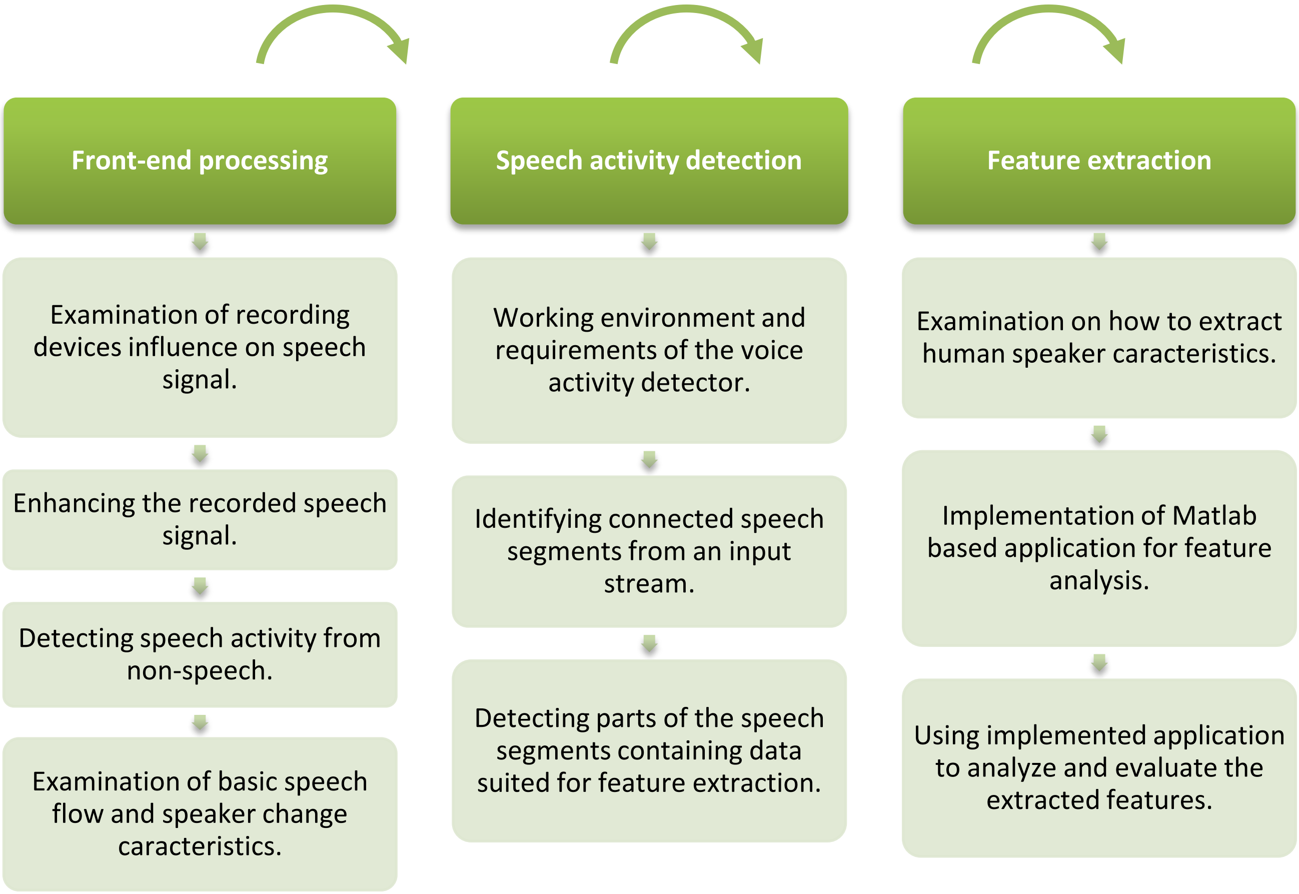


Figure 3 – Strategy overview of process

5.2 Signal & feature analysis (DSP)

In this phase it has been chosen to use an experimental based approach with chronologically ordered activity schemes as in Figure 4. This is because the authors don't quite know what results to expect from each of the 3 activity schemes proposed.

The actual setup in the *Signal & feature analysis* report doesn't necessary reflect this setup but will however include all activities.



Front-end processing

Examination of recording devices influence on speech signal.

Enhancing the recorded speech signal.

Detecting speech activity from non-speech.

Examination of basic speech flow and speaker change caracteristics.

Speech activity detection

Working environment and requirements of the voice activity detector.

Identifying connected speech segments from an input stream.

Detecting parts of the speech segments containing data suited for feature extraction.

Feature extraction

Examination on how to extract human speaker caracteristics.

Implementation of Matlab based application for feature analysis.

Using implemented application to analyze and evaluate the extracted features.

Figure 4 - Strategy overview of Signal & feature analysis

5.3 Classification

In this phase the types of models used to store and classify features as belonging to a specific speaker is well-known from both experience and literature. Thus the focus will be on selecting models with suitable capabilities. The selected models are then expanded and modified to meet our needs and evaluated in an iterative scheme as in Figure 5. This means that it is more like a test →modify/expand → retest scenario.

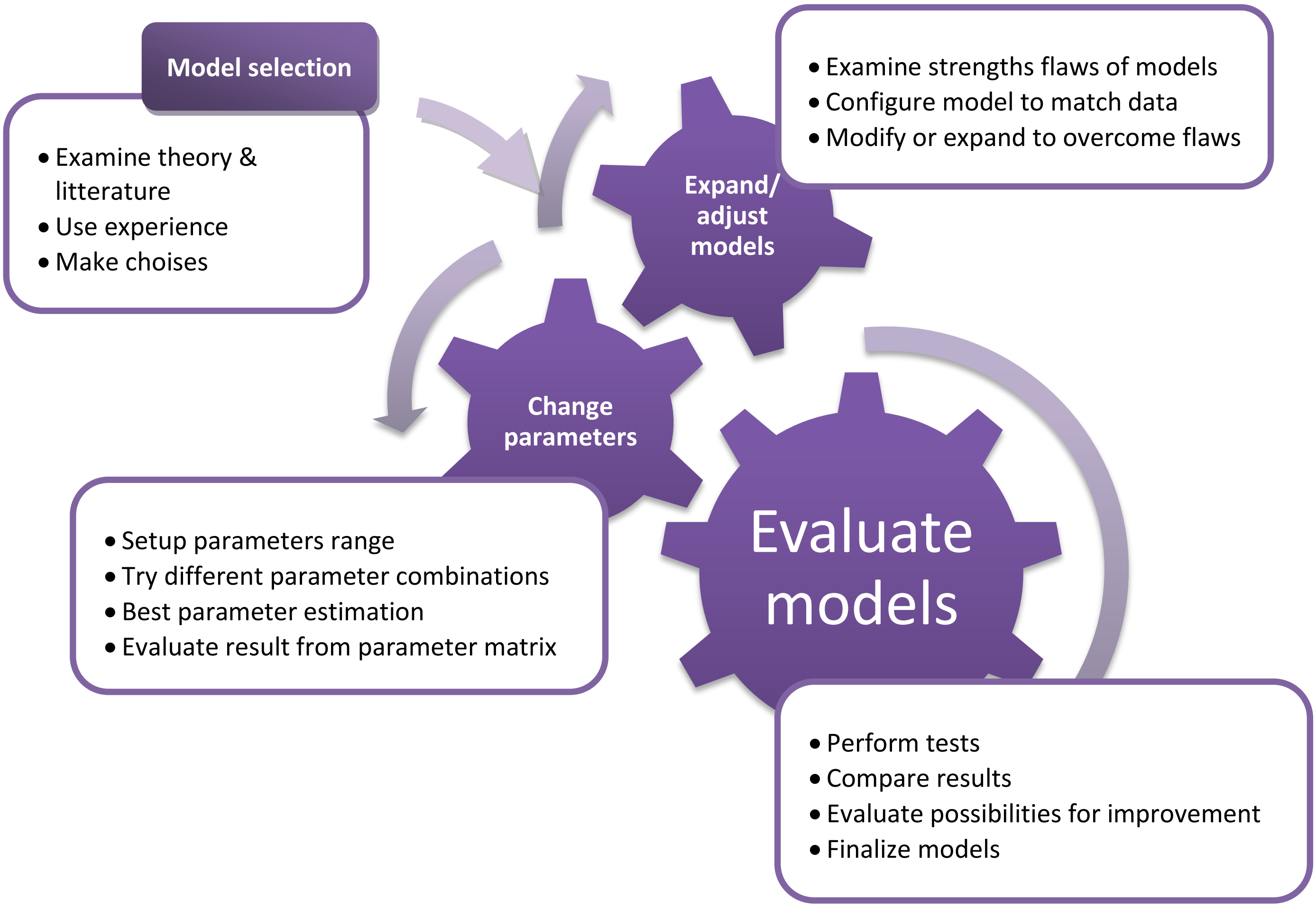

Model selection
• Examine theory & litterature
• Use experience
• Make choises
Expand/ adjust models
• Examine strengths flaws of models
• Configure model to match data
• Modify or expand to overcome flaws
Change parameters
• Setup parameters range
• Try different parameter combinations
• Best parameter estimation
• Evaluate result from parameter matrix
Evaluate models
• Perform tests
• Compare results
• Evaluate possibilities for improvement
• Finalize models

Figure 5 - Strategy overview of Classification

5.4 Implementation

The implementation of software is a core component in this project. It is responsible for providing test results and also a benchmark for the real-time perspective in speaker identification.

The process of software development has proven to be the most structurally complex phase in this project. This is especially true, as the software development process is experimental with regard to not knowing the exact requirements beforehand. They are developed alongside both the *Signal & feature analysis* phase and the *Classification* phase.

For these reasons we have chosen to build upon the Extreme Programming (XP) principles as the method which best fits our requirements for an agile software development method.

5.4.1 Why Extreme Programming?

The argumentation for this is provided by Kent Beck[1] who is the founder of the XP principles mainly used today. He states that projects which can be classified as in
Figure 6 are well suited for XP.

Projects suited to Extreme Programming are those that:
Involve new or prototype technology, where the requirements change rapidly, or some development is required to discover unforeseen implementation problems
Are research projects, where the resulting work is not the software product itself, but domain knowledge
Are small and more easily managed through informal methods

Figure 6 - Projects suited for Extreme Programming

This project certainly meets the conditions in Figure 6.

5.4.2 A few problems!

Assuming the use of XP based on the method by Kent Beck[1], there are (of cause) a few problems involved in using XP for this project:

Feedback
We haven't the availability of a user response teams and review teams also called Consumers in XP (as XP assumes).

Documentation
This project isn't about the software development process itself. XP can generate quite a lot of documentation if used properly, especially for very prototype projects.

Communication
There are no executive buyer whom to communicate with about demands and cost related issues etc.

[1] Kent Beck (2001) – "Extreme Programming Explained: Embrace Change" ISBN 0-201-61641-6

5.4.3 The 12 practices of Extreme Programming

As XP uses 12 main practices and it is suggested not to leave any of them out, we are unable fulfill this without Consumers and Buyers. So we must assume these roles ourselves (although unsatisfactory).

The 12 practices will therefore be as in Figure 7.

Practises	Comments
Fine scale feedback	
Pair programming	√
Planning Game	Authors are customers + buyers
Test Driven Development	√
Whole team	Authors are customers
Continuous process	
Continuous Integration	√
Design Improvement	√
Small Releases	Authors are customers
Shared understanding	
Coding Standards	√
Collective Code Ownership	√
Simple Design	√
System Metaphor	√
Programmer welfare	
Sustainable Pace	√

Figure 7 - The 12 practises of Extreme Programming

5.4.4 Documentation

As stated in (5.4.2) the focus of this project isn't software development. The software is a tool to examine and benchmark the problem of speaker identification in a real world scenario.

From a documentation point of view, we choose a (very) degraded version of the XP documentation scheme. The practices in Figure 7 will however be executed as if the development followed the XP method.

5.4.5 Extreme Programming software development strategy

As seen in Figure 8 there are 4 core activities in XP which meets the 12 practices from Figure 7.

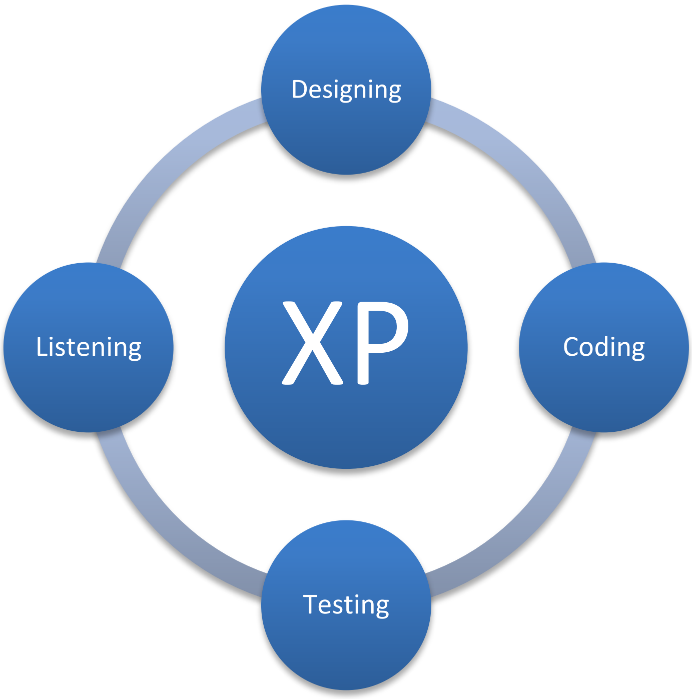


Figure 8 - The 4 core activities of Extreme Programming

Combining the core activities and practices we can model the development process in Figure 9 inspired by Don Wells[2]. The model is tailored to meet the limited scope of software development in this project.

Input and output of the XP development process

This model has 2 input feeds and 1 output feed.

Input feeds are the *architectural spikes*, and *goals*. Output feed is the *small releases* which are working programs at different steps of achieving the final program.

Definition of an architectural spike (by Don Wells 1999)

"A spike solution is an isolated program component to explore potential solutions. Build a solution which only addresses the problem under examination and ignore all other concerns."

[2] http://www.extremeprogramming.org (An introduction to XP by Don Wells 1999).

How architectural spikes relates to this project

In this project, the architectural spikes will be the small test programs developed in e.g. Matlab for solving different tasks related to speaker identification. Namely:

- Speech enhancement and noise removal functions
- Feature extraction functions
- Classification models

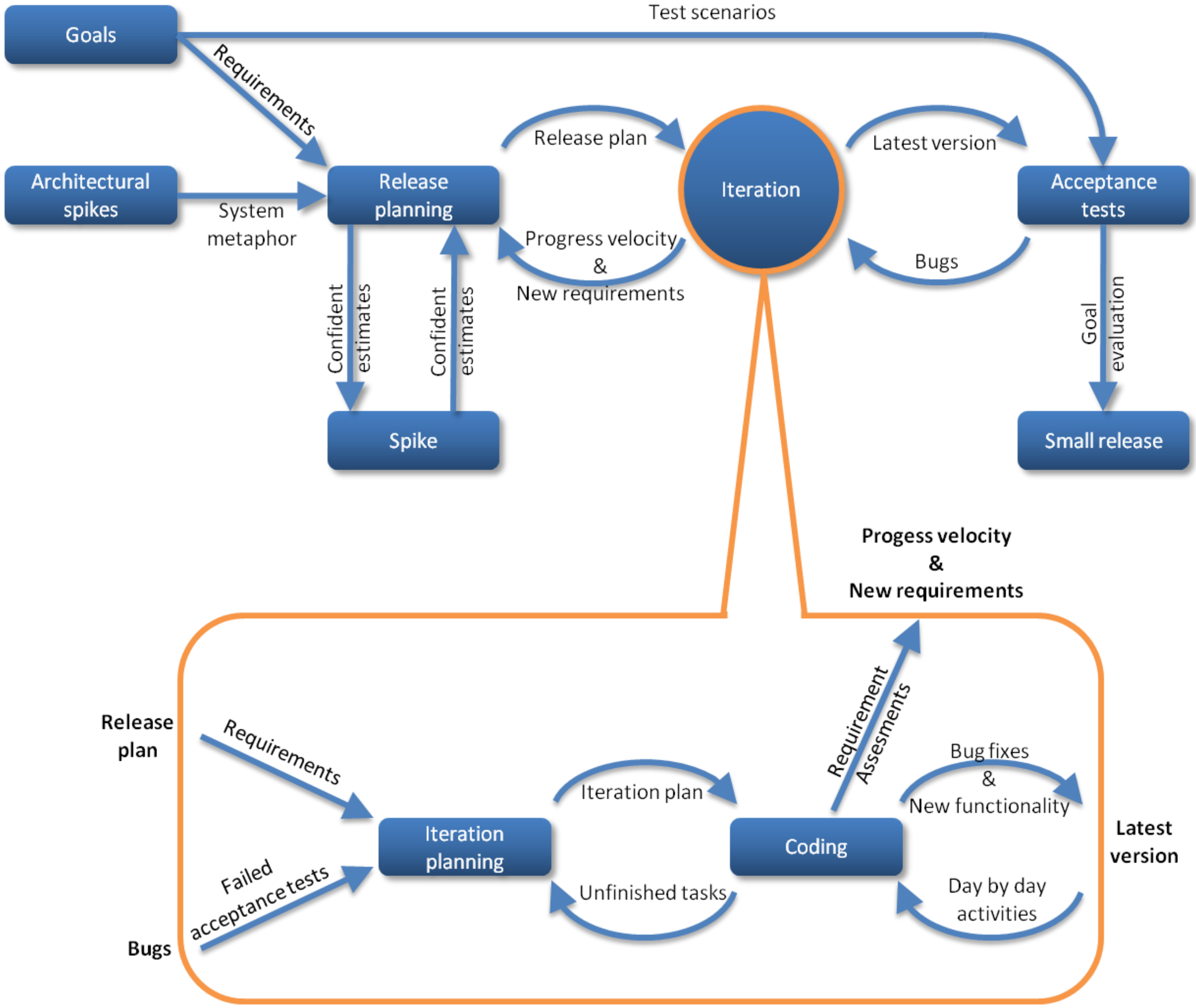


Figure 9 - Work flow of Extreme Programming

6) Contract & Scheduling

6.1 Known external obstructions

6.1.1 Parallel courses

Due to the ECTS requirement of the Civil engineering education, it has been necessary for both authors, to attend a parallel course in “Network and Integer programming”.

It is a 10 ECTS point course.

The expected duration of this course is from the beginning of February to the end of May 2007.

The influence of this course is expected to be rather high. Hence the time scheduled for this master thesis, is reduced during the mentioned period of time.

6.1.2 Planned holidays etc.

Over the summer, both authors have a planned vacation of 3 weeks duration. The vacations have been time aligned, to avoid standstills in the development process, due to critical sections demanding both participants to participate. Hence there will be a standstill in the master thesis development process in the period from August 15 to September 3, anno 2007.

Dan Bakmand-Mikalski is scheduled to vacate from August 15 to September 7, anno 2007.

Anders Havnsø Rasmussen is scheduled to vacate from August 7 to September 3, anno 2007.

6.2 Contract

Included as Appendix A.

6.3 Milestones

Included as Appendix B.

6.4 Time schedule

Included as Appendix C.

7) Work distribution

Figure 10 shows the distribution of work between the authors (main areas only). As both authors have been involved in most activities at some scale, this is only a reference as to who has had most responsibility. Also note that the different activities are not weighed equally so a summation is irrelevant.

To summarize, the authors are: Anders Havnsø Rasmussen (AHR) & Dan Bakmand-Mikalski (DBM).

	Activities	AHR	DBM
Process	General planning	50 %	50 %
	Documentation layout	30 %	70 %
	Risks	70 %	30 %
	Strategies	70 %	30 %
	Scheduling	50 %	50 %
	Project management	75 %	25 %
DSP	Front-end signal processing	60 %	40 %
	Noise removal	70 %	30 %
	Voice activity detection	75 %	25 %
	Feature analysis & extraction	75 %	25 %
Classification	Examination and choice of models	50 %	50 %
	Gaussian Mixture Models analysis	65 %	35 %
	Neural Networks analysis	35 %	65 %
	Implementation of major GMM components	80 %	20 %
	Implementation of major NN components	20 %	80 %
Implementation (Win32 C#)	Structural and multithreaded design	50 %	50 %
	Graphical user interface design	75 %	25 %
	C# Graphical components implementation	30 %	70 %
	C# Signal processing implementation (wrapper)	25 %	75 %
	C# Feature extraction implementation (wrapper)	80 %	20 %
	C# VAD implementation (wrapper)	20 %	80 %
	C# GMM implementation (wrapper)	80 %	20 %
	C# NN implementation (wrapper)	25 %	75 %
	Acceptance tests	? %	? %

Figure 10 - Work distribution between authors in %

Report:

Signal & feature analysis

Figure list:

1) Introduction to signal & feature analysis (DSP)

This part of the project focuses on 3 main subjects:

1. **Front-end signal processing (speech enhancement and noise removal)**
2. **Voice activity detection (Voice activity level analysis & RMS based voice analysis)**
3. **Feature analysis & extraction**

To clarify the relations between components, a simplified overview of the entire process is seen in Figure 1.

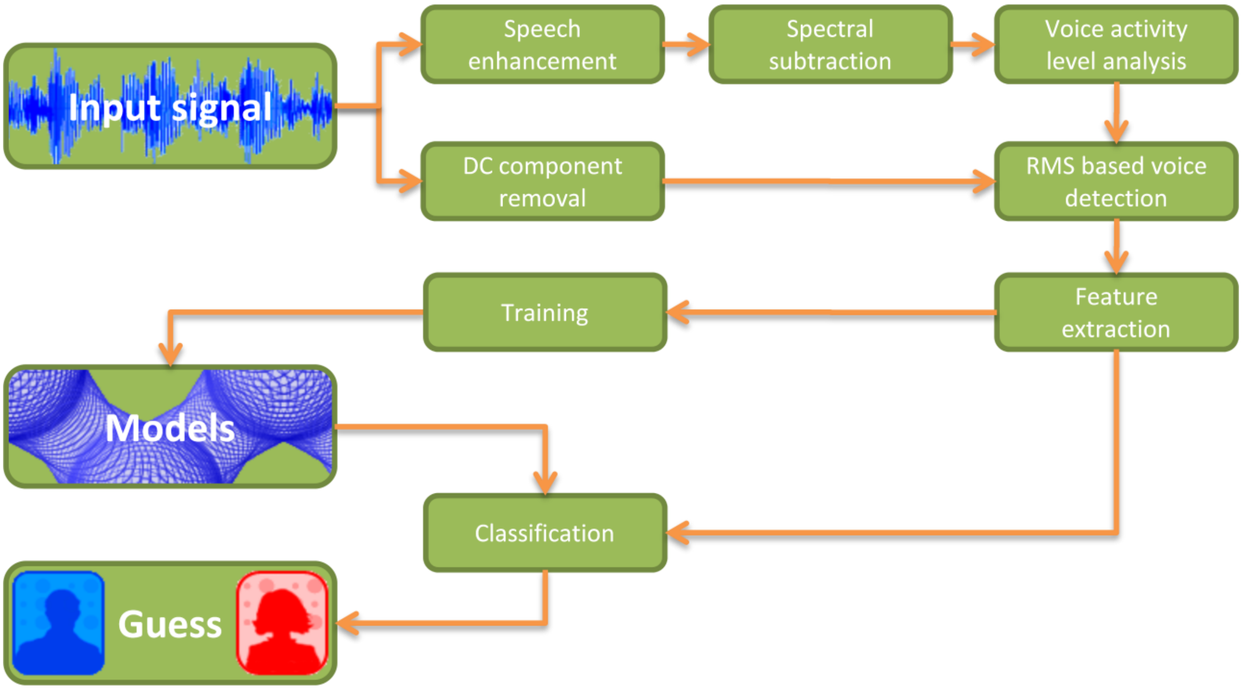


Figure 1 - Overview of Signal & Feature analysis

1.1 Front-end signal processing

This relates to: *DC component removal, Speech enhancement* and *Spectral subtraction* in Figure 1.

The purpose of front-end processing is to improve the input signal. As it is the speech part we are interested in, we enhances the speech through filtering and removes noise by spectral subtraction.

1.2 Voice activity detection (VAD)

This relates to: *Voice activity level analysis* and *RMS based speech analysis* in Figure 1.

Speech activity detection is a classic problem which is discussed in a multitude of whitepapers, articles, thesis's etc. The typical problems concerning robust speech activity detection is tradeoffs between speed/accuracy & scalability/robustness. In this project, speech activity is detected by using a combination of two methods:

- **Voice activity level analysis**
  This method detects voice activity levels. The method works best on an enhanced input signal. The result is speech segments including structural pauses.
- **RMS based voice detection**
  This method uses histogram equalization based on the RMS values. It is applied on the speech segments found by the voice activity level analysis. The benefit of also using the second method is that it is more accurate at detecting the precise speech boundaries.

1.3 Feature analysis & extraction

This relates to: *Feature extraction* in Figure 1.

The Feature analysis & extraction chapter focus on signal processing and how signals can be represented in different domains each providing specific information about the signals. Furthermore different features are described and implemented to find those that unlikely represent the traits of individual humans.

As the authors are relatively familiar with signal processing but are inexperienced with biologic speech production this chapter focus on already known features used for speaker identification. It would be impossible within this project period to get an extensive insight into audiology and use this for inventing new features.

1.4 How to test

Methods described in the chapters Front-end signal processing and Voice activity detection are all tested in Matlab using speech from the ELSDSR database and recordings produced by the authors. These self produced input signal are recorded using a webcam with an integrated microphone.

In the chapter Feature analysis & extraction PCA is used to evaluate individual features. PCA is a technique used to reduce multidimensional datasets. The method is very useful in analysing larger dataset as it is possible to reduce data onto e.g. the two or three most important dimensions which can be plotted in Matlab.

PCA is used to e.g. project LPCC's and MFCC's onto the 2 most important dimensions. But also to determine if the main part of variance is contained in few dimensions which would enable dimensionality reduction (to avoid the curse of dimensionality). This means that we use it as a tool for analyzing which methods to choose for feature extraction. A brief walkthrough of PCA in included as Appendix F.

1.5 The test speech signals

In this report, a variety of analyzes are performed on speech data obtained from both a controlled environment based on the ELSDSR database[1] and real life recordings by a webcam microphone with natural occurring noises recorded by the authors.

1.5.1 ELSDSR database

ELSDSR is a speech database containing speech sentences of 23 different persons (13 males and 10 females) in the age of 24 to 63. An example is given in Figure 2.

The database contains a training set with 7 sentences and a test set with 2 sentences from each speaker. The duration of each sentence is around 16 – 20 seconds. The sentences are sampled at 16 KHz, 16bits.

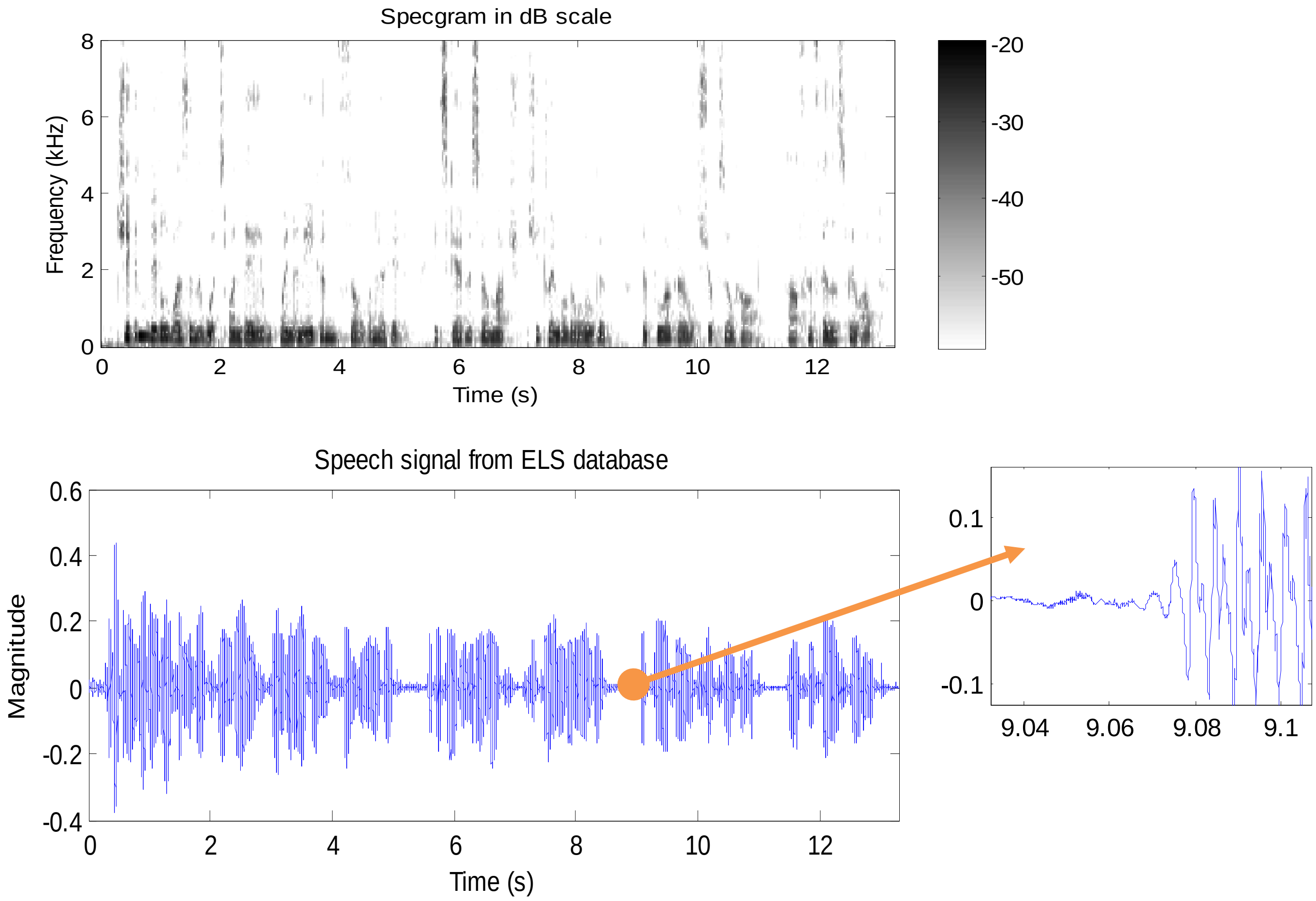


Figure 2 - Signal from ELSDSR database

[1] http://www2.imm.dtu.dk/~lf/ELSDSR.htm

1.5.2 Real life recordings

This data is recorded using a cheap Phillips webcam (Toucam pro series). A webcam microphone provides a realistic recording device of the context a speaker identification system would be utilized on.

In these recordings the speaker is approximately 1 meter away from the microphone. The microphone is turned in a 90° angle to the speaker. An example is given in Figure 3.

Opposite to data from the ELSDSR database, these recordings are used to evaluate how good performance the models yield in a more realistic/everyday scenario.

The recordings contain the following four noise elements:

- Vacuum cleaner (running for 9 seconds including power up and down).
- Drum sticks playing on table ½ meter from webcam.
- Road noise from open window next to main road (Sønder Boulevard 20 in Copenhagen).
- Walking and chair scrambling by other person (2-3 meters away from microphone).

The displayed signal contains drumsticks playing next to microphone.

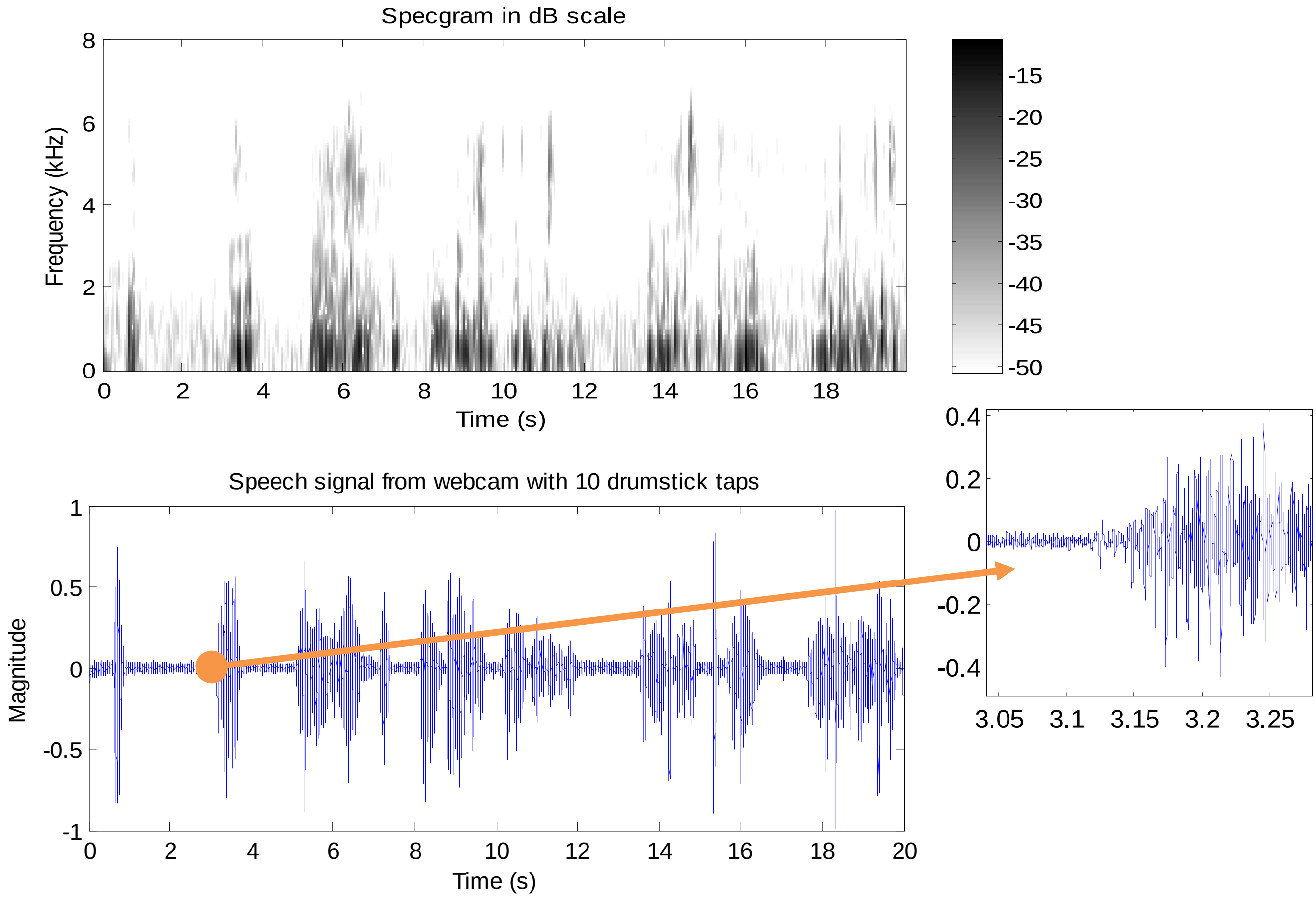
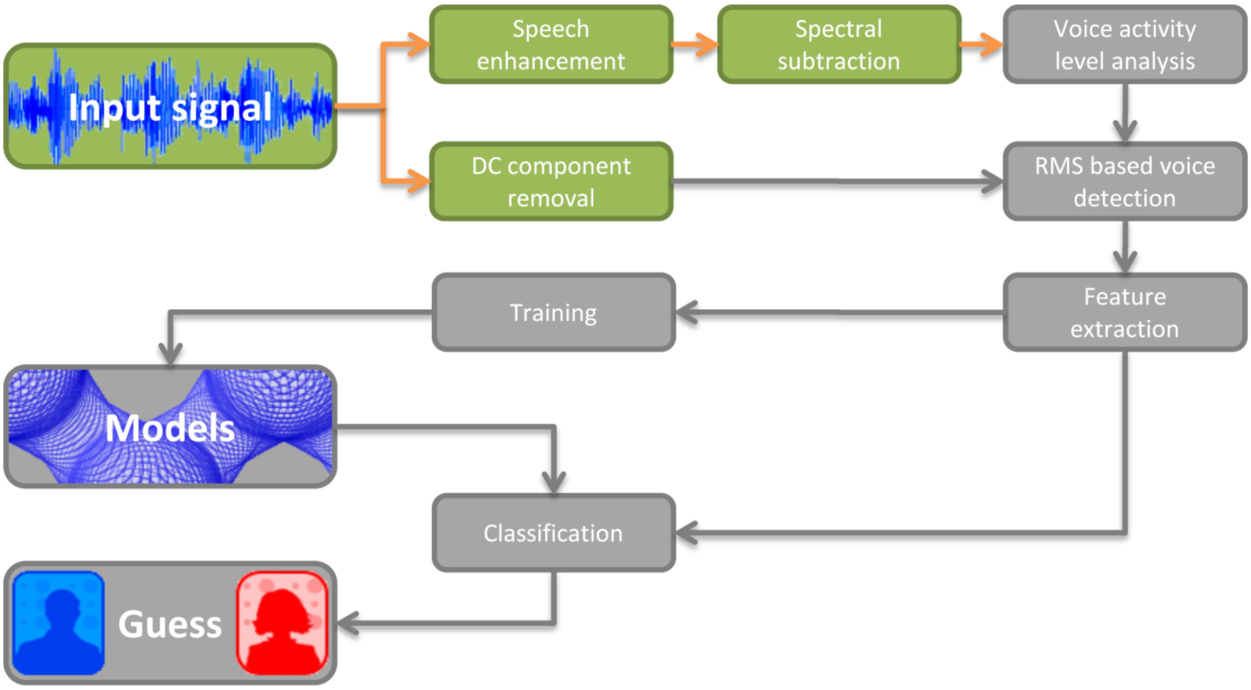


Figure 3 - Signal from webcam recording

2) Front-end signal processing

The purpose of front-end processing is to improve the input signal for both voice activity detection, feature extraction and finally classification.



In the section we look into some basic issues regarding the format of the input signal and some noise elements occurring when a recording is initiated. This is of cause only relevant for the audio recorded by webcam. Not data from the ELSDSR database.

To enhance the speech, we apply a potsband filter which emphasizes the speech band and dampens the sub/super speech frequency bands.

As recordings performed by cheap microphones contain a lot of noise, the removal of this is a priority. We have chosen to use a form of spectral subtraction. The main reason is that we don’t have a reference noise signal from which to estimate the noise, so we need to estimate it from the input signal itself.

A method for estimating and removing the DC component is also suggested. This is important as one of the Voice Activity Detection (VAD) methods we later use is error prone due to DC component.

2.1 Handling of basic input signal issues

2.1.1 *The basics*

Although not directly related to the subject of speaker identification, there are some issues related to the input signal that is important in this project. The explanation is as follows:

- **The recording device**
  Different recording both have differently characteristics and behavior. This is important in this project as we are focusing on the practical application of speaker identification.
- **Sampling frequency**
  Although trivial it is such an essential part of speech sampling so we covers this briefly.
- **Audio format**
  Again this is trivial but relevant due to our practical approach on speaker segmentation.

2.1.2 *The recording devices' influence on the recorded audio*

We have chosen to examine an impact on the recording sometimes referred to as the *signal on* effect or *power up* effect. It happens when a recording is initiated.

Although the development of our speaker identification system is mainly based on speech samples from the ELSDRS database (1.5.1) which is not influenced by this, we have also examined the impact on recordings performed by Matlab and by DirectX Audio. This is because the final application uses both Matlab and DirectX Audio libraries for recording.

Recording from Matlab

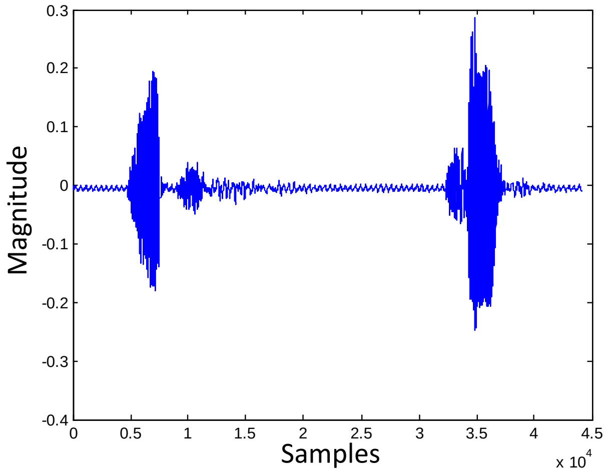


Figure 4 - Matlab recording

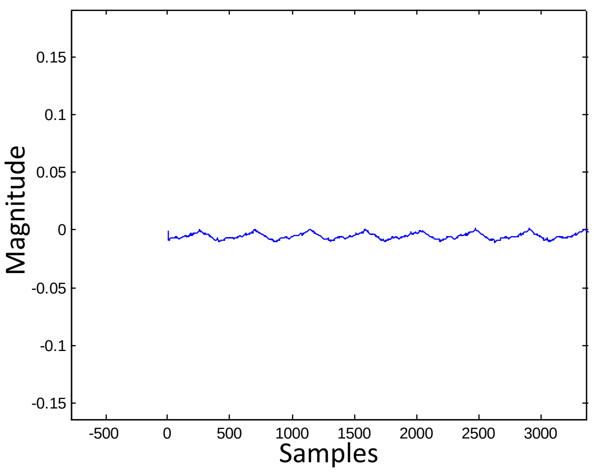


Figure 5 - Enlarged Matlab recording

A Matlab recording (Figure 4) using the *wavrecord* function is blackboxed from our point of view. We don't know how the function is working internally. But what we *do* know is that the recording doesn't have a power up component at the beginning. This is visible from Figure 5. Thus there are no issues involved when using Matlabs' *wavrecord* function.

Recording from DirectX Audio

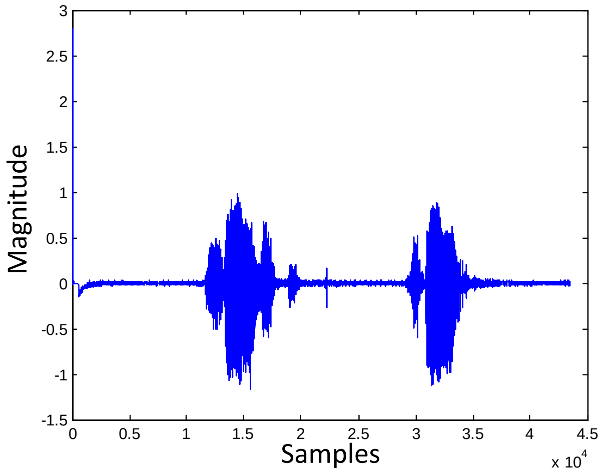


Figure 6 - DirectX Audio recording

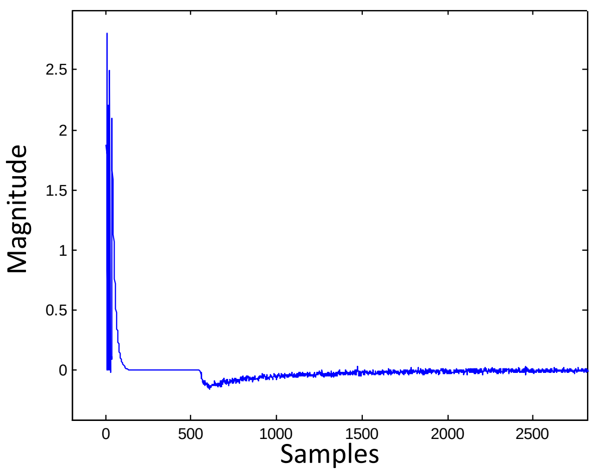


Figure 7 - Enlarged DirectX Audio recording

A recording using the DirectX Audio library (Figure 6) is a bit different though. By enlarging the first part of the signal we can clearly see a *power up* effect which is enlarged in Figure 7 .

One could claim that this has no significance due to the short burst time. This is not true however. As we use long term memory in some of the speech enhancement methods, this effect could significantly impact the computed values up several seconds into the future (relates e.g. to 2.2.4).

Now imagine that all recordings are done using a “press to speech” system, where the user initiates a new recording by pressing a button. It would mean that the *power up* effect would occur often and therefore have a large effect on the total robustness of the speaker identification system.

Another issue is that such a *power up* burst would be expected to be removed by the noise filters and speech enhancement processes. But this is not true for newly initiated recordings as these processes have a certain transient state before going into a steady state. This means for instance that the adaptive noise filters won’t be in effect until a few thousand samples into the recording.

As a result we have chosen to discard the first 2000 samples (125 ms) of each newly initiated recording. More sophisticated methods of detecting when the signal is stable could of cause be developed with some ease. But it is not really necessary know exactly wetter 75 ms or 125 ms should be discarded as the time intervals are so small. We have therefore selected a time interval which is reasonable.

2.1.3 *Sampling frequency*

Here we look into some initial issues related to limitations on sampling frequency.

2.1.3.1 Hardware limitations on sampling frequency

Most modern entry level recording devices has a peak frequency detection of just above 16 kHz[2]. This is also true for the webcam microphone (1.5.2) used for real life recordings in this project.

The general capabilities provided by most entry level soundcards and microphones are shown in Figure 8. We have neglected 12 bit because it is not supported by DirectX Audio API used later.

		Typical entry level gear				ELSDSR databse
		Mono		Stereo		Mono
		8 bit	16 bit	8 bit	16 bit	16 bit
Sampling rate (Hz)	8000			√	√	
	11025		√	√	√	
	16000	√	√	√	√	√
	44100		√		√	

Figure 8 - Audo recording capabilities

2.1.3.2 Sampling frequency of speech

The human voice is generally defined in the interval 500 Hz to 4 kHz[3].

The sampling rate must be at least twice the highest frequency contained in the spectrum also known as the Nyquist interval[4]. This can be stated as:

$$F_s \geq 2 \cdot f_{\max}$$

It would thus be sufficient to use a sampling rate of 8 kHz which enables detection of frequencies up to 4 kHz. This corresponds to the *ITU-T G.711* standard[5].

As the ELSDSR database (1.5.1) used for analysis is sampled at 16 kHz, it is possible to detect frequencies up to 8 KHz from the ELSDSR database.

[2] As of 2007 (check any microphone retailer for verification).
[3] http://en.wikipedia.org/wiki/Sampling_%28signal_processing%29
[4] http://en.wikipedia.org/wiki/Nyquist%E2%80%93Shannon_sampling_theorem
[5] http://en.wikipedia.org/wiki/G.711

Although significant computational advantages could be gained by down-sampling the ELSDSR database to fit G.711 we are hesitant to do this. It is a known issue, that down-sampling from e.g. 16 kHz -> 8 kHz won't produce a signal of equal quality compared to a signal originally recorded at 8 kHz [6]. Additive noise is a common problem when doing so.

Furthermore, the current standard of speech recognition systems (which speaker identification is closely related to) uses 16 kHz/16bits per sample which yields better classification results than 8 kHz 16bps.

It has therefore been chosen to use: *Sampling rate = 16 kHz, 16 bits per second.*

2.1.4 Audio format

By default audio is recorded in *wav* format (at least on WIN32 machines).

This *wav* format can be coded either by *mp3* which is a compressed format or as *Pulse Code Modulation (PCM)* which is an uncompressed format and thus takes up a lot of storage space.

We have chosen to use *PCM* for a number of reasons:

- It is a generic format and therefore compatible on most platforms.
- Being uncompressed it is fast and easy to work with.
- It doesn't degrade quality of original recording due to loss when compressing.
- It is the default format returned by DirectX Audio recordings (other options exist).

PCM is a block based represented of binary digits. Each block is 1 byte = 8 bits as seen in Figure 9.

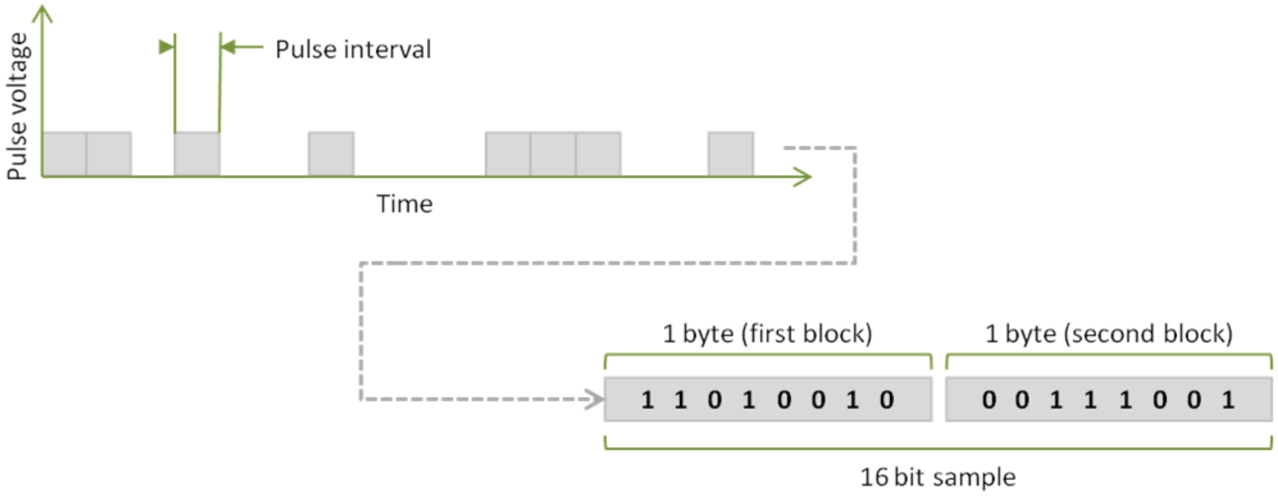


Figure 9 - Pulse Code Modulation

[6] Zhang, S.; Lapie, Y. (2003) – "Speech signal resampling by arbitrary rate"

2.2 DC-component removal

2.2.1 The basics

As the RMS based method used for speech activity detection is error prone to rapid changes or offsets in the DC-component estimate, it is a necessity to normalize it (remove it). The DC-component removal mainly relates to (3.3.3.3- A problem with DC component and RMS).

If the whole signal is known, one can use Lemma 1.

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad \text{Lemma 1}$$

But as the signal is streamed, the DC-component can only be estimated based on already streamed samples at best. This means that it is necessary to estimate the DC-component from the already streamed input data.

The challenge is to make an accurate estimate of the DC component in a computational feasible way.

Two suggestions for removing DC component from streaming input

We have chosen to examine 2 methods capable of achieving DC-removal.

The two approaches are tradeoffs between speed, memory consumption and usability where the first yields results instantly and the latter is faster and uses less memory.

1. *DC-component removal using cache of μ estimates.*
   This method performs estimation and removal of μ using caches of samples and mean values each based on a preset number of input samples. A drawback of using cache to remove the DC component is that it can only remove mean in preset intervals of e.g. 20 ms.
2. *DC-component removal using Filtering.*
   This method uses filtering only. The filter removes the DC-component based on the local mean value within the scope of the filter which is the same as using Lemma 1 on the newest part of the input signal. Significant drawbacks are instability until filter buffer is full and high memory consumption for accurate estimates.

2.2.2 *DC-component removal using cache of μ estimates*

In section *3.3.4.3 - Avoiding re-computing values* it is described how the RMS energy histogram is based on a cache containing RMS values. Each RMS value is yet again based on a preset number of samples. Each time the preset number of samples is streamed, the oldest of these RMS values are removed from cache and a new RMS value is computed and added. This is done continuously during input streaming.

2.2.2.1 An initial problem

The relevance is that when a new RMS value is to be computed it is necessary to have the DC component removed from the particular samples that the new RMS value is computed from.

Any new estimate of the DC-component (equivalent to the μ value of the entire input signal) must have a scale corresponding to the μ value which was subtracted from the already processed samples on which the "old" RMS values in cache are computed from. Otherwise the RMS values are not comparable. It is therefore a requirement that any alterations to the DC-component estimate are performed gradually.

2.2.2.2 Cache based computation of μ with local mean estimates.

In this case however, a solution is at hand.

To avoid storing a lot of samples from the streamed input signal and avoid re-computing any values, a local mean for each frame (containing a preset number of samples) is computed and cached. This ensures that every mean value corresponds to a given RMS frame and also that a global mean can be estimated.

The following components & variables are necessary:

f	Cache of local mean values $[f_0 \quad f_1 \quad f_2 \quad \cdots \quad f_{h-1}]$
s	Samples per frame or RMS value
h	Length of the mean values cache (must at least be equal to length of RMS cache to work properly)
x^{new}	Vector containing s newly streamed samples.

Then Lemma 2 can be used to update the current μ estimate without any re-computations. The equation is a customized extension of the traditional normalized mean equation.

$$\acute{\mu} = \left(\mu - \frac{1}{h} f_0\right) + \frac{1}{h} \sum_{i=1}^{s} \frac{x_i^{new}}{s}$$

Lemma 2

2.2.2.3 Cache flow diagram

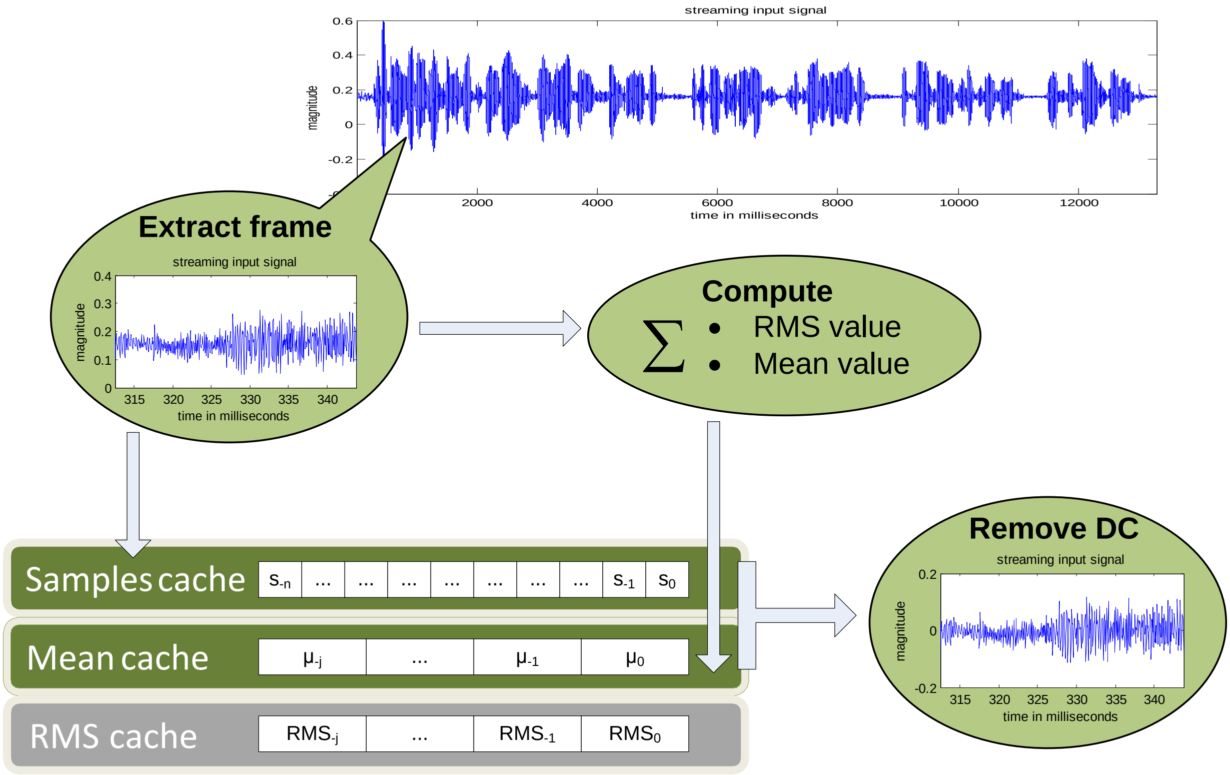


Figure 10 - Cache flow diagram

The 3 step procedure

1) As seen in Figure 10 we cache a preset amount of streamed samples in the *samples cache*.
2) In fixed intervals we compute the local mean value of the *samples cache*. The local mean values are added to the *mean cache*.
3) By averaging over the *mean cache* using the equations exemplified in (2.2.2.4) we can update the global mean estimate without any re-computations. This enables us to remove the DC-component from the newly streamed samples before further processing.

Optimization

The idea behind this setup is to induce long term memory at a very low computational and memory cost. It basically performs the same function as a running average filter, but it is capable of achieving the same result with significantly lower memory consumption and at much higher speed. A comparison of running average filter vs. cache based estimation can be seen in Figure 12 in section (2.2.4).

Drawbacks

The major drawback of this method is that it can only deliver mean estimates in intervals equal to the size of the *samples cache* (each 20 ms in the example above). This makes it more difficult to use in a practical context.

2.2.2.4 Example: Cache based computation of μ

$x = [11 \quad 3 \quad 2 \quad 7 \quad 1 \quad 9 \quad 2 \quad 1 \quad 1 \quad 3] \ (Currently\ streamed\ input\ samples)$

$x^{new} = [3 \quad 1] \ (Not\ yet\ streamed\ input)$

$s = 2 \ (Samples\ pr.frame)$

$h = 5 \ (number\ of\ values\ in\ cache. One\ value\ for\ each\ frame)$

1) Compute μ for each frame and put them in cache (already computed values is not re-computed in actual implementation)

$$f_j = \frac{1}{s}\sum_{i=1}^{s} x_{i+js} \quad for\ j = 0 \ldots (h-1)$$

$$f = \begin{bmatrix} f_0 & f_1 & f_2 & f_3 & f_4 \\ 7 & 4.5 & 5 & 1.5 & 2 \end{bmatrix}$$

$$\mu = 4$$

2) Now suppose that the two new samples labeled x^{new} are streamed. The estimate of μ can be updated by this formula.

Cache based computation of μ with local mean estimates.

Each local mean estimate corresponds to a particular RMS value (described in 3.3.3.2).

$$\acute{\mu} = \left(\mu - \frac{1}{h} f_0\right) + \frac{1}{h}\sum_{i=1}^{s} \frac{x_i^{new}}{s}$$

$$\acute{\mu} = \left(\mu - \frac{1}{5} \cdot 7\right) + \frac{1}{5} \cdot 2$$

$$\acute{\mu} = 3$$

3) Update the cache f with the mean value of x^{new} and continue from 1).

$$f = \begin{bmatrix} f_0 & f_1 & f_2 & f_3 & f_4 \\ 4.5 & 5 & 1.5 & 2 & 2 \end{bmatrix}$$

2.2.3 DC-component removal using filtering

As this method is not chosen we only give a short summary of it in this report. A detailed analysis is available in Appendix D1.

The running average filter (or mean filter as it is sometimes called) is one of the simplest filters. All coefficients are equal and normalized.

$$H(Z) = \frac{1}{k}[1 \ldots k]\,Z^{-1} \qquad Where\ k\ =\ filter\ size\ (order)$$

The precision of the mean estimate based on 10 signals of length 11-15 seconds at a sampling frequency of 16 kHz is shown in Figure 11.

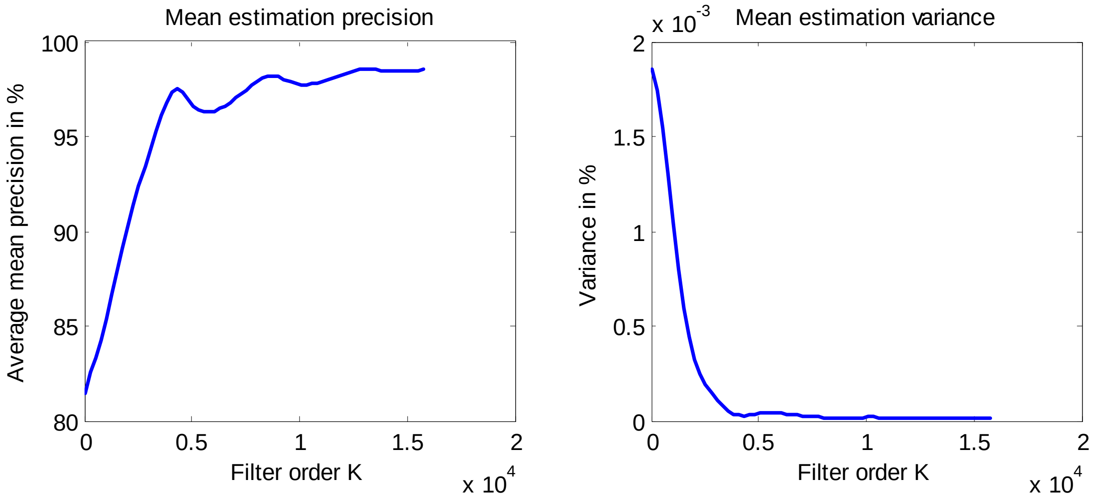


Figure 11 - Relation between filter order and estimate precision

From the Figure 11, it can be seen that the filter order greatly influences the precision of the mean estimate. This can be evaluated because we use known signals in this example.

When the variance between the estimates and the true mean decreases, it is also obvious that the running average converges towards the true mean.

2.2.4 *Results of DC removal on streaming signal*

This section summarizes the result of DC removal. An extended description of the results and how they are computed can be found in Appendix D2.

If we compare the memory and computational cost difference using the configuration in Figure 12, it is clear that Cache based estimation greatly reduces memory consumption and computational cost so this method is chosen.

	Cache based estimation	**Running average filter**
Samples cache size (s)	320 (20 milliseconds)	
Mean cache size (h)	1000 (2 seconds in 20 ms intervals)	
Filter size (n)		32000 (2 seconds of samples)
Long term memory	2 seconds	2 seconds
Computational cost pr. 20 ms	O(s)	O(n)
Total memory use	4640 bytes (320 uint16 + 1000 uint32)	64000 bytes (32000 uint16)

Figure 12 - Cost table of running average filter vs. cache based estimates

Using the parameters from Figure 12 we compute the estimation accuracy by Lemma 3 and get the following result:

$$\frac{100}{\mu + error}\mu \approx 98.4\ \%$$

Lemma 3

The accuracy is typically in the range of: *98% - 99%* based on 2 seconds mean cache size as seen in Figure 13.

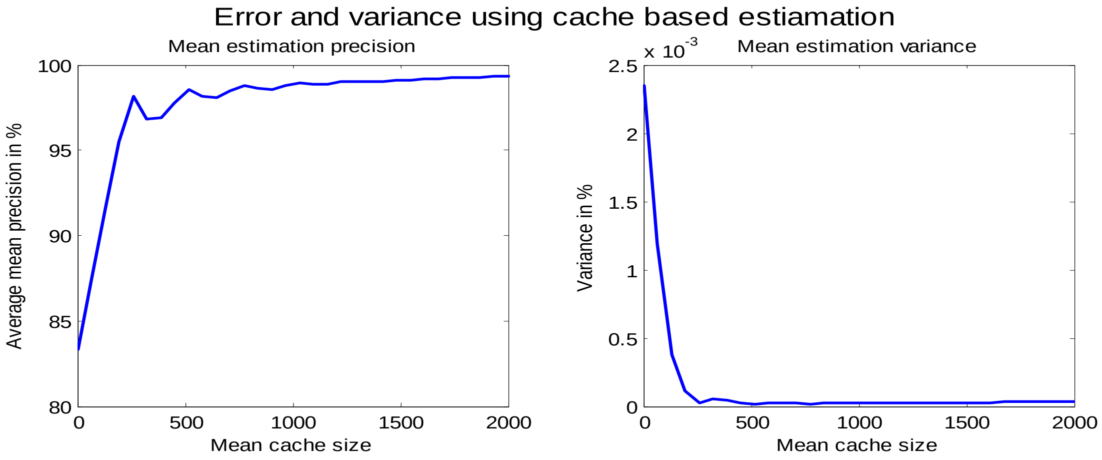


Figure 13 - Relation between mean cache size and precision

2.3 Speech enhancement

2.3.1 *The basics*

The purpose of speech enhancement as we use it, is to emphasize the speech by filtering out the sub-speech frequencies and the super speech frequencies.

We have chosen to use a Potsband filter (named after the inventer) to emphasize speech. Additionally we use a high and a low pass filter to solve some specific issues.

2.3.2 *Potsband filtering*

The bandwidth is based upon the speech bandwidth. The filter corresponds to the specifications of ITU-T G.151 recommendation as shown in Figure 14.

Potsband filter specification	
Lower band	300 hZ
Upper band	3400 kHz
Passband gain	1dB
Gain at passband edges	-3dB

Figure 14 . Potsband filter specification

The filter characteristics is easily visible by the bandwidth of the frequency response (Figure 15) and zero-pole plot (Figure 16)

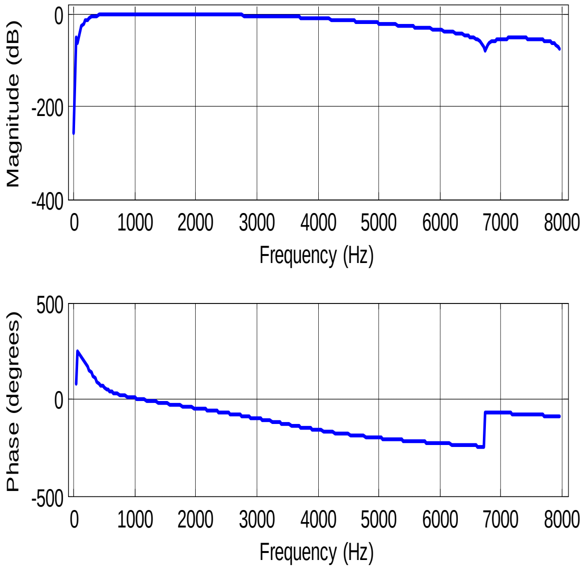


Figure 15 - Potsband bandwidth

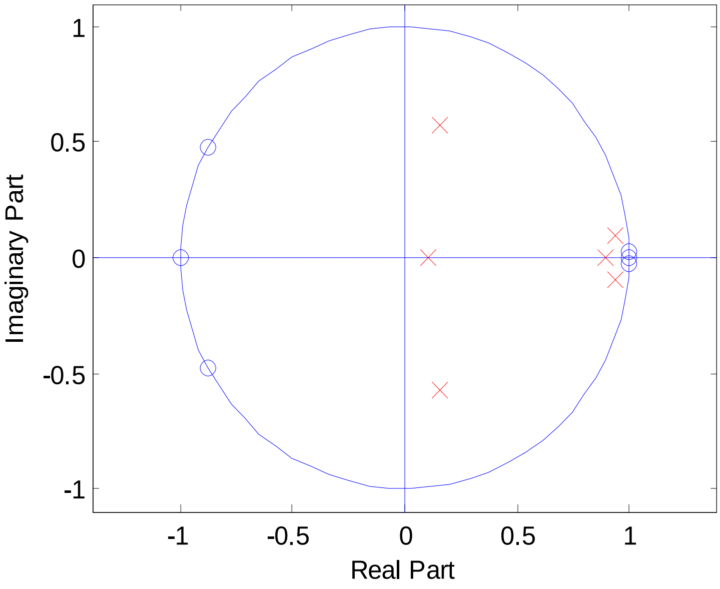


Figure 16 - Potsband zero-pole

From Figure 16 it can be seen that the filter has a tight dampening in the low frequency domain and a more loosely defined dampening in the high frequency domain.

The poles in the middle ensures little or no dampening in the mid speech frequency range at approximately 3.5 kHz.

This can also clearly be seen in Figure 15 where the transition is steep in the low frequency domain and smooth in the high frequency domain.

The end result is a filter with a high dampening of low frequencies and a more gradually dampening of high frequencies above the audible speech frequencies (3.4 kHz).

2.3.3 An additional filter for dampening recording contamination

This is mainly an issue relating to recordings performed by our webcam microphone (1.5.2).

At last minute before deadline some high frequency contamination by some recording equipment has been discovered. The contamination typically occurs when the equipment has a maximum recording frequency range of e.g. 16.000 kHz and we are sampling near the equipments maximum capabilities. This was discovered after discussion with a former technician[7] from Madsen Electronics (hearing aid developer).

The reason this wasn't noticed initially was because of how the frequency spectrums were plotted. Using Matlab's default "Jet" color scheme this contamination simply wasn't visible (or almost). But when applying a custom gray level scheme it was clearly visible. The difference is visible from Figure 17 and Figure 18.

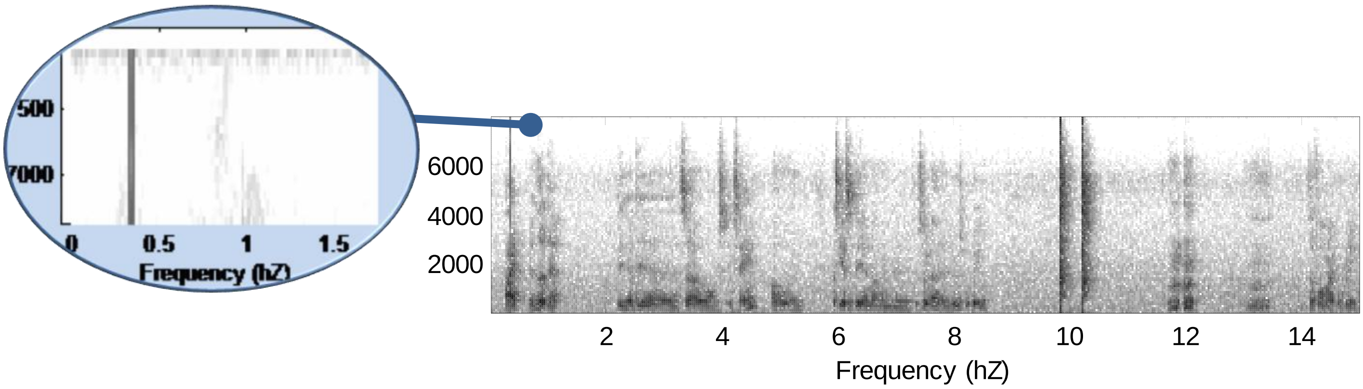


Figure 17 - High frequency contamination slightly visible

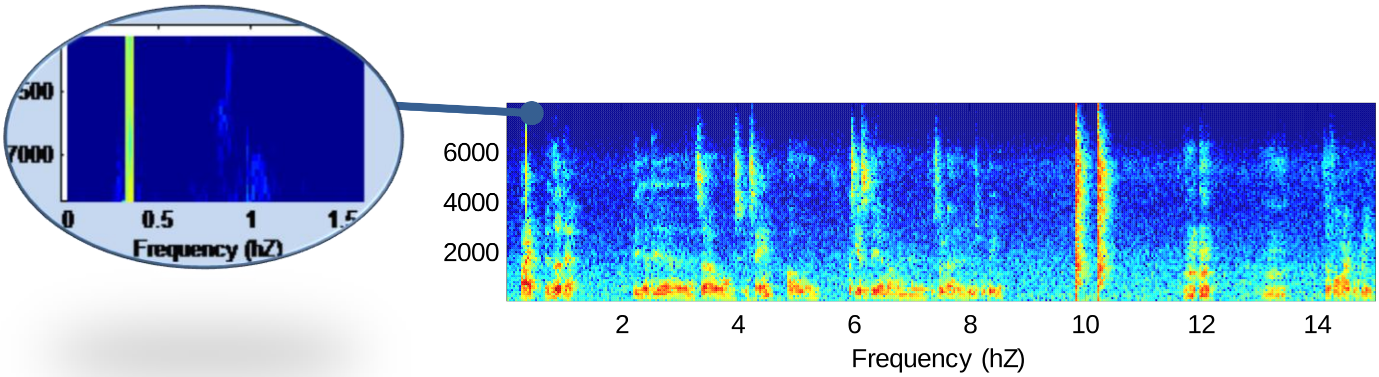


Figure 18 - High frequency contamination invisible

This contamination can be removed by low-pass filtering the signal as seen in Figure 19 and Figure 20.

[7] Jan Mikalski (jan_mikalski@gmail.com), former employee at Madsen Electronics, currently working with electronic design.

Although the potsband filter and this low pass filter could be designed as one we have chosen to use a second filter. This is in order to reduce the required filter order and for various implementational reasons (this is a really last minute addition).

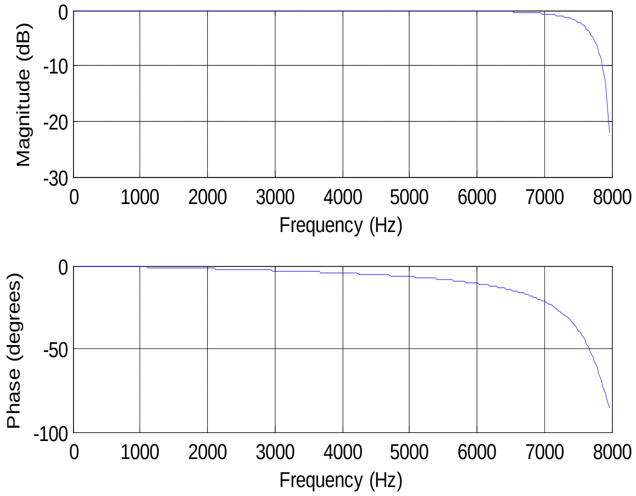


Figure 19 - Lowpass bandwidth

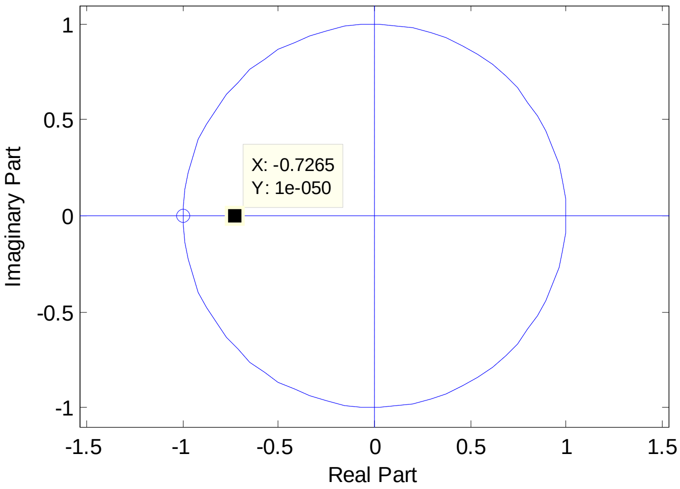


Figure 20 - Lowpass zero pole

2.3.4 Results

By applying the Potsband filter and the lowpass filter described in this section, the improvement is significant.

From Figure 21 we can clearly see that the sub and super speech regions are dampened while speech frequency region is preserved. This was also what was intended.

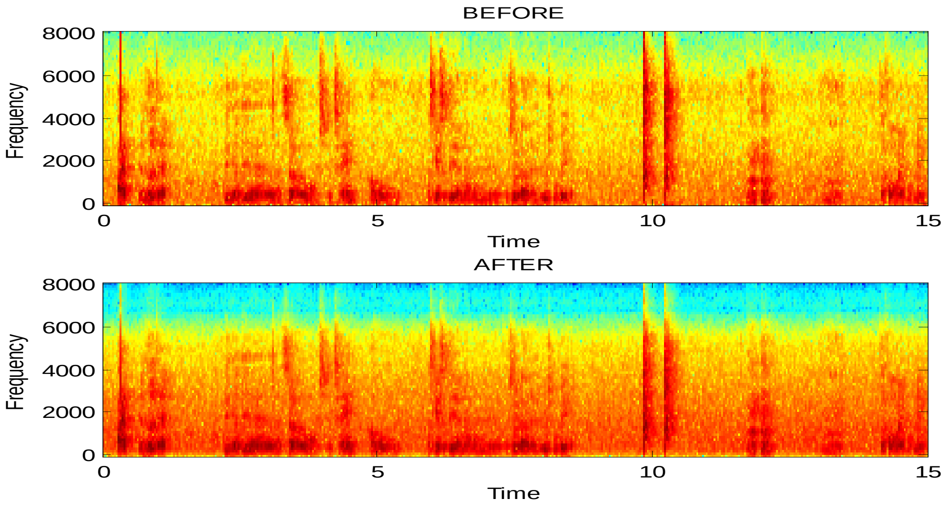


Figure 21 - Result of Potsband + lowpass filtering

Notice the difference in the high frequency region (7500-8000 hZ) after filtering. It happens because the low-pass filter further dampens that region.

By looking at the signal in time domain (Figure 22), we can also see that the speech is virtually intact but the noise is significantly reduced. This is because a large portion of the noise exists in the sub-speech frequency domain.

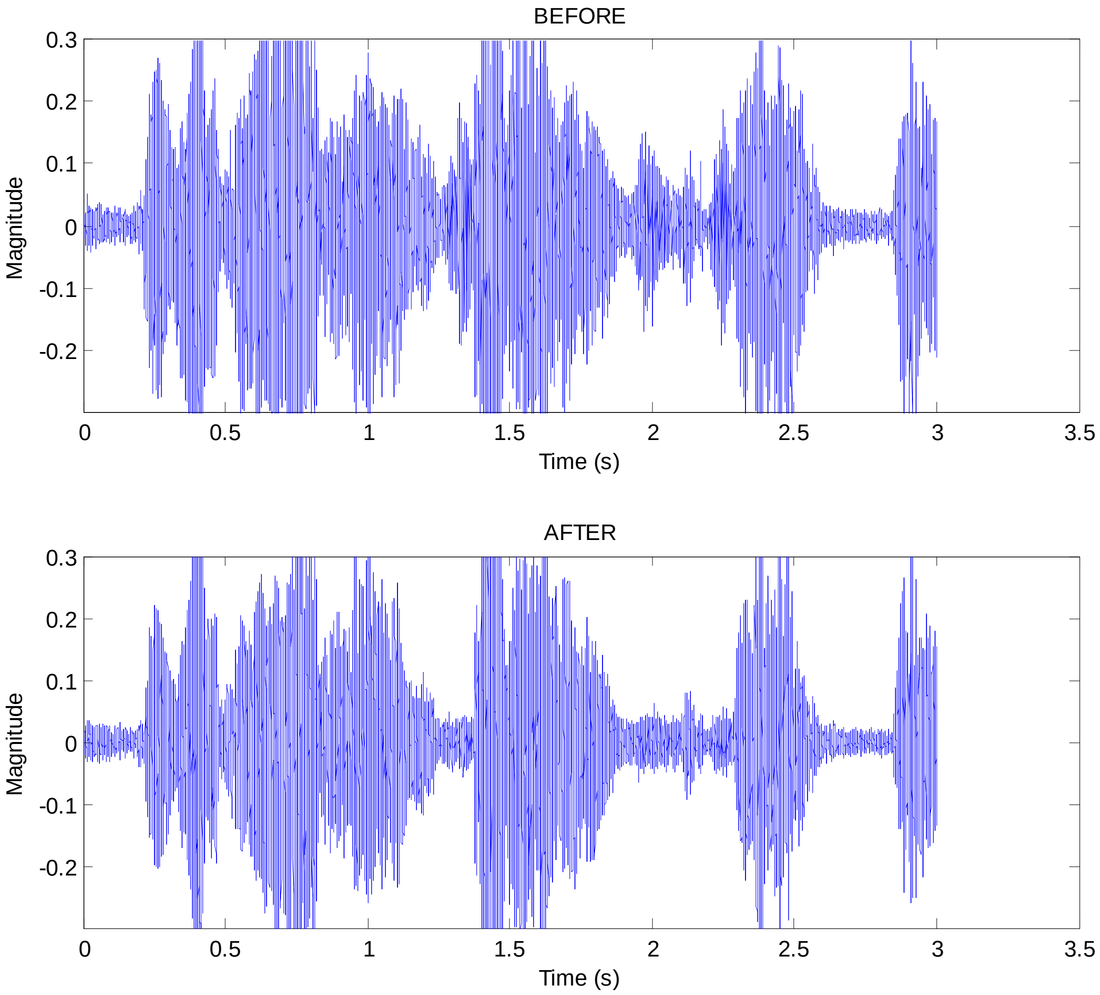


Figure 22 – Zoom in on Potsband filtered signal in time domain

By audible listening to the signal before and after filtering the sound quality is perceptually improved but not excessively. Most noticeable was a significant noise reduction.

2.4 Noise removal by spectral subtraction

2.4.1 The basics

Noise removal is essential for maximizing the input signal quality. We are however limited by only knowing the input signal and not the noise signal (which e.g. would allow for adaptive filtering using the LMS algorithm).

We have therefore chosen to look into spectral subtraction, which is an old but widely used method for noise reduction in the spectral domain.

In this project the noise problem can be stated as how to extract an estimate of the desired signal from a noisy input signal. This is formulated in Lemma 4.

$$\left|\hat{S}(e^{i\theta})\right| = \left|S(e^{i\theta}) + N(e^{i\theta})\right| - \left|\hat{N}(e^{i\theta})\right|$$

Lemma 4

The phase of the noise is unknown. It is in praxis therefore only necessary to use the real part of the Fast Fourier Transform (FFT) when converting input signal S into spectral domain.

The input signal $s(n)$ is conceptually a frame of samples in spectral subtraction.

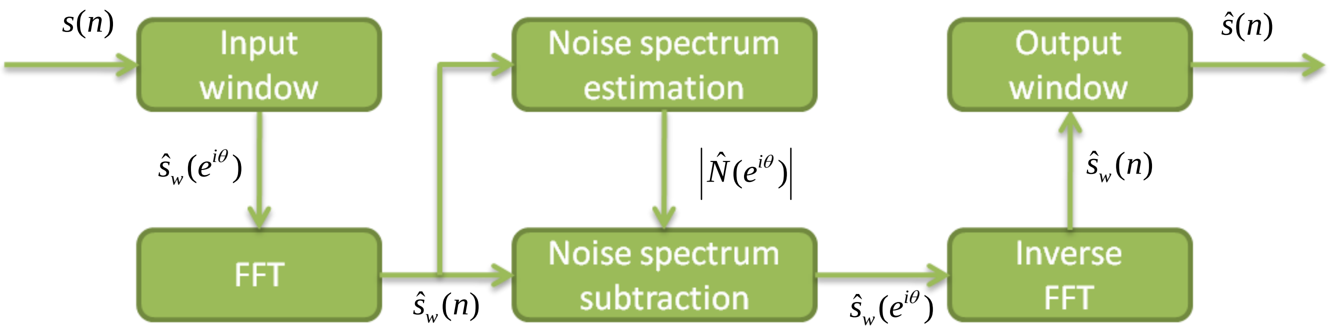


Figure 23 - Overview of spectral subtraction

From Figure 23 we can see that the samples are initially windowed to avoid spectral artifacts resulting from the discontinuities in the boundaries of the frames processed. A final window is used to cancel the effect of the first window.

In this project, we expand on the spectral subtraction model proposed by Mike Brooks[8] (2001)

[8] Mike Brooks (2001) - http://www.ee.ic.ac.uk/pcheung/teaching/ee3_Study_Project/speechen_lab.pdf

2.4.2 Problems when estimating noise

There are some problems we have to address when estimating the noise.

2.4.2.1 Unknown phase

Consider that the estimated noise $\left|\hat{N}(e^{i\theta})\right|$ in spectral domain results in an estimated desired signal:

$$\left|\hat{S}(e^{i\theta})\right| < 0$$

If the estimated desired signal in spectral domain is negative it contradicts the laws of physics as the frequency spectrum of a signal can't contain negative values.

This happens because we don't know the phase of the noise $\left|N(e^{i\theta})\right|$.

One obvious way to handle this is to raise all negative estimates to zero as in Lemma 5.

$$\left|\hat{S}(e^{i\theta})\right| = \left\{ \begin{matrix} 0 & \quad if \left|\hat{S}(e^{i\theta})\right| < 0 \\ \left|\hat{S}(e^{i\theta})\right| & \quad if \left|\hat{S}(e^{i\theta})\right| \geq 0 \end{matrix} \right\}$$

Lemma 5

2.4.2.2 Over/under estimates of noise

There is a problem which could occur when estimating noise in regions containing speech. If the estimated noise is significantly larger than the real noise as in Lemma 6, then a part of the desired signal (speech) will be corrupted which leads to distortion.

$$\left|\hat{N}(e^{i\theta})\right| >> \left|N(e^{i\theta})\right|$$

Lemma 6

The opposite problem also exists. A significant part of the noise will still remain in $\left|\hat{S}(e^{i\theta})\right|$ in the likely case that the noise estimate is too pessimistic as in Lemma 7?

$$\left|\hat{N}(e^{i\theta})\right| << \left|N(e^{i\theta})\right|$$

Lemma 7

These problems are handled in the following sections.

2.4.3 Oversampling

Aliasing is a concern when using spectral subtraction. This *could* happen if the frequency component changes rapidly.

Therefore we use oversampling which in this case means that we use an overlap of a number of previous samples for each new frame processed.

Let’s assume:

$$\textit{Oversampling constant} = 4$$

If using a framesize of 512 samples (32 ms) then we must process a frame each time 64 samples are streamed (4 ms) as in Figure 24.

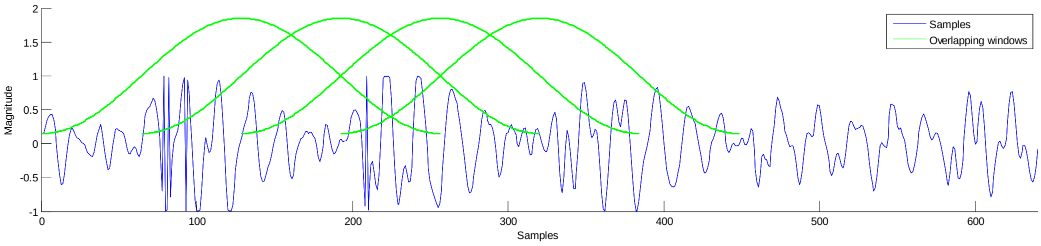


Figure 24 - Oversampling input

This means that an input and output buffer is required for this. It must contain 640 samples. As soon as 128 samples are streamed, they are processed and the buffers are shifted.

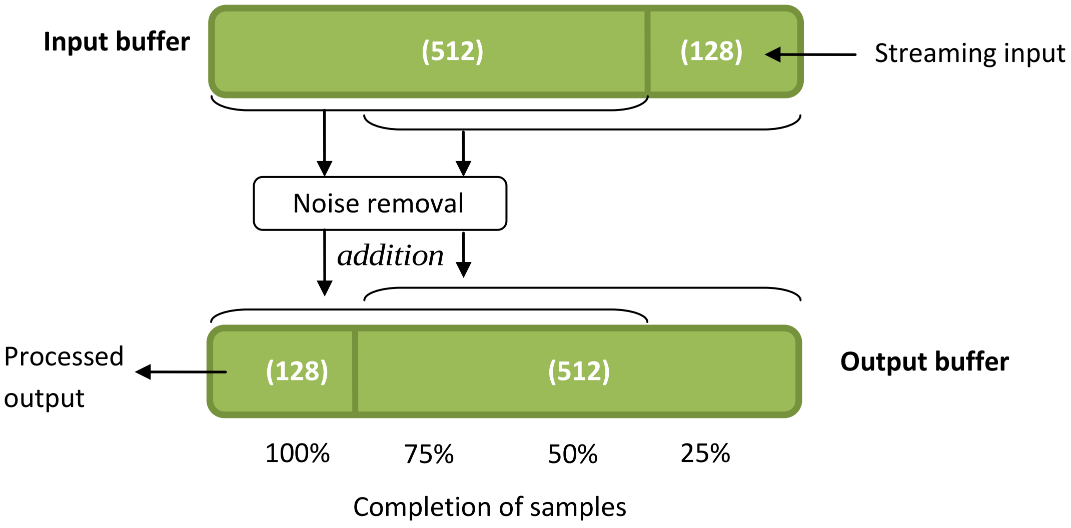


Figure 25 - Oversampling buffer operation

From the illustration in Figure 25 we can see that as we processes overlapping frames we need to add the input in chunks as well. This means that the samples are reformed in 4 steps by addition.

2.4.4 Input output windows

To avoid discontinuities at frame borders we apply a window of type Hamming to the input samples.

After the noise is removed some discontinuities will exist due to the noise removal process. These are removed by the output window.

Assume that each input signal frame is 32 ms.

This means that we require a window size of 512 as in Lemma 8:

$$\frac{fs}{1000}\cdot framesize$$

Lemma 8

$$\frac{16000}{1000}\cdot 32 = 512$$

It is very important that the windows sum to 1 because of the way samples are reconstructed. Each sample is the sum of a number of steps (the oversampling constant) as in (2.4.3). As each step is based on a separate window operation the total summation must remain unscaled by the window operations no matter where in the windows the sample is located.

This can be ensured by the window definitions in Lemma 9.

$$input\_window(k) = 1 - 0.85185\cdot\cos((2k+1)\pi / N)$$
$$output\_window(k) = \sqrt{1 - 0.85185\cdot\cos((2k+1)\pi / N)}$$

Lemma 9

As seen in Figure 26 the windows sum to a constant (the oversampling constant) in the overlapping region.

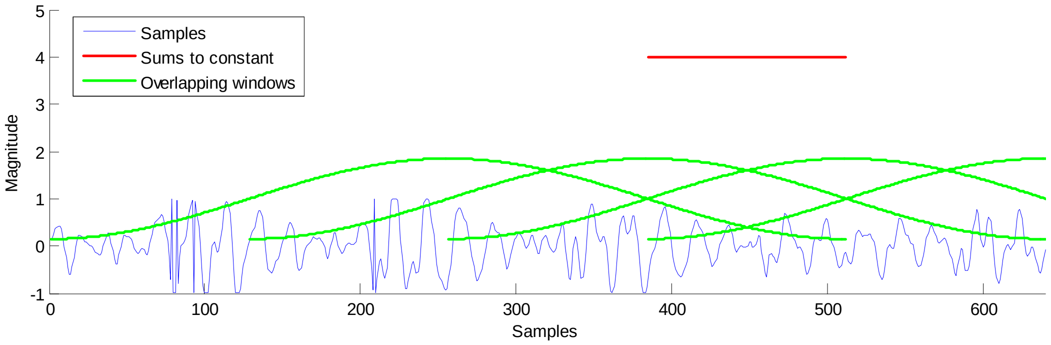


Figure 26 - Overlapping windows

2.4.5 *Estimating noise using minimum buffers*

This is the core of spectral subtraction. We need to know some spectral signature of the noise in order to minimize it. For this we use minimization buffers.

The minimization buffer, is a buffer containing the spectral signatures of previous frames.

The reason for storing multiple spectral signatures is to have a range of possible pure noise signatures to average on. Then it is possible to find the best spectral subtraction match by trying to subtract each of the possible pure noise signatures from the "new" frame to be de-noised. An evaluation of and afterwards measure on how well the "new" frame can absorb the different pure noise signatures is used to determine which one to use.

There are some challenges to this:

1. In order for minimization buffers to work, we must assume that at least 1 frame within the buffers scope contains a pure noise frame with no speech.
2. A signature should be as new as possible for best results.
3. The minimization buffer should be large enough to span a spatial region which is certain to include a pure noise frame.
4. Saving a lot of spectral signatures (1 vector for each frame) uses a lot of memory if buffer is large.
5. Computational cost of finding minimum spectral signature is computational infeasible for buffer with excessive signatures stored.

To overcome these problems, we use an approach with a buffer where each item contains the "best" noise signature from a preset number of frames (solves challenge 4 and 5).

We choose the buffer to span a few seconds (e.g. 2 seconds) true time (solves challenge 1, 2 and 3). The minimization buffer combined with noise signatures of e.g. 32 ms ensures that a pure spectral noise signature can be found even in regions containing speech (there are always pauses in natural speech).

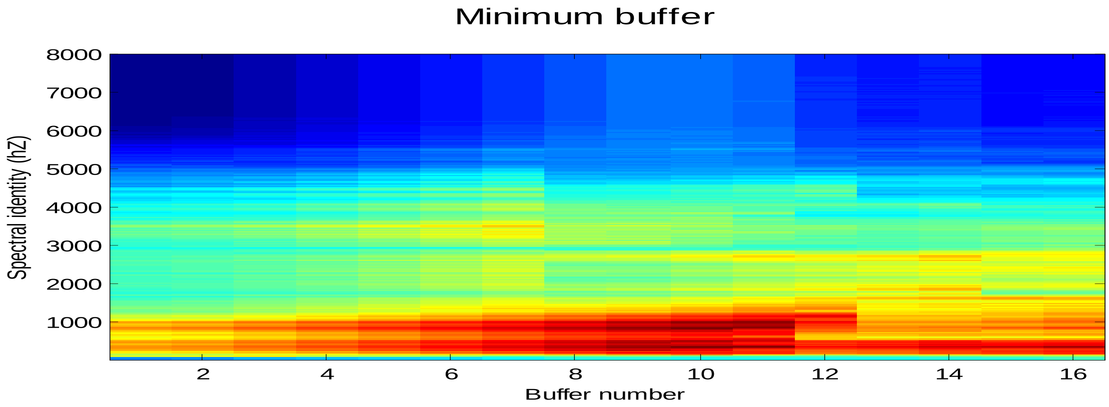


Figure 27 - Minimum buffer

As seen in Figure 27 we use a minimum buffer storing 15 previous spectral identities. The 16th is the one currently being build. When finished building the 16th buffer item, the oldest item will be discarded and so on and so forth.

In this example we chose to use a buffer spanning 2000 milliseconds. This means that each buffer item will contain the "best" spectral noise signature in an interval of approximately:

$$\frac{buffer\_span}{buffer\_items}\cdot oversampling\_constant = item\_span\ (ms)$$

Using the variables from earlier this equivalent: $\frac{2000}{16}\cdot 4 = 500\ (ms)$

If we chose to use frames of 32 ms this means that each buffer item will contain the "best" spectral noise signature from approximately 15 frames. The "best" spectral noise signature is the frame with the minimum PDS (Power Density Spectrum) in an interval of 500 milliseconds.

2.4.6 Subtracting noise spectrum

By averaging over the assumed pure noise spectral signatures in the minimum buffer and subtracting it from the input signal, the noise will be significantly dampened. The noise estimate can be computed by Lemma 10 where M is the minimum buffer length.

$$\left|N(e^{i\theta})\right| = \frac{1}{M}\sum_{m=0}^{M-1}\left|N_m(e^{i\theta})\right| \qquad \text{Lemma 10}$$

Recall the initially stated problems in (2.4.2)? We don't know the phase of the noise. Therefore we are forced to subtract the powers based on the PSD. After that we just leave the phase unchanged. This can be accomplished by:

$$\hat{S}(e^{i\theta}) = S(e^{i\theta})\cdot\frac{\left|S(e^{i\theta})\right| - \left|\hat{N}(e^{i\theta})\right|}{\left|S(e^{i\theta})\right|} = S(e^{i\theta})\cdot\left(1-\frac{\left|\hat{N}(e^{i\theta})\right|}{\left|S(e^{i\theta})\right|}\right) = S(e^{i\theta})\cdot g(e^{i\theta})$$

$g(e^{i\theta})$ can be negative if the estimated noise exceeds the input signal (in spectral domain). We are therefore required to limit $g(e^{i\theta})$. This can be stated as in Lemma 11.

$$g(e^{i\theta}) = \max\left(\lambda, 1-\frac{\left|\hat{N}(e^{i\theta})\right|}{\left|S(e^{i\theta})\right|}\right) \qquad \text{Lemma 11}$$

Lets choose epsilon (very small possitive floating point number), for the sake of the argument.

This however only solves one of the initially stated problems. The problem with over/under estimates is left unhandled as the test results have shown sufficient results.

2.4.7 Results

When evaluating spectral subtraction we look at:

- Signal noise ratio
- Impact on VAD (Voice Activity Detection)
- Audible perception
- Impact on classification results

2.4.7.1 Signal noise ratio

It is notoriously hard or impossible to measure the signal noise ratio in recorded speech with no prior knowledge of the pure speech signal (often referred to as the desired input signal). We are interested in knowing the signal to noise ratio both before and after spectral subtraction.

To make a crude estimation we use a single recording containing only silence (noise) in first part and speech + noise in second part as in Figure 28.

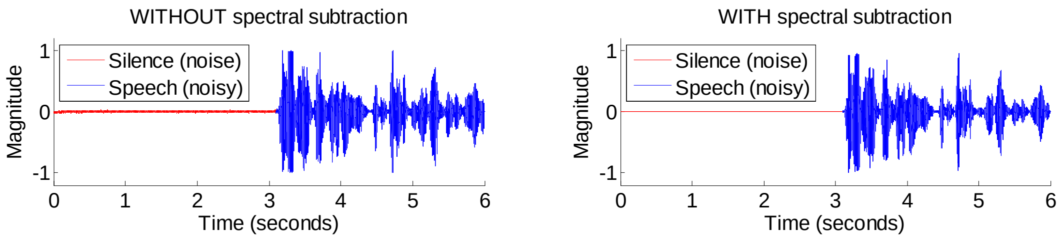


Figure 28 - Before and after spectral subtraction

We use the method described in Lemma 12 to compute the signal to noise ratio in dB scale.

$$eS = \sqrt{\left(\sum_{j=1}^{K} speech_n\right)^2}$$

$$eN = \sqrt{\left(\sum_{j=1}^{K} silence_n\right)^2}$$

$$SNR = 20 \cdot \log 10\left(\frac{eS}{eN}\right)$$

Lemma 12

The results (using best parametric settings) is seen in Figure 29

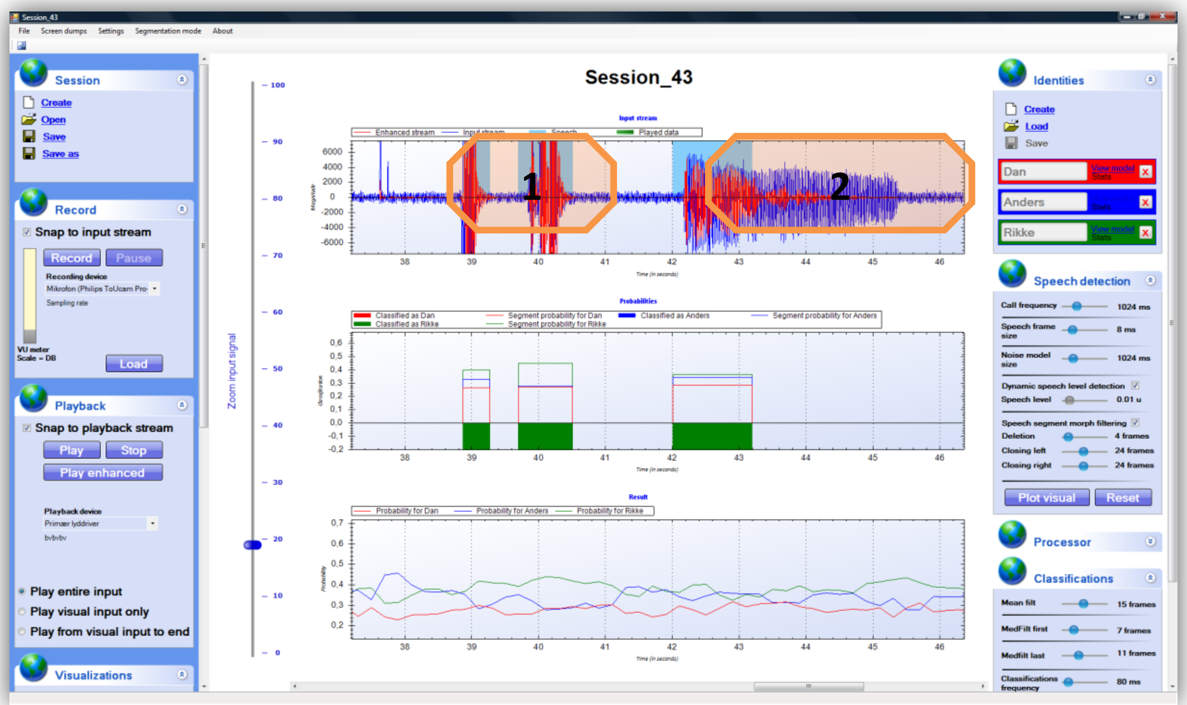
	Without spectral subtraction	With spectral subtraction
eS (dB)	36.6906	33.4546
eN (dB)	1.4847	0.10812
SNR (dB)	27.8586	49.8106

Figure 29 - Signal noise ratio before and after spectral subtraction

From Figure 29 we can see that spectral subtraction greatly increase the signal to noise ratio. The importance of this is evident when we later look at the impact on classification results (2.4.7.4).

2.4.7.2 Impact on VAD (Voice Activity Detection)

To look at the impact on VAD we need to turn to the application we have developed for this project.



Call frequency 1024 ms
Speech frame size 8 ms
Noise model size 1024 ms
Dynamic speech level detection ☑
Speech level 0.01 u
Speech segment morph filtering ☑
Deletion 4 frames
Closing left 24 frames
Closing right 24 frames
Mean filt 15 frames
ModFilt first 7 frames
Modfilt last 11 frames
Classifications frequency 80 ms

Figure 30 - Screen dump of spectral subtraction influence on VAD

The *red* line graph is the signal after Potsband filtering and spectral subtraction. It is clearly visible from the block labeled “1”, that the background noise is removed as expected in the first half of the signal.

The sound in block “2” is actually the “hummmm” from a vacuum cleaner. As the vacuum cleaners hum is somewhat constant from a frequency spectrum point of view (Figure 31) the minimum buffer is soon filled with this particular high powered pattern which is therefore estimated to be noise. We can see how the spectral subtraction algorithm gradually adapts to the new noise environment. Notice that it takes approximately 1 second as this is the buffer size we use for this example (the fluctuant nature of the startup of a vacuum cleaner makes the transient state a little longer in reality).

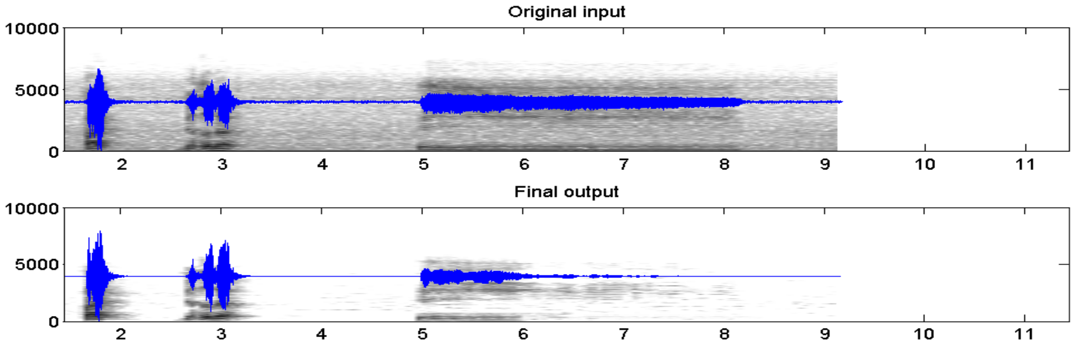


Figure 31 - Specgram of high power noise

2.4.7.3 Audible perception

This is of cause hard to illustrate on paper, but the 3 persons at scene when this result was made, agrees after a playback of both the original and the improved signal, that the difference is major.

In the original signal the noise is clearly audible at all times.

In the improved signal the noise is only audible in the short time (1½-2 seconds) when the vacuum cleaner is turned on.

2.4.7.4 Impact on classification results

Let's start out with a preview of how the noise removal affects the final classification results, just to emphasize the importance of noise removal. Optimal parametric combination is used for both cases. We have tested on more than 2 minutes to ensure the result is stable.

In Figure 32 we can see that the classification error is much greater when not using spectral subtraction. Best results are around 25% correct classifications *without* spectral subtraction and around 99% *with* spectral subtraction on ELSDSR database samples based on Neural Network classifications.

The main reason is that later processing is much better at detecting useful voice activity features since we don't classify on all data but only data containing voiced speech. But also the noise in the features used for classification plays a role of cause.

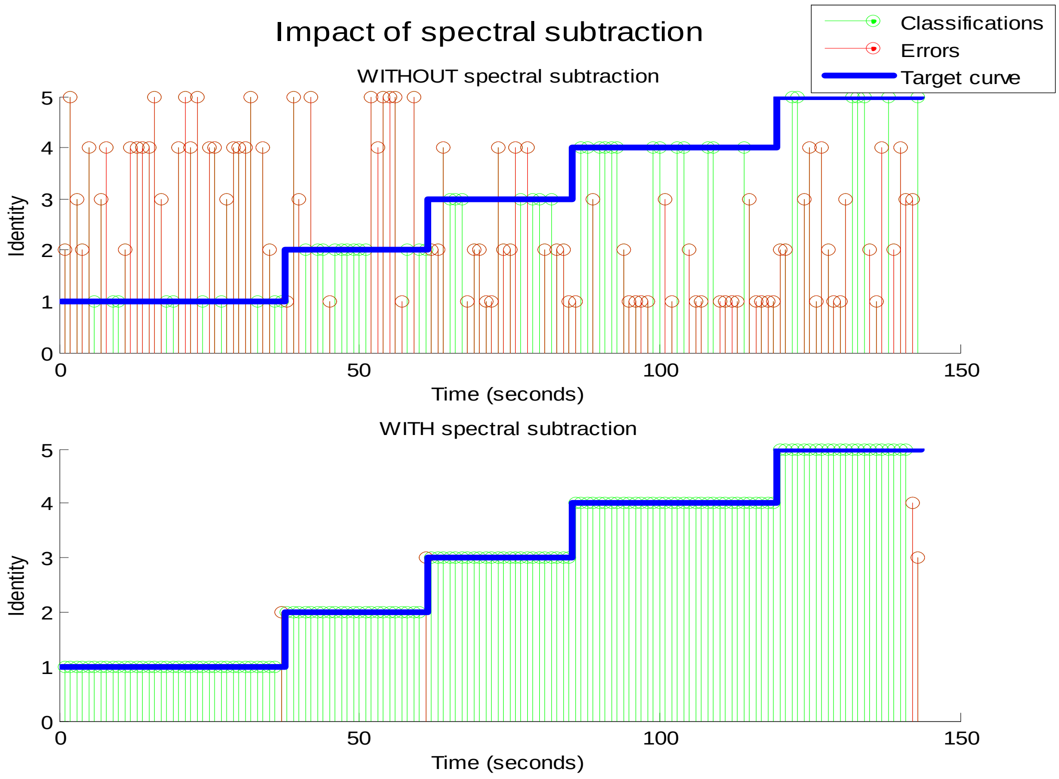
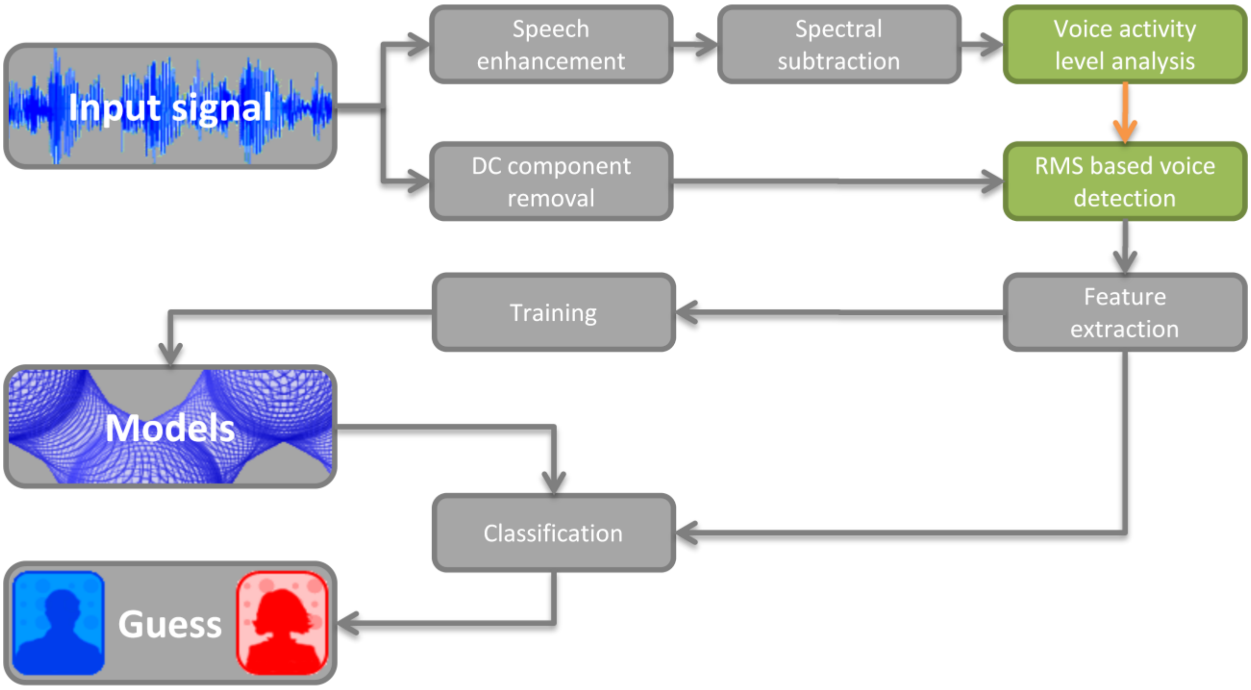


Figure 32 - Impact of spectral subtraction on classification results

3) Voice activity detection

Voice activity detection (VAD) is the process of detection speech segments in an input signal.



VAD is a thoroughly covered subject mainly due to its importance in the telephone industry (e.g. in minimizing bandwidth usage). Although several approaches have been suggested, none have yet proven to be truly robust for varying environments.

To further add to the challenge we only have single observation recordings (1 microphone) as opposed to humans that have two observation points (two ears). At least in most cases anyway… This removes the option to use independent component analysis which is known to improve VAD significantly.

3.1 Problem domain and approach

If the recording context of the input signals were known and had little or no variance in speech and noise levels, then it would be relatively easy to simply filter out noise and then set a predefined threshold between speech and silence.

It becomes more problematic when it is assumed that the speaker can move around relative to the microphone thus changing the loudness of the signal and the frequency range (often degrades when moving relative to the microphone). Also the noise level is likely to change over time due to events such as someone walking around or vacuuming in the next room and so forth.

This leads to the following expected problems:

- Detection of thresholds for speech/silence in varying audio environments.
- Make the model able to dynamically adapt to changes in speech/silence magnitudes over time.
- Make the model fast enough to be used in real-time applications.

Our aim is to develop a method for an indoor environment, typically a meeting. The noise characteristics expected to be present are:

- Static room ambient.
- Paper rattling, coughs etc. by meeting participants
- Hallway intrusive sounds such as vacuum cleaners, door knocking, background speech etc.
- Open window intrusive such as birds song, cars on street etc.

A second but just as important aim of our VAD is to be able to pinpoint parts of speech with a significant amount of speaker dependent content.

The approach used in this project is actually a combination of two different methods used in a 2-stage scheme. The reason for this should be evident after reading the following short explanation of the methods.

- **Voice activity level analysis**
  This VAD method is surprisingly robust in detecting the overall locations of speech in both high and very low SNR environments. It is also computational feasible for real-time use.
  The downfall is that it is very imprecise at defining the precise boundaries of speech. This means that it can’t detect the data most relevant for feature extraction either.

- **RMS based voice detection**
  This VAD method infeasible for detecting the speech segments from a streaming input signal mainly because it needs to have a significant amount of speech within its caches otherwise it fails. Although it would be computational feasible to let the caches span the entire input signal, this would disable its ability to adapt to changing SNR conditions.
  It is however very good at identifying the precise boundaries of speech within a small scope with most of the data being speech. It is therefore well suited to further process already discovered speech segments by another VAD method thus extending the voice activity level analysis.

3.2 Voice activity level analysis

3.2.1 The basics

The basic property of this method is to find the boundaries of speech segments. Our approach is strongly based on the ITU-T[9] recommendation P.56[10] for measuring the speech activity level.

In this method we define a speech segments as somewhat connected segments like expected from a dialog. This means that a speech segment includes structural pauses in speech. As some people tend to (rudely one might say) interrupt others while speaking this can result in speech segments containing speech from several speakers. It is therefore necessary to later divide a segment into sub-segments and process them in small frames. This is performed by: RMS based voice detection (3.3).

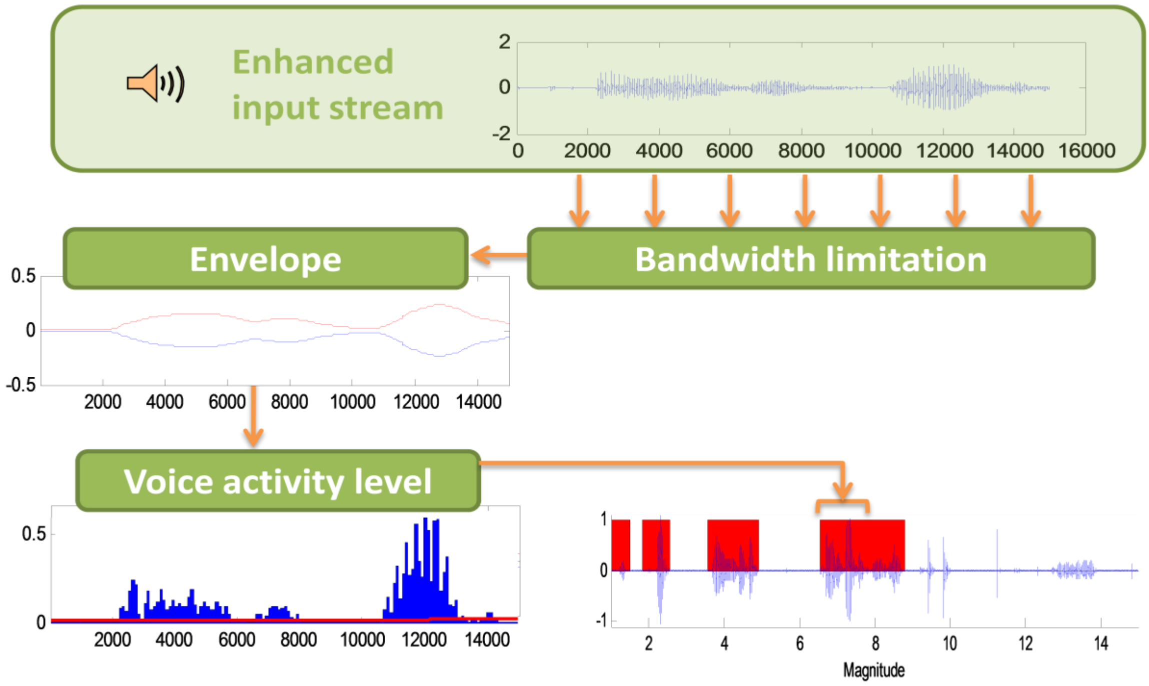


Figure 33 - Overview of VAD using PSD analysis

To keep it simple we first bandwidth limit the signal to 300-700 Hz to isolate first formant and increase computational speed.

Afterwards we perform a voice activity level analysis as seen in Figure 33.

Finally we connect high voice activity chinks by filtering. This is very effective at separating high pitched noise and humming from speech and form connected segments including structural pauses.

[9] ITU-T (International Telecommunication Union, Telecommunication Standardization Sector)

[10] http://www-mmsp.ece.mcgill.ca/Documents/Reports/1999/KabalR1999.pdf

3.2.2 *Formant frequencies analysis for bandwidth limiting*

It is almost a convention that the formant frequencies are important in speech processing. We use them to determine a bandwidth of interest with regard to measuring the voice activity level.

The formants (as opposed to e.g. fundamental frequencies) are produced by the vocal tract.

This is relevant because what we are really interested in is regions of the input signal which contains a high power of speech which is near the first couple of formant frequencies. This should give a good indication of wetter the actual part of the input signal is speech.

Figure 34 shows how the 2 first formants F1 and F2 relate to the vowels.

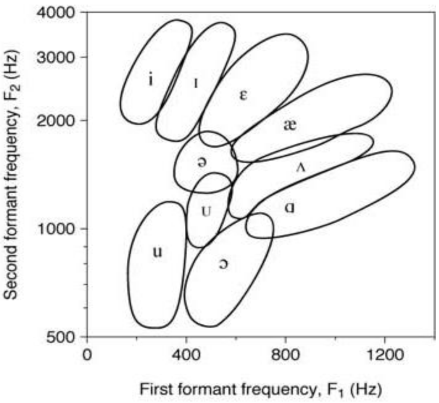
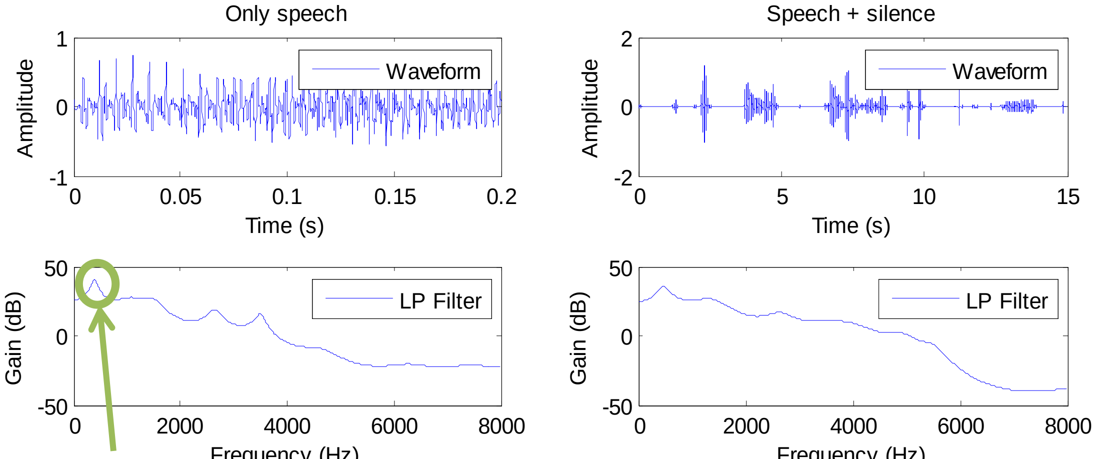


Figure 34 - Formants in vowel domain

3.2.2.1 Properties of formants

Figure 35 illustrates a 2 waveforms and long term predictions. The graphs to the left are speech only, and the graphs to the right contain speech, silence and noise.



Peak in magnitude

Figure 35 – Waveform and LPC for finding formant frequencies

From a LP filter analysis it is possible to find the formant frequencies. In short they are estimated by finding roots[11] between 0-8 kHz (because of Nyquist interval). The main principle is that a pair of poles (Lemma 13) has a magnitude peak as marked in Figure 35. This peak is situated in an angular frequency.

$$\frac{1}{\left(1-re^{j\theta}z^{-1}\right)\left(1-re^{-j\theta}z^{-1}\right)}=\frac{1}{1-2r\cos\left(\theta\right)z^{-1}+r^{2}z^{-2}}$$

Lemma 13

[11] http://www.cs.tut.fi/kurssit/SGN-4010/LPsovellus_2004_en.pdf

$$F_s \frac{\theta}{2\pi}$$ Lemma 14

From Lemma 14 we can see that if the angle θ is steep then the bandwidth must be small. We can compute a bandwidth of a pair of poles from the LP filter by Lemma 15.

$$-F_s \frac{\ln r}{2\pi}$$ Lemma 15

Now we can estimate the formants. There are a couple of suggested ways to do this but we have chosen to use factorization as in Lemma 16.

$$a(1)z^{-1} + \cdots a(p)z^{-p}$$ Lemma 16

The formant frequencies are angles of the roots. To find the roots we use the Newton-Raphson algorithm for two reasons:

1. We have used this particular method extensively in a Numerical Algorithms course so we have already implemented this method.
2. Newton-Raphson performs reasonable fast if we can give some good start guesses which is possible as we just use the previous found roots as starting conditions.

Only speech	Speech & silence
Formant 1 Frequency 394.8	Formant 1 Frequency 467.5
Formant 2 Frequency 1079.5	Formant 2 Frequency 1270.6
Formant 3 Frequency 1525.7	Formant 3 Frequency 1430.4
Formant 4 Frequency 2652.9	Formant 4 Frequency 2654.5
Formant 5 Frequency 3492.9	Formant 5 Frequency 3557.4
Formant 6 Frequency 3802.1	Formant 6 Frequency 4075.2
Formant 7 Frequency 4665.4	Formant 7 Frequency 4916.7
Formant 8 Frequency 6289.8	Formant 8 Frequency 5522.7
Formant 9 Frequency 7443.8	

Figure 36 - Formant frequencies

This produces the results in Figure 36. When investigating the formants it is clear that (although the values differ) there is a trend in the location of the formants independent of the input signal as long as it contains a significant part of speech. If we look at the results for F1 and F2 in Figure 36 and compare them to the vowel domain of F1 and F2 in Figure 34 we can see that the found F1 and F2 are located at the centre of this domain. Typically it is assumed that the first formant F1 is centered in the interval 300-700 Hz and the second formant F2 is centered in the interval 900-1300 Hz[12].

[12] http://en.wikipedia.org/wiki/Formant

3.2.2.2 Selecting bandwidth limit

Let's start with looking at the input signal where the authors have noted what the actual contents of the signal are. This is seen in Figure 37.

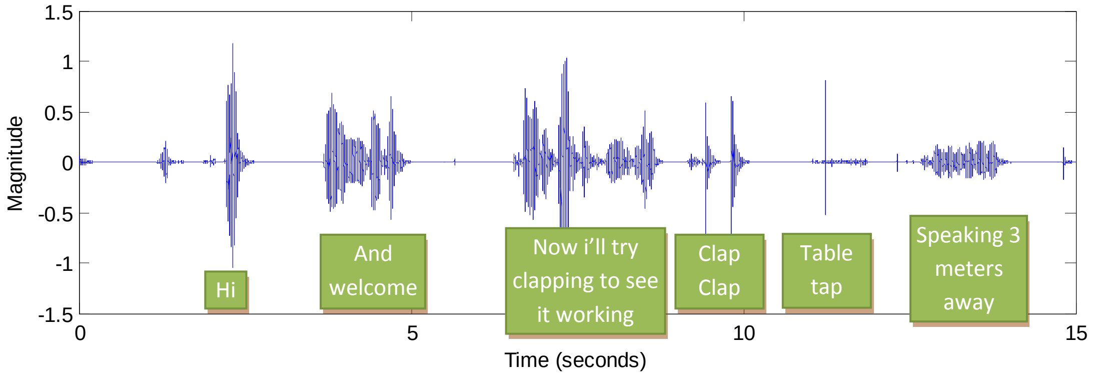


Figure 37 - Annotation of input signal

Then let's look at the difference of the Power distribution in frequency domain when using 1 formant (300 Hz - 700 Hz) or 2 formants (300 Hz - 1300 Hz) respectively. The PSD when using either F1 or F1+F2 is shown in Figure 38.

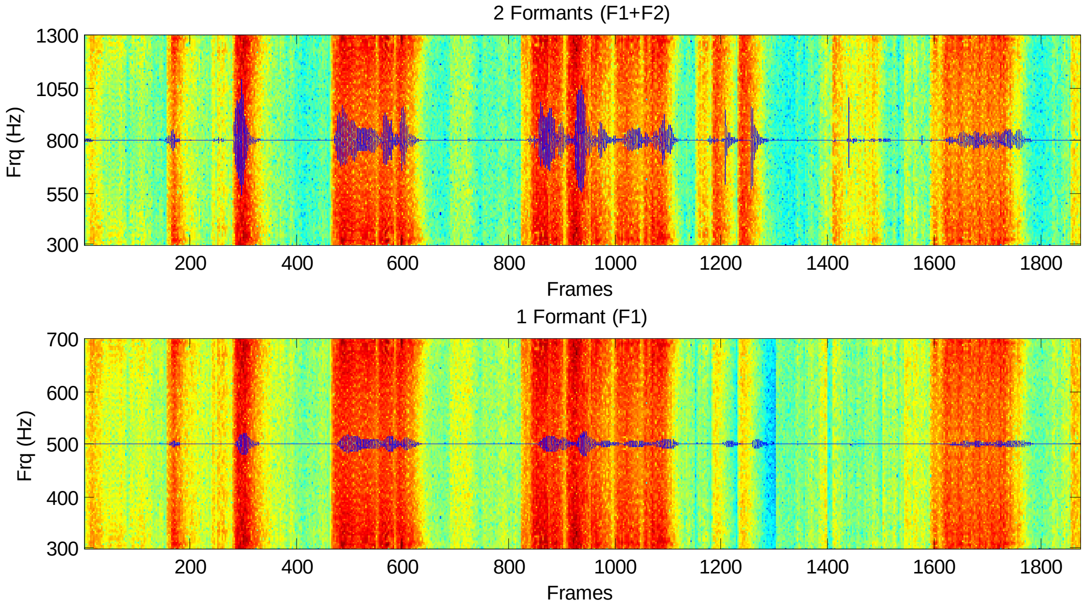


Figure 38 - PSD using F1 and F1+F2

From Figure 38 it is clearly visible that the speech part of the signal is more distinct when only using the bandwidth of F1 as opposed to using F1+F2.

For this reason we use the bandwidth of 300 Hz – 700 Hz in the following processing.

3.2.3 Smoothing envelope

As earlier stated speech contains pauses between utterances of words. These are referred to as structural pauses.

The purpose of the envelope is to allow the inclusion of structural pauses in the active speech measurement but at the same time avoid the inclusion of short time "noisy" pulses such as claps etc.

There are some different types of envelopes that can be computed. The one we are interested in is called the "delayed smoothed envelope". This envelope fulfills the purpose as opposed to the instantaneous types of envelopes which does not avoid hangover from short noise pulses.

3.2.3.1 Final filter conditions

The smoothing envelope is constructed by utilizing the final conditions of the previous filter state in steps. When calling the filter again we use the previous final filter state condition as an input to adjust the filter state. Suppose the static filter in Figure 39.

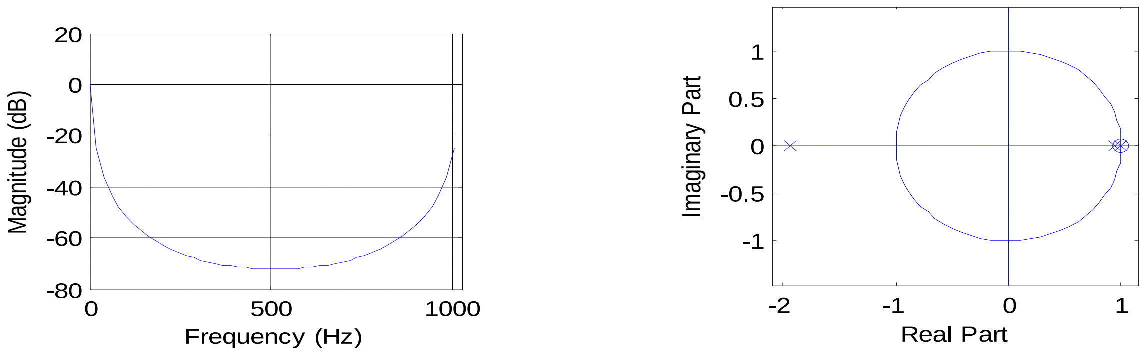
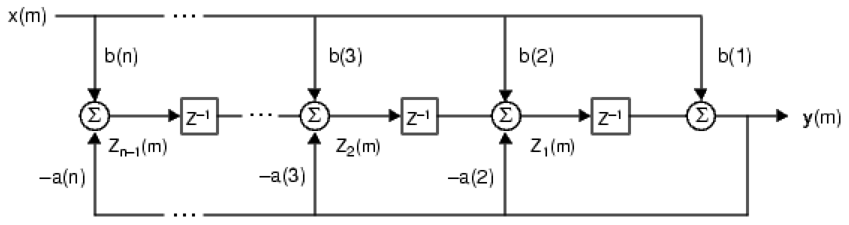


Figure 39- Smoothing filter

The filter is based on these normalized coefficients (stated by ITU-T P.56) but adjusted to the downsampled frequency range.

$$\begin{bmatrix}1 & -1.94 & 0.94\end{bmatrix}$$

When looking at the filter diagram as seen to the right this means that we can compute the filter time domain difference equation for the final filter state by Lemma 17.



$$z_{n-1}(m) = b(n)\cdot x(m) - a(n)\cdot y(m)$$

Lemma 17

This means that the final description of this filtering operation in the z-transform domain is a transfer function which can be computed as in Lemma 18.

$$Y(z) = \frac{b(1) + b(2)z^{-1} + \cdots + b(nb+1)z^{-nb}}{1 + a(2)z^{-1} + \cdots + a(na+1)z^{na}} X(z)$$

Lemma 18

3.2.3.2 Computing the delayed smoothed envelope

To acquire the delayed smoothed envelope we can compute it as in Lemma 19.

$$p_i = gp_{i-1} + 1(1-g)|x_i|$$
$$q_i = gq_{i-1} + 1(1-g)|p_i|$$

Lemma 19

Note that the envelope is computed on a signal which is bandwidth limited to the first formant F1. We expand it however when plotting for ease of sight.

The variable p and q are the computed envelopes at two different delay states.

The variable g is a scaling parameter. It is computed as a relative between the size of each block in which we make an estimate called T (typically around 20-40 ms) and the interval in which we sample called t. It is computed as in Lemma 20.

$$g = e^{-t/T}$$

Lemma 20

The envelope q is shown in Figure 40 where the first graph is a close up of the second graph.

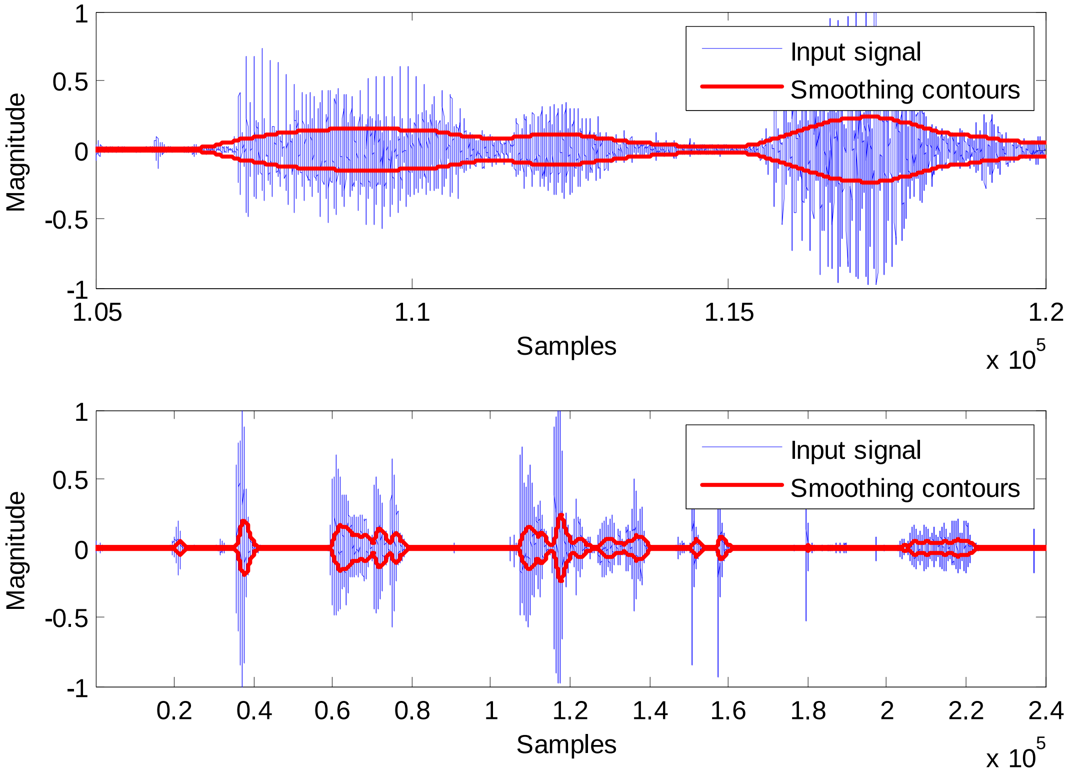


Figure 40 - Effect of smoothening filter

It is also visible from Figure 40 that the 2 peaks at approximately 1.5-1.6 +5e are far less dominant than expected. This is due to bandwidth limiting and the use of delayed smoothing envelope.

3.2.4 Speech activity level computation

Within the envelope from the previous section it is possible to compute the number of times the smoothed envelope is distributed within specific activity levels. This is done by creating an activity counter called a_j.

For each sample in the smoothed envelope q_i we compare it to a set of thresholds called c_j. The thresholds in c_j is suggested by ITU-T to be 15 values as a power of 2 which means that:

$$c_1 = 1 \quad c_2 = 2 \quad c_3 = 4 \quad \cdots \quad c_{15} = 16384$$

The activity counts can thus be computed as in Lemma 21 by iteration over i and j.

$$\begin{aligned} &\text{if } q_i \geq c_j \text{ then} \\ &\qquad a_j \pm 1 \end{aligned}$$

Lemma 21

We now use the accumulated activity count a_j and the enveloped values in q. As we need to find the activity levels in a streaming input we do this in segments of N samples.

By ordering the envelope values in q in incremental order and use them as thresholds h_j it is now possible to compute the activity level as a function of the assumed speech level l as by Lemma 22.

$$a(l) = \begin{Bmatrix} 1 & , & \text{for } \; l < \min(c_j) \\ a_j / N & , & \text{for } \; \mathrm{c}_{j-1} \geq l < c_j \\ 0 & , & \text{for } \; l < \max(c_j) \end{Bmatrix}$$

Lemma 22

As N is increasing over time and thus also c_j we need to do this in steps to avoid excessive computations. This means that we compute the speech activity in steps of a predefined interval e.g. 0.03 ms.

The result is visible in Figure 41 where it is clear that the peaks which is actually “claps in the hands” by the speaker is not included.

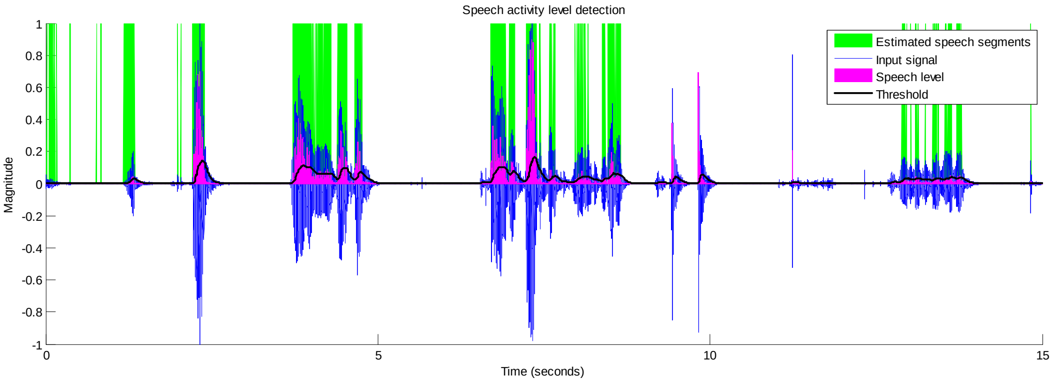
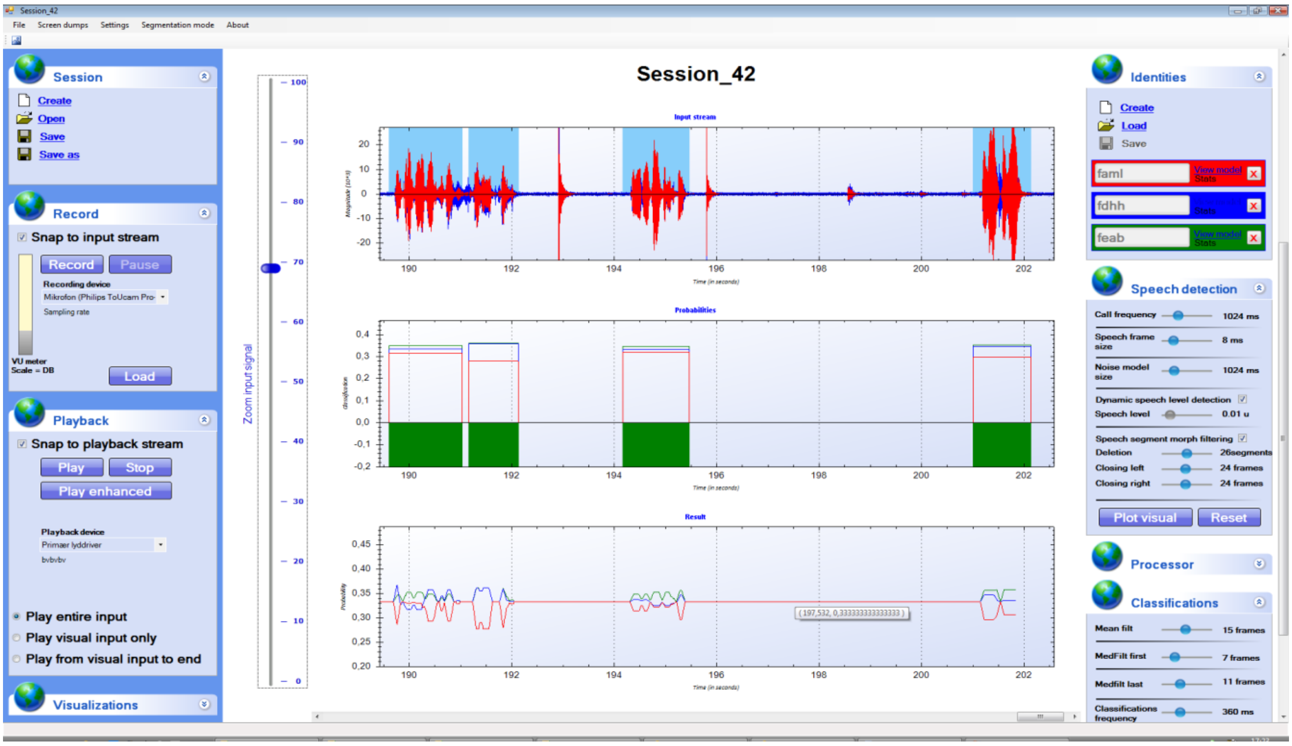


Figure 41 – Result SAL based VAD detection

3.2.5 Results

As we are interested in connecting the speech segments such that they include all the speech we apply a maximum filter to connect them. The maximum filter length is empirically chosen (and is adjustable from within the Win32 application). From the example displayed in Figure 42 (screen dump of the Win32 application) we can see a couple of things.

- The peaks at time 193 and 196 are not included. These peaks are actually “claps in the hands”.
- The small fluctuations at time 197 are actually a “hrmm sound”.
- The first 2 speech segments would have been connected if the analysis was done on the raw input signal and not on the first formant frequency range F1.
- The last speech segment is connected although there is a small gab in between. It is because the pause is a structural pause (a pause between utterances).
- We only classify on speech segments as seen in the bottom graph.



☑ Snap to input stream
Recording device: Mikrofon (Philips ToUcam Pro-
Sampling rate
VU meter
Scale = DB

☑ Snap to playback stream
Playback device: Primær lyddriver
bvbvbv

● Play entire input
○ Play visual input only
○ Play from visual input to end

faml
fdhh
feab

Call frequency 1024 ms
Speech frame size 8 ms
Noise model size 1024 ms
Dynamic speech level detection ☑
Speech level 0.01 u
Speech segment morph filtering ☑
Deletion 26segments
Closing left 24 frames
Closing right 24 frames

Mean filt 15 frames
MedFilt first 7 frames
Medfilt last 11 frames
Classifications frequency 360 ms

Figure 42 - Screen dump of maximum filtered speech segments

Extended results have be achieved by running the application and visually denote insertion and deletion errors. We have discovered the following results based on webcam microphone recordings and ELSDSR database respectively based on approximately 80 seconds.

	Webcam based input stream	ELSDSR database
Insertion errors % of segments count	4 %	0-1 %
Insertion errors % of segments length	0-1 %	0-1 %
Deletion errors % of segments count	2 %	1 %
Deletion errors % of segments length	2 %	0-1 %

3.3 RMS based voice detection

3.3.1 The Basics

A speech segment found by the voice activity level analysis may contain silence inside the segment. This can be due to either a speaker change or a structural pause by the speaker. These small but important pauses can be considered unlabeled data. A foreseen and well known problem can arise when training classification models (e.g. Neural Networks) on unlabelled data. The models can converge towards this data unlabelled data. Therefore this method for detecting the actual speech frames inside a speech segment is needed.

Please note that the speech segments used during this section are larger than the typical ones found in a "normal" conversation. It has no effect from a method point of view. We only use large segments for explanatory reasons.

This proposed method for VAD I is derived from NIST[13] although it differs in usage. The benefit of using this particular method over others (for instance zero-crossing) is its speed and dynamic nature**.**

The idea is, as seen in Figure 43, to divide a speech segment into small frames. Based on a number of frames a histogram is created where each frame updates the appropriate bin. This is continuously done over time by adding and removing root mean square power (RMS) values to and from the appropriate bins.

The histogram can now be used to decide whether a frame is silence or speech. The sharp low energy peaks marks out a lower limit of where the silence level is expected to be. This is due to the high frequency low magnitude of silence. When speech is present the magnitude increases and lower frequencies are more present in the signal. A method like this produces a bi-modal histogram which is exactly what is desired.

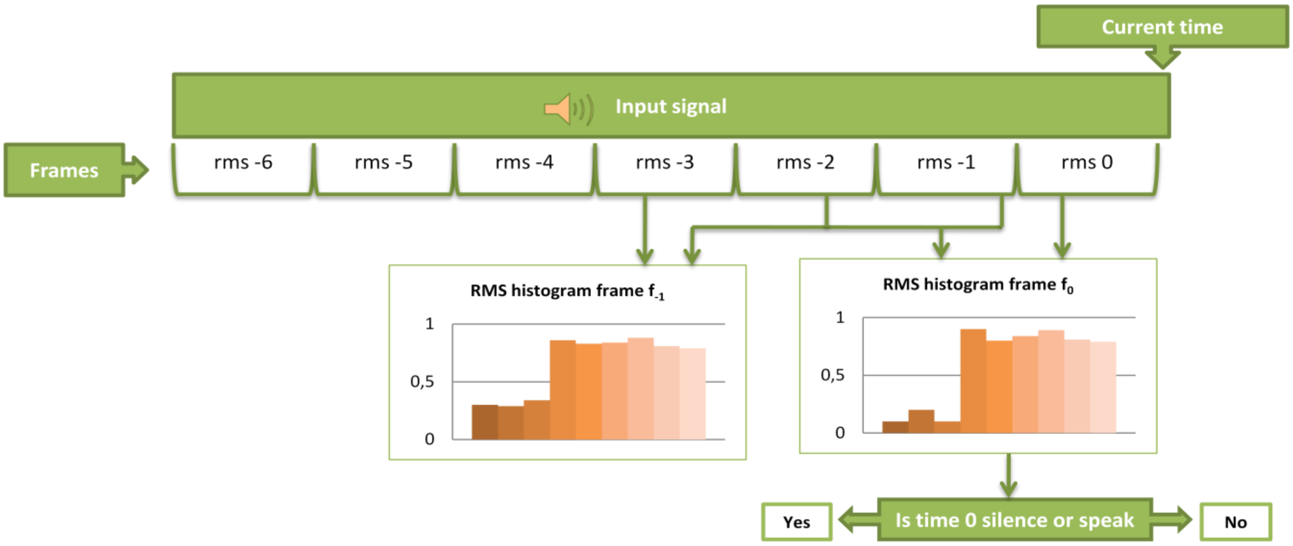


Figure 43 - Princip of RMS based voice detection

[13] Casimir Wierzynski and Jon Fiscus, "stnr.doc" included with the NIST Speech Quality Assurance Package Version 2.3.

3.3.2 *The test input signals*

The speech segments used for analyzing this method is approx. 10-15 seconds long with a sampling frequency at 16000. A set of 10 different speech segments are used for this analysis. Each of the 10 speech segments contains several combinations of noise as shown in Figure 44.

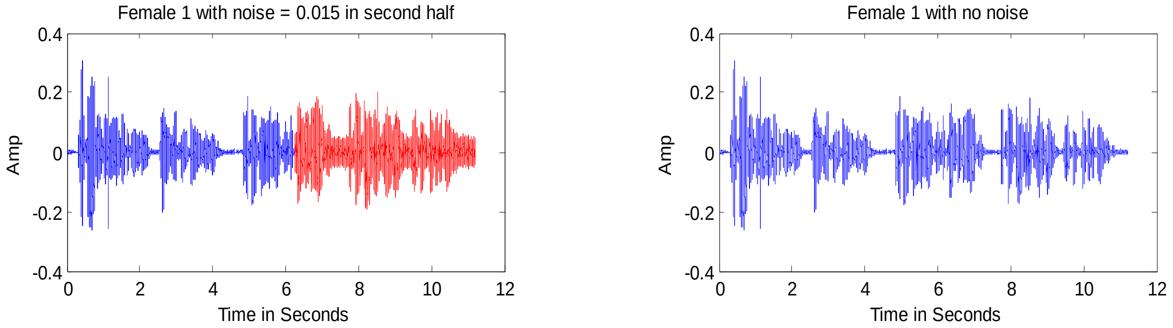


Figure 44 - Example of long speech segment

3.3.3 *Root Mean Square Power*

First a suited frame size is chosen in which the signal is somewhat stationary (typically frame sizes of 10 - 40 milliseconds). The latter frame size gives faster computation with only small loss in precision its better.

3.3.3.1 Why use RMS?

The main argument for using RMS is that what we really want is a term that expresses the variations inside the speech segments and can be computed fast.

Let's look at RMS' relation to the standard deviation in Lemma 23.

$$x_{rms}^2 = \overline{x}^2 + \sigma_x^2$$

Lemma 23

From Lemma 23 we can conclude that RMS will always be equal to or greater than the average since it includes the "error" as well as the standard deviation. This means that RMS offers a true average of the root mean square power of multiple values in a single frame.

As this term only includes information about a short frame of time, it can by itself not be used to make any decisions as whether a frame is speech. For this purpose it must be used in conjunction with RMS values from other frames.

3.3.3.2 Computing RMS values

For each frame, the RMS is computed as in Lemma 24.

$$x_{rms} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2} = \sqrt{\frac{x_1^2 + x_2^2 + \cdots + x_n^2}{n}}$$

Lemma 24

3.3.3.3 A problem with DC component and RMS

The RMS value for a given frame is dependent on the DC-component. To obtain a uniform RMS value distribution base it is therefore necessary to remove it. This dependency is illustrated in Figure 45.

Samples	μ (mean)	RMS value
[1.5 ; 0.5]	1	1.25
[- 1.5 ; -0.5]	0.5	1.25
[0.5 ; -0.5]	0	0.25

Figure 45 - RMS dependency on DC component

As seen in Figure 45, the RMS value is clearly dependant of μ or the DC-component. This is a problem, because each RMS value is used in a histogram in conjunction with other RMS values to automatically estimate the threshold between silence and speech.

The obvious way to deal with this is to remove the DC-component. This is the reason why we use DC component removal in the front-end signal processing phase.

3.3.4 Histogram of frame based RMS values

By using the RMS from a consecutive number of frames we can create a histogram of these. This histogram is then used to determine a threshold between silence and speech.

3.3.4.1 Histogram size

It is important to notice, that the number of frames included in this histogram must be sufficient to catch the overall variations in the signal. This is typically true for speech segments of +2 seconds. Anything less than that will be sensitive to local extremes. I will still work although the accuracy drops significantly.

This means that if the speech segment being analyzed is smaller than 2 seconds, we include previous non-speech data thus utilizing a minimum of 2 seconds.

Any number of bins for the histogram can be used as long as it is greater than 1. Using 4-10 bins has empirically provided robust results.

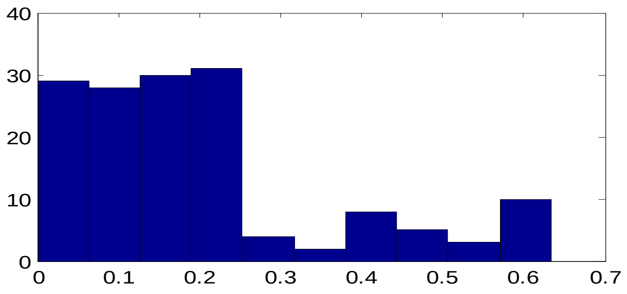


Figure 46 - RMS energy histogram

The histogram based on the RMS values will be so called bi-modal. It contains a sharp low-energy mode and a flatter high-energy mode.

3.3.4.2 Locating speech

It is the bi-modal property of this histogram that enables us to estimate a threshold for speech.

It was initially assumed that the threshold would be found at the transition point in the bi-modal histogram (approximately 0.23 Figure 46) but this turned out not to be true. The threshold is found at the center of the low-energy mode. This is approximately at 0.1 in the example shown in Figure 46.

How to extract threshold from bi-modal histogram

The first task is to divide the bi-modal histogram into its two sub-modals. The obvious solution is to look at the first order derivative of the energy histogram (Figure 47).

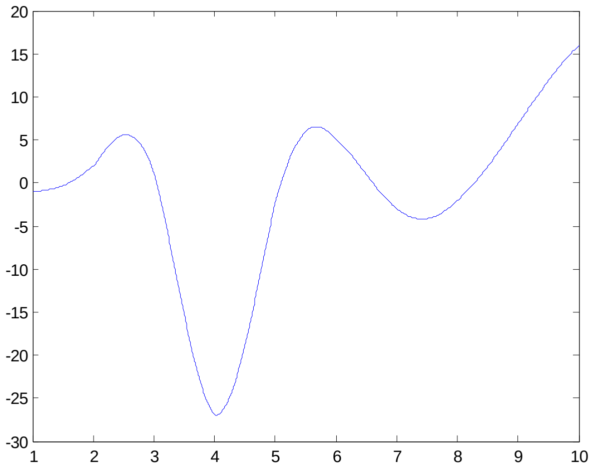


Figure 47 - First order derivate of energy histogram (inpterpolated for clarity)

The minima can then be extracted. In this case the minimum equals 4 which is the bin number of the transition point between the two sub-modals in the bi-modal histogram. As the bi-modal transition point is now known, the threshold is found at the center between the location of the first bin and bin 4 (in this example).

The threshold found for this example equals 0.0975 in correspondence to Figure 46.

Appendix E shows an example on how the histogram fits to changes in SNR ratio.

3.3.4.3 Avoiding re-computing values

Another benefit of using this method is that a histogram can be dynamically updated to avoid re-computing any values.

Suppose a speech segment is divided into frames each containing 320 samples (which is equivalent of a signal sampled at F_s = 16000 with a framesize of 20 ms.). The stepsize is set to 160 samples.

If we want to estimate a silence threshold based on the previous 1 second (which would include 100 frames) we could make an equally long vector, which serves as a container for the RMS values of these past frames. Whenever additional 10ms has passed, it is simple to remove the oldest element of the vector and compute a RMS value for the frame which is then added to the vector.

By subtracting the outdated RMS value and adding the newest to the histogram as in Figure 48, the histogram can be updated every 10 millisecond at the cost of computing the new RMS value. This means that no RMS values are recomputed.

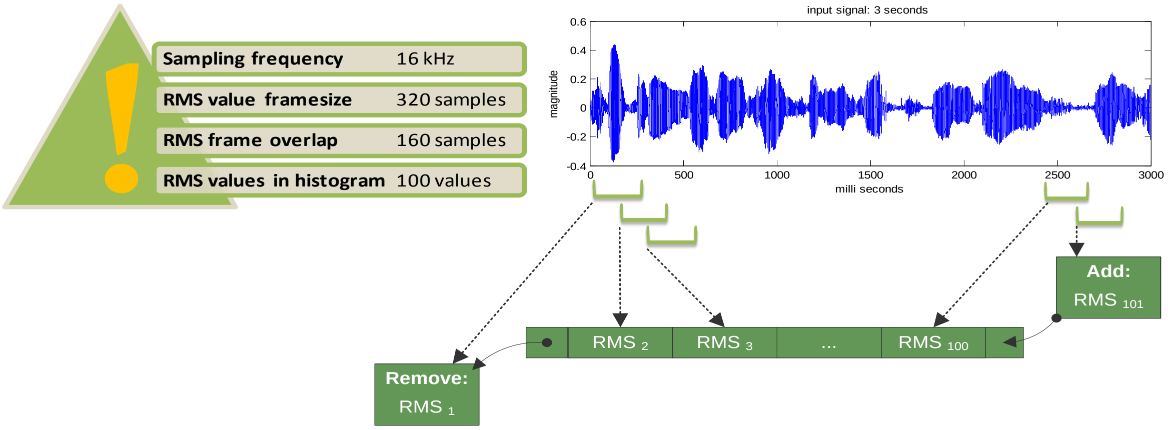


Figure 48 - RMS cache

3.3.5 Time complexity

When computing the threshold for silence the following factors are decisive.

The fixed size operations constants such as additions and subtractions are neglected.

Dependancies

- s = Frame size (samples in each frame)

The computational time complexity for pr. frame

- Mean removal: $O(s)$
- RMS computations: $O(s)$
- Histogram updates: *No BigO cost*

3.3.6 Results

Although this method is used on speech segments with an enhanced signal, we also put it to the test in noisy conditions to measure its performance under different circumstances.

It is hard to make an absolute measure of how good a voice detection method is as the result is dependent on the input signal. One measure is audible listening to the input signal after it has been cleared of silence.

A good result would be when all the words spoken by the speaker is clearly understood and there are no pauses in speech at all. Another measure is the distance of features extracted from the voice detected for one particular compared to features extracted from voice detected for another speaker. This is elaborated in the section of Feature extraction.

Female voice with no noise

As seen in Figure 49, the method clearly detects dynamic threshold values that detects the silence or pauses in speech. The speech is clearly understandable after removing theses silence frames. There are no audible pauses or silence periods in the signal.

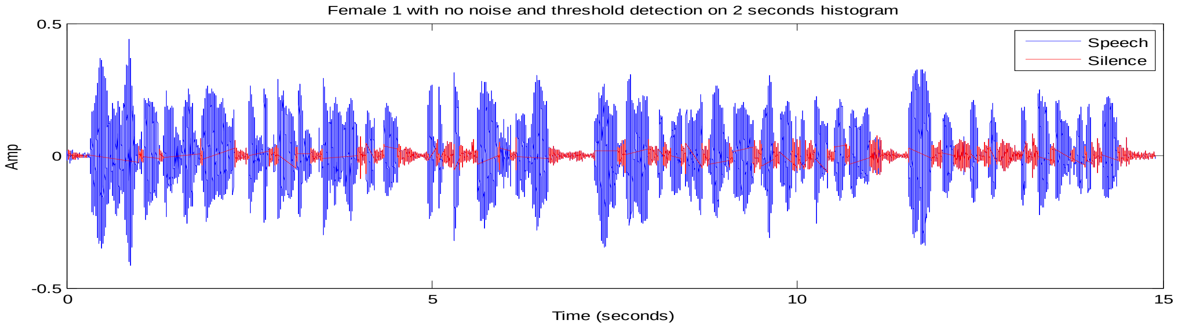


Figure 49 - RMS based voice detection with no noise

Female voice with noise = 20 % on last half of signal

As seen in Figure 50, the method quickly adapts to the changes in the noise at 20% occurring approximately after 7 seconds. The It sounds somewhat blurry in the last half due to the noise though. There are no audible pauses in the signal.

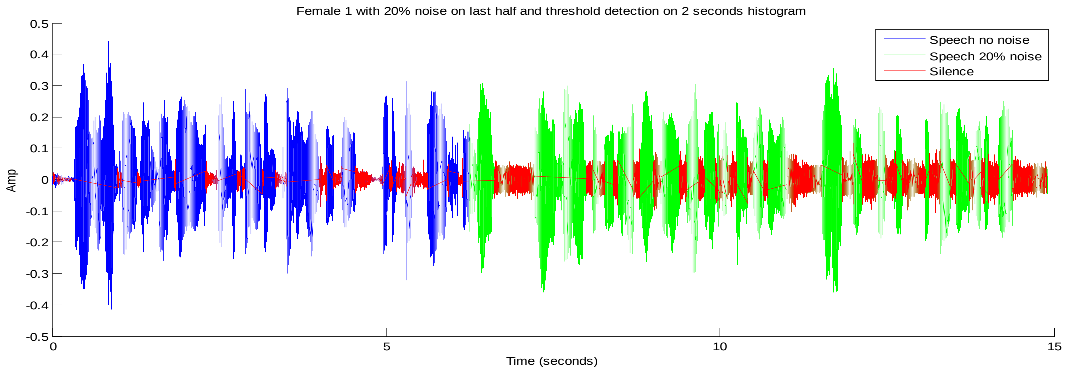


Figure 50 - RMS based voice detection with 20% noise

Female voice with noise = 60 % on last half of signal

When the noise level is increased significantly as in Figure 51, the method needs longer time to adapt to it. As in this case with 60% noise (almost sounds like someone is drilling near the microphone), the histogram from which the threshold is estimated, needs to fully cover the noise part in order to adapt to it.

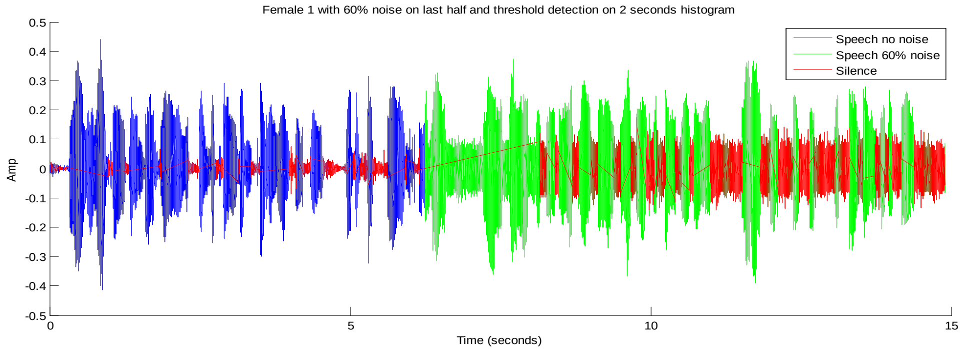


Figure 51 - RMS based voice detection with 60% noise

This is evident when looking at the first green chunk (at second 6-7). A similar phenomenon occurs when a human is listening to another human talk and a vacuum cleaner is turned on. Humans miss a short period of sound as well. The speech in the second half of the signal is hard to understand due to the noise excessive noise.

Reducing the number of frames in the histogram speeds up this adaptation to large noise variations, but makes the threshold estimate less precise. There are no audible pauses or silence periods in the signal. This situation is considered beyond normal use of speaker identification.

Summary

The method has been tested on a set of 10 males and females.

As shown in the illustrated examples, the method performs robust even when noise hasn't been removed very successfully.

The method is subject to time/precision trade-offs selected by the user/implementer.

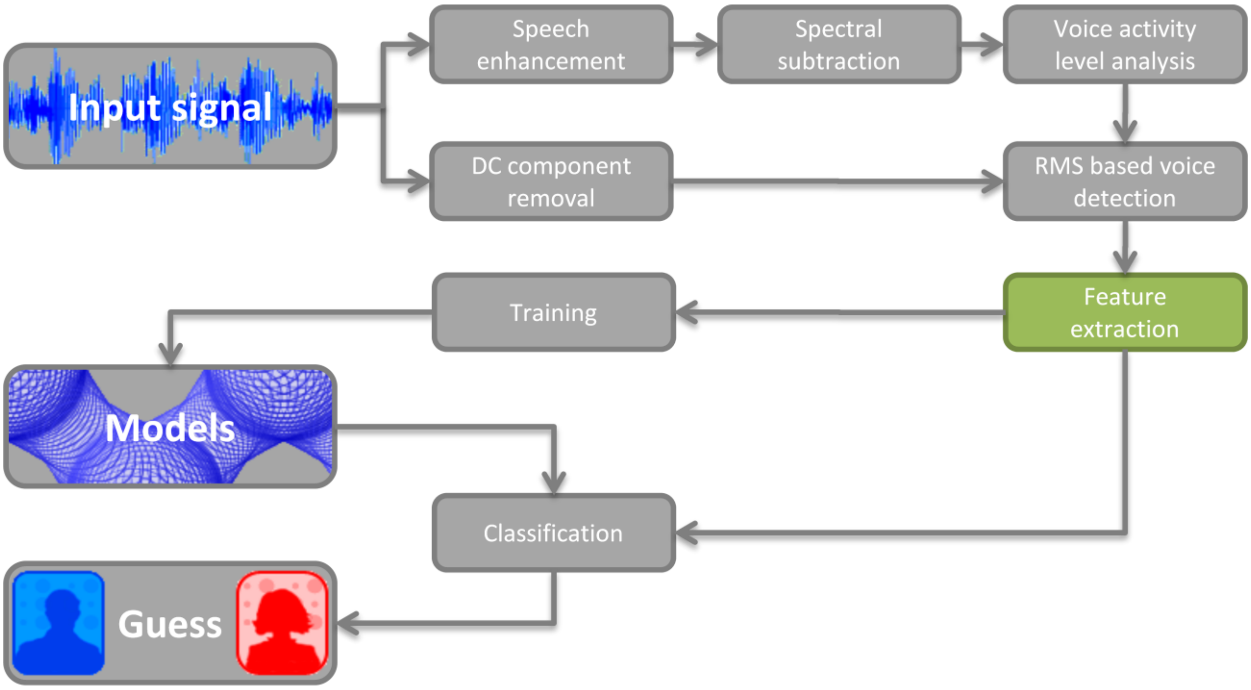
If signal/noise relation is poor the time it takes for the method to adapt is longer and vice-versa.

The method performs extremely fast with a linear relation to the sampling frequency only.

4) Feature analysis and extraction

4.1 Introduction

Feature extraction is essential in developing a robust speaker identification system.



The objective of feature extraction is to transform a speech signal into a number of variables (features/patterns) that represents the voice of individual humans in a relatively low-dimensional space.

These features are used in classification systems like Neural Networks (NN), Gaussian Mixture Models (GMM), the Hidden Markov Model (HMM) etc.

A number of important criteria's must be fulfilled to estimate good features:

- Easy to measure
- Specific for individual speakers
- Stable over time
- Reduced in dimensionality in relation to the original signal
- Not affected by ambient noise

4.2 Spectral analysis

4.2.1 The basics

A speech signal can be represented in the time domain by a vector of amplitudes $X = [x_1, x_2, \ldots, x_n]$. A Fourier Transform (FT) can be used to transform the amplitudes in the time domain into frequencies in the frequency domain.

The method of extracting the frequencies and examine the signal in the frequency domain is called a spectral analysis.

4.2.2 Short Time Fourier Transformation

Speech signals represented in the time domain shows the amplitude over time. This representation is rather poor as it won't describe specific information about individual speakers. Different speakers often use the same amplitudes (loudness) over time.

A better representation of a speech signal is obtained by a FT. In the frequency domain it is easier to distinguish between individual speakers. As an example the fundamental frequency for a female is often higher than the fundamental frequency for a male.

In the frequency domain the occurrence of the individual frequencies is not known. A Short Time Fourier Transform (STFT) is often used to detect at what time different frequencies occurs. The main idea of a STFT is to use a FT one smaller time intervals, called frames. This way we know the location of different frequencies as the signal changes over time.

The STFT is illustrated in Figure 52. Besides framing a STFT uses high-pass filtering, windowing and a Discrete Fourier Transform (DFT).

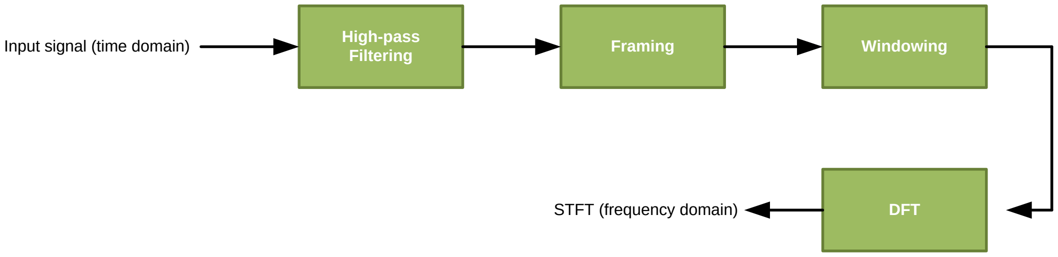


Figure 52 – STFT process

4.2.2.1 Spectral properties

In Figure 53 a spectrogram is used to show the effect of a STFT. The first graph shows the time domain input signal. The input signal contains 5 seconds of data and its amplitudes are located in the interval from -1 to 1. The second graph shows the spectrogram of the input signal.

The spectrogram illustrates the energy of each frequency as the signal changes over time. The input signal is sampled at 16 kHz and the spectrogram shows time at the x-axis and frequencies between 1Hz and 8KHz at the y-axis.53¨´2

In this example it is quite obvious that the input signal contains three letters ("D T U") .

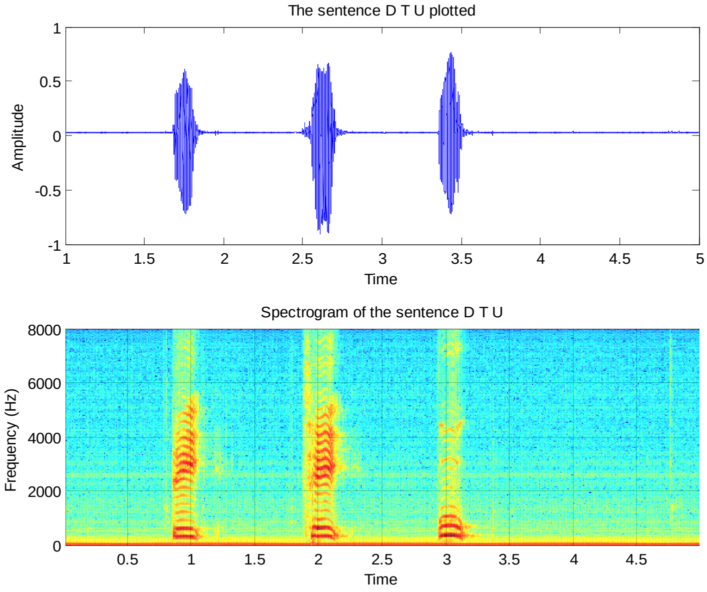


Figure 53 - STFT graphs

From the spectrogram it is obvious that the letter "T" contains higher frequencies than "U", which also appears by pronouncing the two letters.

In the subsequent chapters each individual step in an STFT is described in details. We focus on the purpose of high-pass filtering, framing, windowing and the DFT.

4.2.2.2 High-pass filter

A high-pass filter is a filter that passes the high frequencies in e.g. a speech signal and damps the lower frequencies. The reason for using a high-pass filter in relation to the STFT is to imitate the human voice where the vocal tract almost works as a high-pass filter.

In speaker identification a Finite Impulse Response (FIR) digital filter is often used. The FIR filter is generally based on Lemma 25.

$$H(e^{j\hat{w}})=1-\frac{18}{19}e^{-j\hat{w}}$$

Lemma 25

The effect of the above filter can be illustrated by converting the filter into the Z-domain as in Lemma 26

$$H(e^{j\hat{w}})=1-\frac{18}{19}e^{-j\hat{w}}\xleftrightarrow{z}H(z)=1-\frac{18}{19}z$$

Lemma 26

In the Z-domain the filter can be visualized by a zero-pole plot in Matlab. The zero-pole plot in Figure 54 illustrates that lower frequencies are damped (the circle) whereas the higher frequencies seams unchanged (the cross).

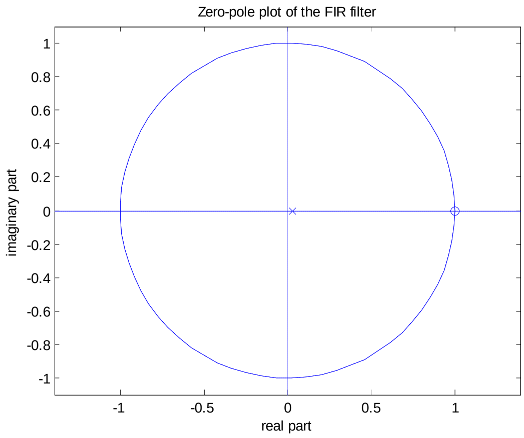


Figure 54 - Zero-pole plot of the FIR filter

4.2.2.3 Framing

A speech signal is non-stationary over time, but seen in short periods the signal is fairly stationary. To examine the signal in the frequency domain a Discrete Fourier Transform (DFT) can be used, but due to the fact that this transform performs best on stationary signals one has to block the signal into frames.

Another important factor is that framing ensures that we know the location of different frequencies as the signal changes over time.

In Figure 55 we have illustrated how framing is used on an input signal. The input signal is the sentence ("D T U") that we used earlier. Speech is detected in the signal before it is blocked into frames.

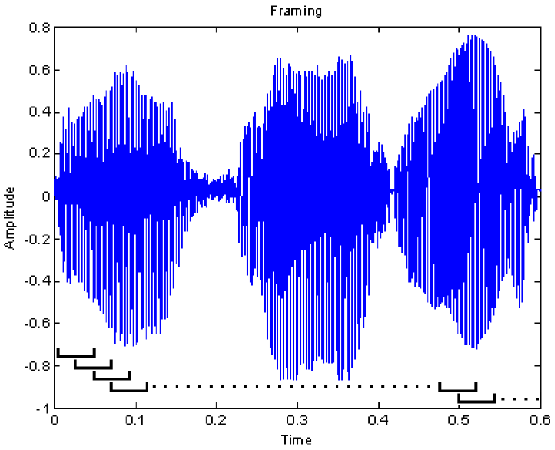


Figure 55 - Framing

To ensure that the signal is fairly stationary the length of the frame must be in the interval from 10 - 40 ms. Furthermore an overlap of 50 to 75 percent between frames must exists to ensure that windowing won't result in leaking of information.

The size of the frame is also an issue of resolution. With a large frame size one will get a good resolution in frequencies and with a small frame size a good resolution in time. In Figure 56 we have illustrated the resolution issue. The first graph shows a spectrogram where a window size of 10 ms have been used (160 samples from a 16KHz signal). The second graph shows a larger frame size of 40 ms (640 samples from a 16KHz signal).

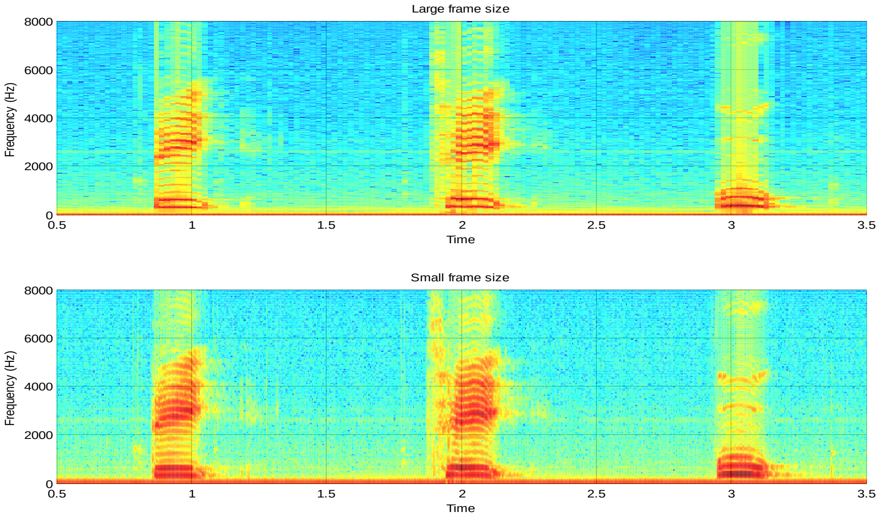


Figure 56 - The resolution issue

Figure 57 and Figure 58 shows a zoom of the pronouncement of the letter 'T' from Figure 56.

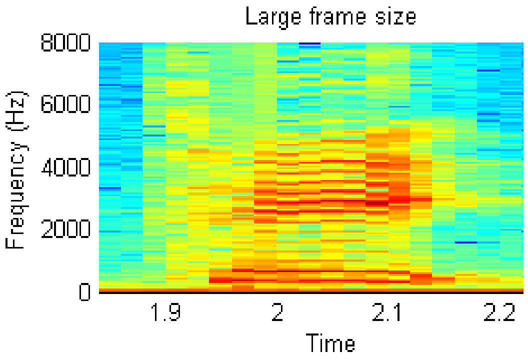


Figure 57 - Large frame size

As mentioned earlier using a large frame size results in good resolution in frequency which is illustrated in Figure 57.

The harmonics (vertical red lines) are easily seen. On the other hand the timeline resolution is poorly represented in relation to Figure 58.

A small frame size will result in a good resolution in time. By using a small frame size we know exactly which frequencies belongs to a given time.

On the other hand the frequencies seem blurred (see Figure 58) and it can be difficult to tell where the different harmonics are located.

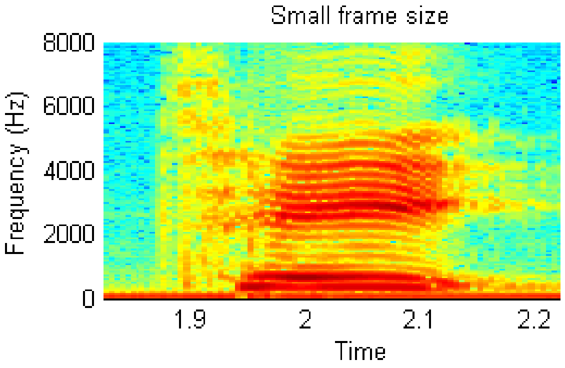


Figure 58 - Small frame size

4.2.2.4 Windowing

Window functions in signal processing are often used to damp neighbour frequencies or ensure that a signal seams fairly stationary over time. In signal processing the Hamming-, Hanning- and Rectangular window is among the most commonly know window functions. The difference between windows functions is based on the zeros at the slopes. A Rectangular window is exactly one inside a given interval and zero outside. The Hamming- and Hanning window only damps the samples at the slopes.

The reason for using a window on each frame in STFT is to get the signal fairly stationary at the end-points. If DFT is used on a non-periodic signal it is likely that the frequency response outputs undesirable results. Multiplying a window on every frame will have the effect of increasing the continuity at the endpoints of each frame.

In speaker identification a number of different window functions have been used, where the Hamming- and Hanning window are the most commonly used. The equation for these window functions are showed below in Lemma 27 for Hamming and Lemma 28 for Hanning.

$$w(n) = 0.53836 - 0.46164\cos\left(\frac{2\pi n}{N-1}\right)$$

Lemma 27

$$w(n) = 0.5\left(1-\cos\left(\frac{2\pi n}{N-1}\right)\right)$$

Lemma 28

As mentioned earlier a rectangular window won't have the effect of increasing the continuity at the endpoints wherefore it is not used in speaker identification systems. In Figure 59 the Hamming- and Hanning windows are illustrated. The windows seem identical except at the endpoints where the Hanning window reduces the amplitude more than the Hamming window.

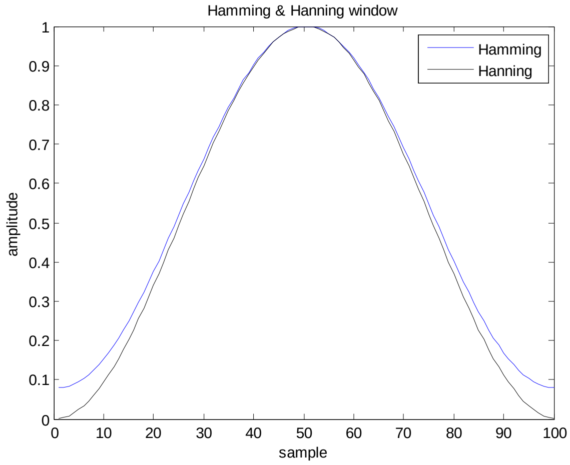


Figure 59 - Hamming and Hanning window

Figure 60 illustrates the effect of multiplying a window on a frame in the time domain.

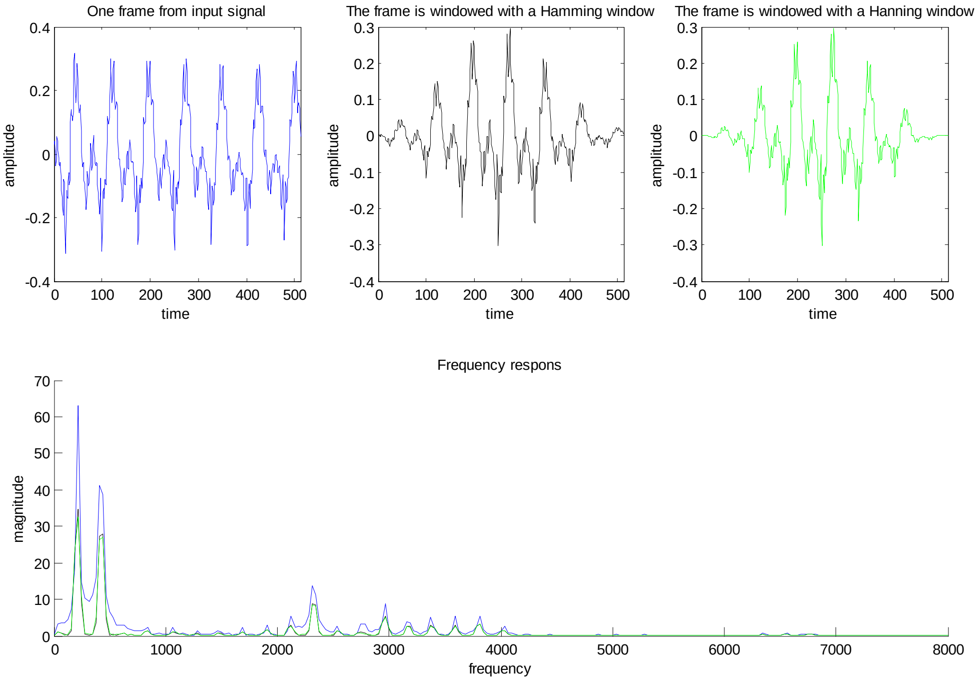


Figure 60 - Windows and frequency response

The frequency response in Figure 60 contains to peaks around 500 Hz. The different between the original frame and the two windowed frames is that the peaks in the windowed frames are much sharper whereby the neighbour frequencies are more damped – exactly what we are looking for. Again it can be difficult to see the difference in the frequency response between the Hamming- and Hanning window.

4.2.2.5 Discrete Fourier Transform

Discrete Fourier Transform (DFT) is a method that can be used to convert a speech signal $x[n]$ from the time domain into the frequency domain, where $x[n]$ represent a vector of amplitudes. The method is a discrete and finite transformation of the time domain (an approximation of continues time Fourier transform) which makes it ideal for processing data from e.g. digital signals. In Lemma 29 the DFT equation is illustrated.

$$X[k]=\sum_{n=0}^{N-1} x[n]e^{-j(2\pi/N)kn}, \text{ where } k=0,1,\ldots,N-1$$

Lemma 29

The Inverse Discrete Fourier Transform (IDFT) which is used e.g. in the derivation of MFCC is illustrated in Lemma 30.

$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k]e^{j(2\pi/N)kn}, \text{ where } 0 \le \text{n} \le N-1 \qquad \textbf{Lemma 30}$$

The effect of using a DFT and IDFT is illustrated in Figure 61. The sentence 'D T U' used in an earlier chapter is plotted in subplot 1. Subplot 2 shows the transform of the time domain signal into the frequency domain. It is seen from subplot 2 that the signal contains frequencies around 500Hz and 3KHz. Signal 3 shows the IDFT of the signal which converts it back into the time domain.

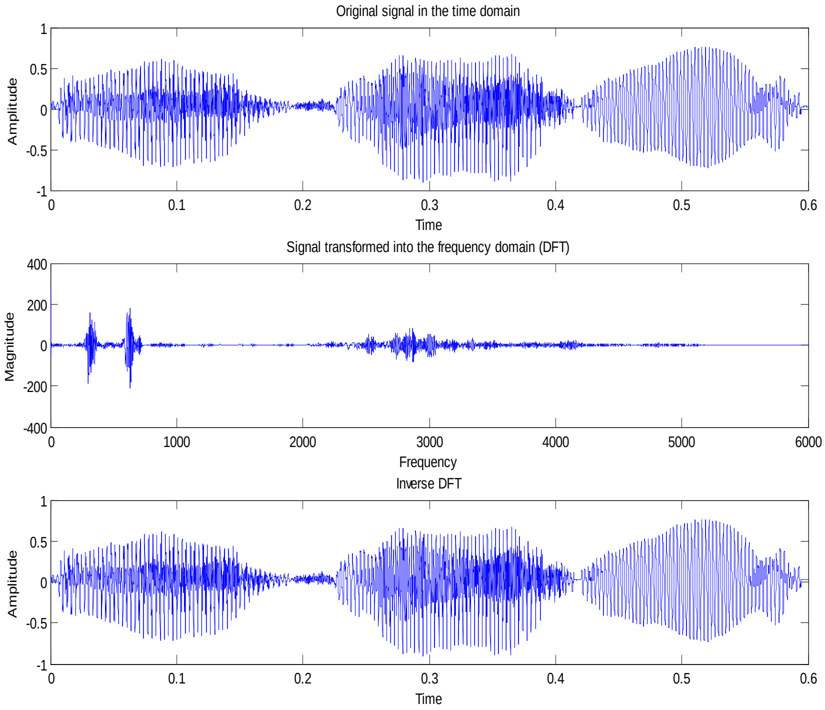


Figure 61 - Original signal, DFT and IDFT

In computer systems the DFT is normally calculated by a Fast Fourier Transform (FFT). The FFT is a faster method to transform the speech signals from the time domain into the frequency domain than DFT.

4.3 Cepstral analysis

4.3.1 *The basics*

In the previous chapter we mentioned how a speech signal can be analyzed in the frequency domain by a spectral analysis. Speech signals can also be transformed into cepstrum space. This method is often used when information about changes in different spectral bands is needed.

The ceptrum is defined to be the Fourier Transform of the logarithm of the spectrum. This means that the input signal $X = \left[x_1, x_2, \ldots, x_n\right]$ is transformed by a DFT into the frequency domain. The frequencies are then mapped into a logarithm space an a DFT is used on the result. This result is also known as the spectrum of a spectrum.

Transforming X into cepstrum space is illustrated in Figure 62.

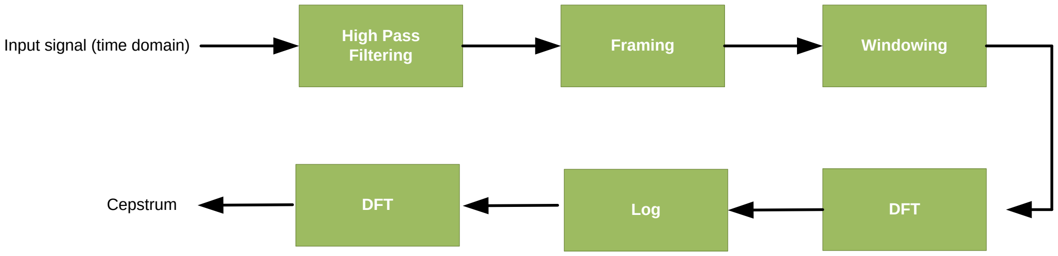


Figure 62 – Cepstrum representation

The mathematical representation of the transformation is showed in Lemma 31.

$$Cepstrum_{\text{signal}} = \text{DFT}(\log(\text{DFT}(\text{x[n]}))+\text{j}2\pi\text{m}) \qquad \text{Lemma 31}$$

Transforming X into cepstrum space gives the opportunity to compute Linear Prediction Cepstral Coefficients (LPCC) and Mel Frequency Cepstral Coefficients (MFCC). These are often used features in speaker identification and will be described in the next chapters.

4.3.2 *Linear Prediction Coding*

Linear Prediction Coding (LPC) is closely related to filter theory and based on estimating a linear function on a certain number of input samples. The weights used in estimating the linear function are called LPC coefficients.

LPC coefficients are calculated on smaller intervals in the time domain. The representation of the coefficients is showed in Lemma 32.

$$\hat{x}(n) = -\sum_{i=1}^{P} a_i x(n-i) \qquad \text{Lemma 32}$$

The error between the input sample $x(n)$ and the estimated value $\hat{x}(n)$ is showed in Lemma 33. The mean-square error is often minimized by an autocorrelation.

$$e(n) = x(n) - \hat{x}(n)$$

Lemma 33

The prediction coefficients can be found by solving the matrix in Lemma 34. The matrix can be solved relatively fast with the use of Levinson-Durbin Recursion.

$$\begin{bmatrix} r(0) & r(1) & \dots & r(p-1) \\ r(1) & r(0) & \dots & r(p-2) \\ \dots & \dots & \dots & \dots \\ r(p-1) & r(p-2) & \dots & r(0) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \dots \\ a_p \end{bmatrix} = \begin{bmatrix} r(1) \\ r(2) \\ \dots \\ r(p) \end{bmatrix}$$

Lemma 34

In this project LPC coefficients are not used as features. The reason for not using LPC coefficients is due to the fact the cepstral coefficients estimates the human voice more closely than LPC coefficients. Furthermore cepstral coefficients are more rubust in relation to noise.

The reason for including the LPC coefficients in this master thesis, is that Linear Prediction Cepstral Coefficients (LPCC) are derived by LPC coefficients.

4.3.3 Linear Prediction Cepstral coefficients

In section (4.3.2) LPC coefficients was described. As the LPC coefficients don't take the vocal tract into account the LPC can be transformed into cepstrum space and represented as Linear Prediction Cepstral Coefficients (LPCC).

LPCC can be derived from LPC using the recursion below:

$$c_0 = r(0), \text{ where r is derived from the LPC toeplitz autocorrelation matrix}$$

$$c_m = a_m + \sum_{k=1}^{m-1} (\frac{k}{n}) c_k \cdot a_{m-k}, \text{ where } 1<\text{m}<P$$

$$c_m = \sum_{k=m-p}^{m-1} (\frac{k}{n}) c_k \cdot a_{m-k}, \text{ where m}>P$$

a_m represent the m'th LPC coefficient and P is the number of LPCC's needed to be calculated. c_m is the m'th LPCC. As the equation shows it is possible to convert the LPC coefficients to an arbitrary number of LPCC's.

In speaker identification the number of LPCC's is often larger than 3 ($P > 3$). This makes it difficult to analyze the data visually wherefore PCA is used. In Figure 63 we have used PCA to transform LPCC's for two different persons into the two most important dimensions. The "red stars" belongs to a female and the "blue stars" belongs to a male. Both of the signals are sampled at 16 kHz with a frame size of 20ms and a step size of 10ms.

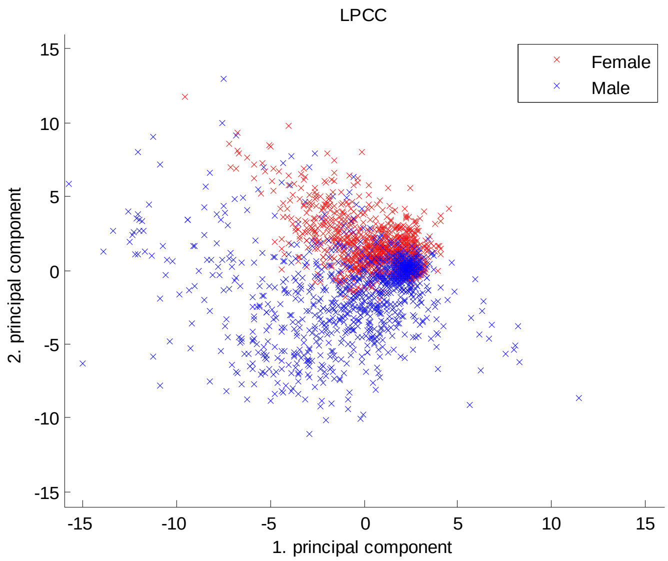


Figure 63 - 16 LPCC

Visually it can be difficult to distinguish between the two persons as the transformed coefficients overlap each other. It needs to be mentioned that only the LPCC have been used for this plot. In later chapters we see how delta-, delta-delta coefficients and the pitch can be used to optimize separation among speakers.

4.3.4 *Mel Frequency Cepstral Coefficients*

The most commonly used feature in speaker identification systems is the Mel Frequency Cepstral Coefficients (MFCC).

The MFCC models the human voice more closely than LPCC because the frequencies are scaled logarithmically on the "mel scale" which can be compared with the way human produces sounds. Furthermore these features have showed to be more robust against noise than any other method.

MFCC's are derived by taking the logarithm of the DFT of a windowed signal. One of the main differences between this method and LPCC is that MFCC uses triangular overlapping windows to map the logarithmic amplitudes into the Mel scale. The MFCC's are then the Discrete Cosine Transform (DCT) of the Mel Scale values.

Figure 64 illustrates the triangular overlapping windows (filterbank) and the transformation from frequency in Hz into the Mel scale.

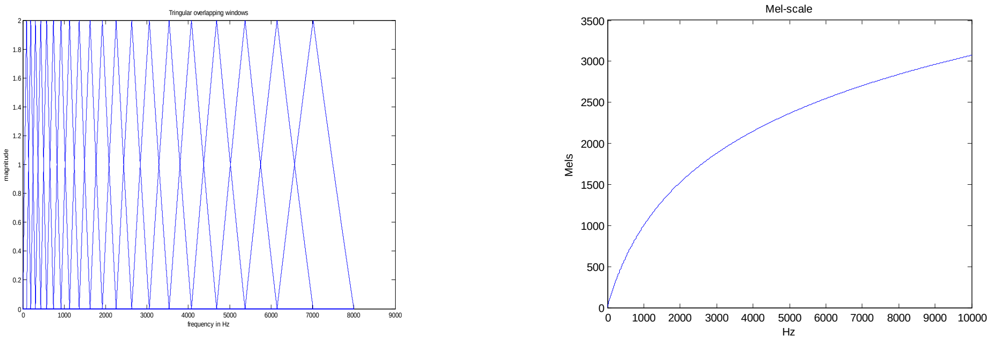


Figure 64 – Tringular overlapping windows and the Mel-scale

The mathematical conversion from Hz to the Mel-scale is showed in Lemma 35:

$$mel(f) = 1127.01048 \cdot \log(1 + f/700)$$

Lemma 35

The logarithm of the DFT multiplied with the filterbank is showed in Lemma 36:

$$X'(m) = \ln\left(\sum_{k=0}^{N-1} |X(k)| \cdot H(k,m)\right)$$

Lemma 36

The derived Mel scale amplitudes are converted into MFCCs by a DCT. In Lemma 37 $c(l)$ is the l'th MFCC.

$$c(l) = \sum_{m=1}^{M} X'(m)\cos(l\frac{\pi}{M}(m-\frac{1}{2})),\ \text{where } l = 1,2,....,\mathrm{M}$$

Lemma 37

In the previous chapter we showed how it visually can be difficult to separate two different speakers by their LPCC's. In Figure 65 we have computed MFCC's for the same two test persons and plotted them by PCA. Again the “red stars” belongs to the female and the “blue stars” belongs to the male.

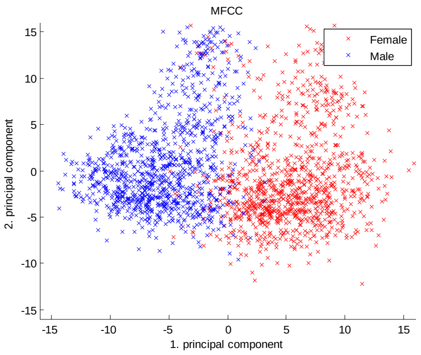


Figure 65 - 16 MFCC

From Figure 65 it is obvious that the MFCC coefficients separates the two test persons better than the LPCC coefficients. Still some transformed coefficients overlap each other but it almost seems like there exists two clusters that could be separated linearly.

4.3.5 Cepstral liftering

The cepstral coefficients (LPCC and MFCC) are usually weighted by a sine window. The most commonly used window is showed in Lemma 38.

$$l(n) = 1 + \frac{L}{2} \cdot \sin(\frac{\pi n}{L}), \text{ where } 1 \leq \text{n} \leq \text{p and L is a constant}$$ Lemma 38

The window is multiplied on each of the cepstral coefficients as showed in Lemma 39.

$$\hat{c}(n) = c(n) \cdot l(n)$$ Lemma 39

The idea of using a lifter on the cepstral coefficients is to smooth the spectral peaks. Liftering is generally used to improve the recognition rate of speaker identification systems.

The liftering window is plotted in Figure 66.

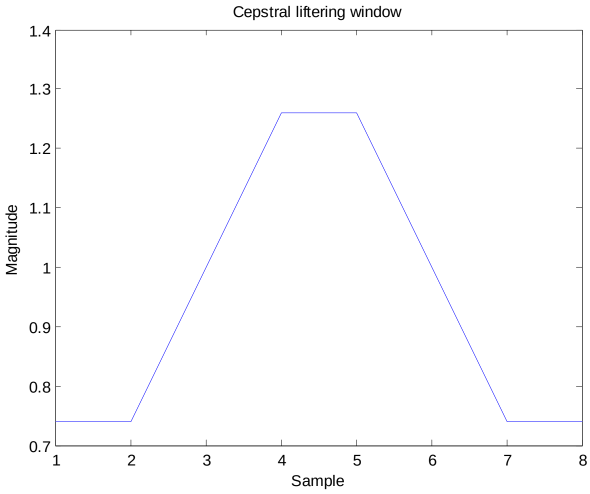


Figure 66 - Liftering window

4.3.6 Fundamental frequency / pitch period

The lowest or most dominant frequency in a periodic signal is the fundamental frequency f_0. The fundamental frequency also known as the pitch period, is one of the most commonly used features in speaker identification systems. The pitch period tells how many times the vocal folds vibrate per second (in Hz).

f_k is called the k^{th} harmonic of f_0. The greatest common divisor of f_k equals the fundamental frequency:

$$f_0 = \mathrm{gcd}\{f_k\}$$

Males often have lower pitched voices than females. The pitch period for a male typically ranges from 100 to 150Hz while the pitch period for a female voice typically ranges from 170 to 220Hz. This is the reason why a threshold value of 160Hz normally is used for gender identification.

No perfect algorithm for pitch detection exists, but a few methods can be used to make good estimates. Some methods derive the pitch period directly from the time domain e.g. by autocorrelation. Others use the frequency domain e.g. by a cepstrum analysis. We have used the last one for this project.

The general problem of finding the pitch period in a speech signal is the influence of noise and that the fundamental frequency can change over time.

In Figure 67 we have illustrated the pitch period for 23 speakers from the ELSDSR database. It is obvious that the pitch period can be used for gender identification as the male pitch period is below 160Hz and the female pitch period is above 160Hz.

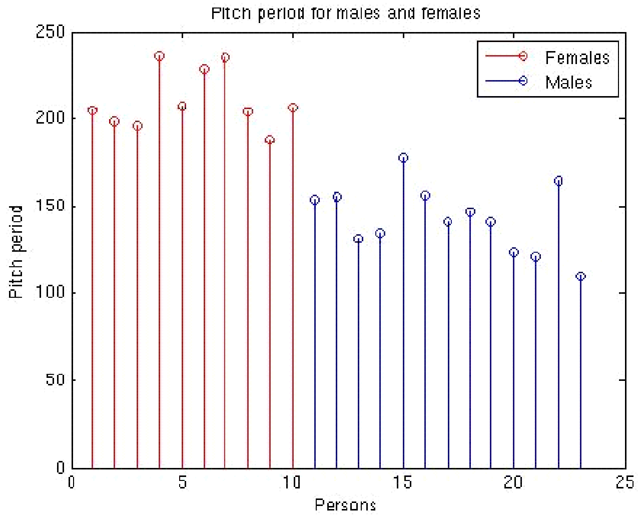


Figure 67 - Pitch period

4.4 Delta space coefficients

4.4.1 The basics

In the previous chapters we focused on how a speech signal can be represented by a number of coefficients, where the MFCC represented the human voice best. Ones the MFCC's are calculated, relatively low cost operations like delta MFCC's (dMFCC) and delta-delta MFCC's (ddMFCC) can be estimated. dMFCC is the first order derivative of MFCC and ddMFCC is the second derivative.

dMFCC and ddMFCC are often used features in speech modeling systems. The reason for using these derivatives of the MFCC's is due to the fact that the speed and acceleration of voice is very different among speakers. Therefore specific information is hidden between these coefficients.

We look at dMFCC and ddMFCC in both time and frequency space. The reason for this is explained in the following sections.

4.4.2 DMFCC & DDMFCC in time

The difference between the MFCC's in frame n and frame $n+1$ is called dMFCC in time. This can be computed as in Lemma 40 where t is the time index and i is the MFCC index.

$$\Delta c_{t,i}^{time} = \frac{1}{2}\left(c_{t+1,i} - c_{t,i}\right)$$

Lemma 40

The reason for looking at this is that we expected them to contain speaker dependent information on how a particular speakers voice is able to change between frequencies in time.

By using PCA the two most important dimensions of the dMFCC and ddMFCC has been plotted. The "blue stars" belongs to a male and the "red stars" belongs to a female.

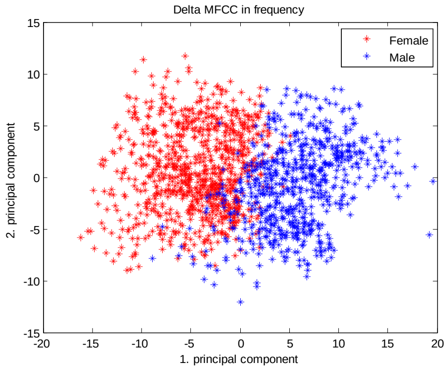


Figure 68 - dMFCC in time

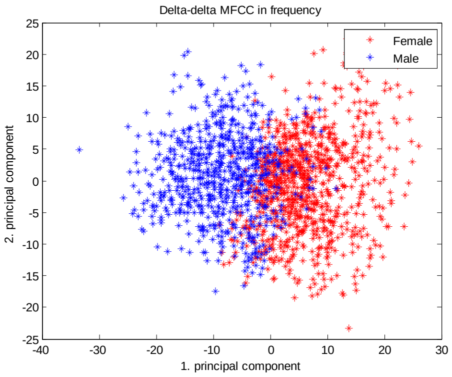


Figure 69 - ddMFCC in time

From Figure 68 and Figure 69 it is clear that the dMFCC and ddMFCC in time gives rather specific features. The transformed coefficients for the male and female are more separated than dMFCC and ddMFCC in frequency.

4.4.3 DMFCC & DDMFCC in frequency

The dMFCC and ddMFCC are describing the difference between the different order coefficients in one frame.

This information is not based on time and is therefore describing the static shape of the glottal valve. Due to differences in individual humans' biology we expected that each speaker has unique restrictions in the shape of his or hers glottal valve and thus each shape is unique for this individual. It is these shapes we model in frequency space.

Even though we had an assumption that these features could be useful, Figure 70 and Figure 71 indicate that no speaker dependant information is hidden in dMFCC and ddMFCC in frequency.

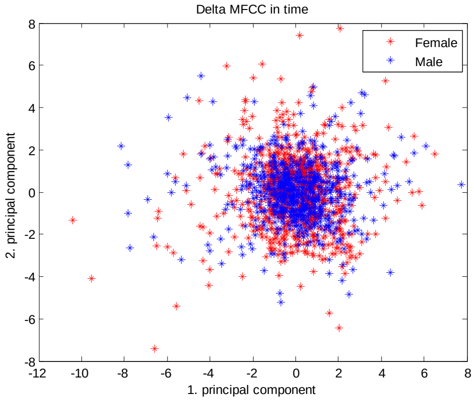


Figure 70 - dMFCC in frequency

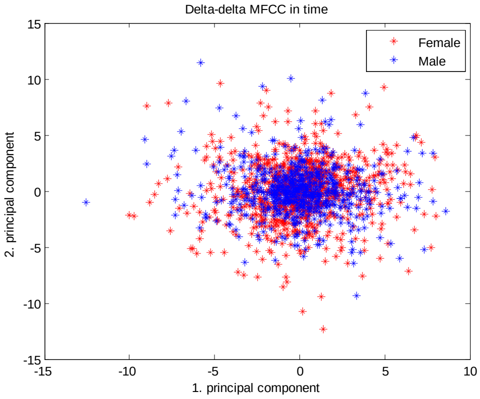


Figure 71 - ddMFCC in frequency

4.5 Results

In the feature extraction chapter we have focused on the representation of a speech signal in the time- and frequency domain.

Through a spectral analysis we were able to examine the speech signals in the frequency domain. A STFT was used to converts the speech signal in the time domain into the frequency domain. The STFT used a high-pass filter to imitate the human vocal tract, framing to ensure that frequencies can be detected in shorter time intervals, windowing to damp the neighbour frequencies and a FT to convert the speech signal into the frequency domain.

To represent a speech signal in a low dimensional space we detected different coefficients by a cepstral analysis. A LPCC and MFCC function was implemented in Matlab and tested visually using PCA. The two most important dimensions was plotted and visually it was possibly to distinguish between a male and female using MFCC's. This was not the fact by using LPCC's.

A lifter can be used on each frame of cepstral coefficients to smooth the spectral peaks. Liftering is generally used to improve the recognition rate of speaker identification systems.

Ones the MFCC's was derived, relatively low cost operations like calculating delta and delta-delta coefficients can be computed. Delta coefficients are the first derivative of MFCC and describe the speed of the speech. Delta-delta coefficients are the second derivative of MFCC and describe the acceleration of the speech.

Delta and delta-delta coefficients was analysed in frequency and time. Again PCA was used to plot the two most important dimensions. Visually the dMFCC and ddMFCC in time from two humans were easier to separate than the dMFCC and ddMFCC in frequency based on the same two humans.

The fundamental frequency or pitch period was also analysed. This feature is often used in gender identification. By plotting the pitch period we were able to separate females from males.

The features that will be used in the classification phase is MFCC, dMFCC and ddMFCC in time and the pitch period.

Report: Classification

Figure list:

1) Introduction

In the previous chapter different features were analysed to identify those that uniquely represent the human voice. MFCC, dMFCC and ddMFCC in time, cepstral liftering and the pitch period are among features and methods frequently used in speaker identification systems. These provide specific information about the traits of individual humans.

Pattern recognition is closely related to machine learning and used for classifying the above features (patterns). In this chapter we analyse and implement the two following classification systems:

- Gaussian Mixture Models
- Neural Networks

The reason for choosing these methods is caused by good results obtained previous e.g. by AT&T and MIT[1]. The reason for not including HMM is due to the fact that this model works best on text-dependent systems[2].

The above systems will be described mathematically, implemented and tested in Matlab. The purpose of this chapter is to provide information about how to set up parameters based on the number of speaker models in the speaker identification system.

Some important criteria's related to the classification systems are listed below:

- Speech data must be classified with high recognition rates
- It must be possible to classify speech data on smaller time intervals
- The system must classify speech data relatively fast
- Speaker models must be trained within a reasonable period of time

The classification system fulfilling the above criteria's will be implemented in the final application.

In the next section we describe how the classification systems are tested.

[1] Reference: http://www.ll.mit.edu/IST/pubs/000101_Reynolds.pdf

[2] Reference: http://www.ll.mit.edu/IST/pubs/000101_Reynolds.pdf

2) How to test

The classification systems are implemented and tested in Matlab. Different parameters can be tuned to achieve as good recognition rates as possible within a reasonable time used for training and classification.

The tests are based on 10 speakers (5 males and 5 females) from the ELSDSR database. The best set of parameters is found for systems containing 2, 4, 6, 8 and 10 speakers. This way we know exactly how to set up a system based on the number of speaker models.

From each speaker one sentence is used for training and another is used for testing the classification systems. This is an important point as using the same sentence for training and testing may result in overfitting. Both sentences contains approximately 6 seconds of “pure” speech.

Due to the fact that GMM and NN contains different parameters e.g. the number of mixture models in GMM and the number of neurons in NN, the following two subchapters describe how the individual classification systems are tested.

2.1 GMM test

The parameters that can be tuned for the GMM are:

- Filterbank size [28:4:36]
- Number of Gaussian Mixture Models [4:2:18]
- Number of MFCC’s [16:2:24]
- Frame size [10:10:40]
- Step size [25:25:75]
- Liftering [0 1]
- Window function [0 1]
- Ms of speech needed for classification [50:100:450]

The numbers in the brackets above ($[\,from:stepsize:to\,]$) are used in the test to find the best set of parameters based on the number of speaker models. A brief introduction to the parameters and a explanation of the chosen test values are listed below.

Filterbank size

The filterbank size is used for calculating the MFCC’s. The size of the filterbank can assume the values [28:4:36]. These are selected on the basis of already known information about filter sizes in speaker identification systems.

Number of Gaussian Mixture Models

A Gaussian is defined by its mean and variance. Speaker identification systems are often based on several Gaussian Mixture. The range of the values we test on is [4:2:18] as they are empirically promising.

Number of MFCC's

The MFCC's are used to describe the traits of individual humans. The numbers of MFCC's in speaker identification systems are usually in the low end of our selected values [16:2:24]. The reason is that dMFCC and ddMFCC are often concatenated with the MFCC. We use a wider range as these features are first tested in section (3.4).

Frame size

The values for the frame size [10:10:40] is in milliseconds and derived from the feature analysis. This range is normally used as the speech signal seams stationary in the above interval.

Step size

The step size [25:25:75] is measured in percent of the frame size. The lower step size the more MFCC's are calculated. The step size is described in the feature analysis.

Liftering

Liftering can assume the values [0 1], where 1 indicates that liftering is used. Liftering is often used on cepstral coefficients to smooth the spectral peaks (see feature analysis).

Window function

In the feature analysis the Hamming and Hanning window functions were described. This parameter can assume the value [0 1] where 0 is a Hamming and 1 a Hanning window.

Ms of speech needed per classification

The parameter ms of speech needed for classification [50:100:450] is used to tell how many milliseconds of speech is needed per classification to obtain a certain recognition rate in percent. The values this parameter can assume are relatively low due to the fact that we are implementing a real time system.

The reason for only testing on 10 out of 23 speakers, not including dMFCC, ddMFCC and the pitch period is due to the fact that this would result in enormous calculations. The classification systems and parameters are tested on the Gridterm DTU server that contains several kernels and still it is not possible within a reasonable time period to test all combinations of different parameters.

The parameters that are not included in the tests are as follows:

- dMFCC in frequency
- ddMFCC in frequency
- Fundamental frequency / Pitch period

The above parameters will be used in a later chapter to optimize the recognition rate ones we roughly know the best value of the eight parameters. We do not ignore these parameters as they mentioned later, provide important information about individual speakers.

2.2 NN test

The parameters that can be configured in the NN are:

- Filterbank size [28:4:36]
- Size of hidden layer[0:25:100] (0 = no hidden layer)
- Number of MFCC's [16:2:24]
- Sliding window size [50:100:450]
- Pseudo Gauss Newton steps [0:1:100]
- Temperature (pruning factor) [0:10]
- Weight decay [0.001:0]

The parameters Filterbank size, Number of MFCC and Sliding window size (Ms of speech needed for classification) is the same as for GMM.

Size of hidden layer

The size is equal to the number of neurons included in the hidden layer. The parameter can assume the values [0:25:100].

Pseudo Gauss Newton steps

This parameter indicates the number of Pseudo Gauss Newton steps used in the training phase to minimize the error rate before using gradient descent. The parameter can assume the values [0:1:100].

Temperature

This parameter controls how the pruning acts. It sets a delay for when to start pruning and a factor of how insignificant a connection must be before it gets pruned.

Weight decay

This parameter counters overfitting by preserving network state info from previous state.

3) Gaussian Mixture Models

Gaussian Mixture Models (GMM) is a commonly used classification system for speaker identification. The reason for using GMM for speaker recognition is due to the fact that speech features are usually assumed to be Gaussian distributed.

In text-independent systems where no prior knowledge what the speaker might say is known, GMM is one of the most successful classification systems[3].

GMM is a statistical model which estimates the mean and covariance for each individual speaker based on a feature vector. Mixture models are used where one Gaussian is not enough to describe a density. Below is a description of how the GMM works.

The equation for the D-dimensional Gaussian density function $b_i(x)$ is defined in Lemma 1.

$$b_i(x) = \frac{1}{2\pi^{D/2}\left|\Sigma_i\right|^{1/2}} \cdot \exp^{-(1/2(x-\mu_i)^T \Sigma_i^{-1}(x-\mu_i))}$$

Lemma 1

By using the density function $b_i(x)$ and the mixture weights w_i, the GMM can be defined as a weighted sum of M component densities as showed in Lemma 2.

$$P(x \mid \lambda) = \sum_{i=1}^{M} w_i b_i(x)$$

Lemma 2

Each speaker is represented by a mixture of means, variances and weights ($\lambda_i = \{w_i, \mu_i, \Sigma_i\}$) where $0 \le w_i \le 1$ and the sum of the mixture weights equals $\sum_{i=1}^{M} w_i = 1$.

To obtain the parameters for the individual speakers models the GMM needs to be trained. The Expectation Maximization (EM) algorithm is often used to train a GMM. The EM algorithm for updating the parameters is an iterative procedure with an Expectation and an Maximization step.

[3] Reference: http://www.ll.mit.edu/IST/pubs/000101_Reynolds.pdf

The Expectation step is showed in Lemma 3. This step computes the probability $P(i\,|\,x_n)$ that a datapoint x_n belongs to a given class/mixture i.

$$P(i\,|\,x_n) = \frac{P(i\,|\,\lambda)\cdot p(x_n\,|\,i_i)}{p(x_n\,|\,\lambda)}$$

Lemma 3

The above Lemma is also called a soft classification of x_n as it e.g. in the case of three classes/mixtures can belong 70 percent to class 1, 20 percent to class 2 and 10 percent to class 3.

The Maximization step is showed in Lemma 4.

$$\mu_i^{new} = \frac{\sum_n p^{old}(i\,|\,x_n)x_n}{\sum_n p^{old}(i\,|\,x_n)}$$

$$\Sigma_i^{new} = \frac{1}{M}\frac{\sum_n p^{old}(i\,|\,x_n)\left\|x_n - \mu_i^{new}\right\|^2}{\sum_n p^{old}(i\,|\,x_n)}$$

$$w_i^{new} = \frac{1}{N}\sum_n P^{old}(i\,|\,x_n)$$

Lemma 4

The mean, covariance and weights are updated iteratively. The EM algorithm typically converges after 10 – 40 iterations. Furthermore a threshold value is used as stopping criteria if the convergence difference is minimal as we don't want to overfit to the training set.

Ones the speaker models have been trained the objective is to find a speaker model in the speaker database $S = \{s_1, s_2 ..., s_n\}$ that result in the best match with a given input vector $X = \{x_1, x_2 ..., x_n\}$. The best match is found through a log likelihood calculation.

3.1 Training

This chapter focus on how speaker models are trained using GMM and the EM algorithm. We identify different factors that are important when training these models. The chapter is divided into the following subchapters:

- Initialization of the GMM
- Training of speaker models
- Seconds of speech needed for training
- Overfitting

3.1.1 Initialization of the GMM

Section (3) described how speaker models are represented by a mixture of means, variances and weights ($\lambda_i = \{w_i, \mu_i, \Sigma_i\}$). The initialization of those parameters is important to avoid achieving undesirable test result e.g. by overfitting.

Different methods can be used for initialization. In this chapter the main focus is on the methods described next:

- Randomly chosen parameters based on the training set
- Estimating the parameters based on the K-means algorithm

To describe the difference between the two methods these are implemented, tested and plotted in Matlab. Figure 1 illustrates the randomly chosen parameters and Figure 2 the K-means based parameters for the initialization.

Both figures plot a histogram showing the MFCC's (gray bars). Furthermore each individual mixture model is plotted in different colours (light blue, blue, red and green). The thick red line is the sum of Gaussian Mixture Models.

The following example is based on 4 Gaussian Mixture Models and the 5th MFCC from a female. The first method uses randomly chosen parameters for the mixture models.

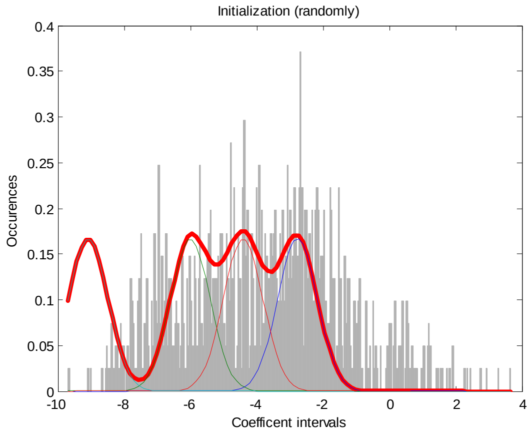


Figure 1 - Randomly chosen initialization

This method is easily implemented but converges relatively slow. Furthermore the method may result in an overfit.

Observing the Gaussians in Figure 1 it is obvious that the light blue mixture is located where few data points are represented (possible).

Furthermore the right side of the histogram is not represented by any of the mixtures.

In Figure 2 the second method is illustrated. This method is based on the K-means algorithm to initialize the mixture parameters.

The figure illustrates how each Gaussian represents different areas of the density.

This method for initialization is often used as it gives a better basis for the EM algorithm and it is more robust in avoiding overfitting.

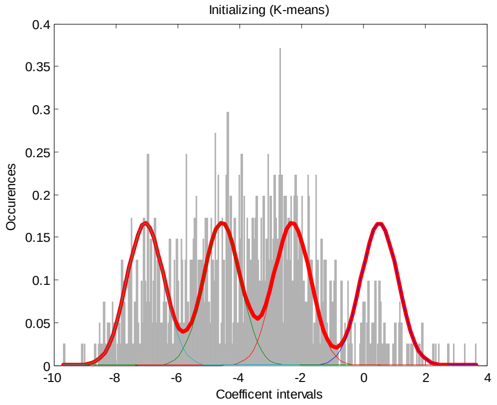


Figure 2 - K-means based initialization

Even though it seems that the K-means is an optimal solution for initialization of the GMM parameters, K-means is also able to ovefit datapoints. Alternatively the initialization can be based on well chosen datapoints e.g. by looking at the variance of the data set.

3.1.2 Train speaker models

In a speaker identification system each speaker needs to be represented by a speaker model. This speaker model is trained using the EM-algorithm. In this chapter we focus on how the speaker models are trained.

The example uses 4 mixture models, 8 MFCC and one speaker (female) for training. The training signal contains approximately 6 seconds of “pure” speech.

Figure 3 illustrates 8 different subplots – the 1th to 8th MFCC. The 4 mixture models are initialized based on the K-means algorithm.

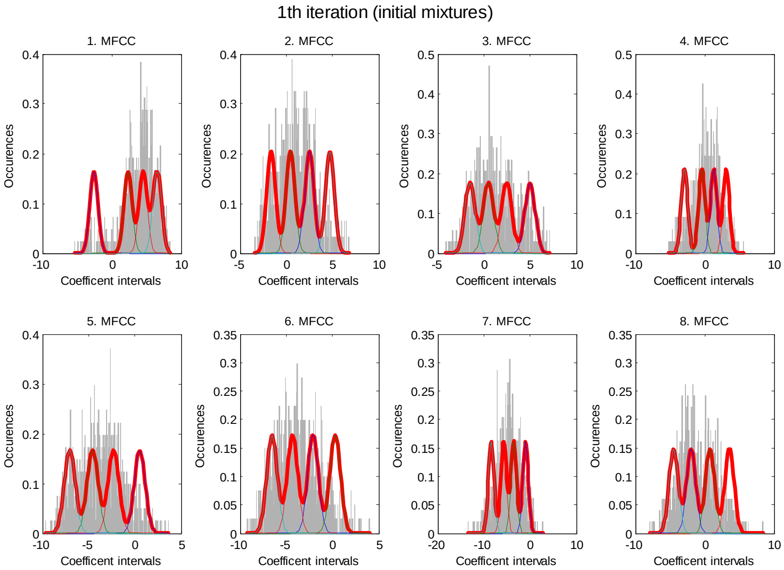


Figure 3 - Initialization of the GMM based on the K-means algorith

Figure 3 illustrates that the majority of the initialized mixture models gives a relatively good approximation of the individual densities (1th to 8th MFCC). The initial mixture models for the 1th MFCC is chosen as an example for training the Gaussians. Training the mixtures from the initial to the near "optimal" Gaussians are illustrated in Figure 4.

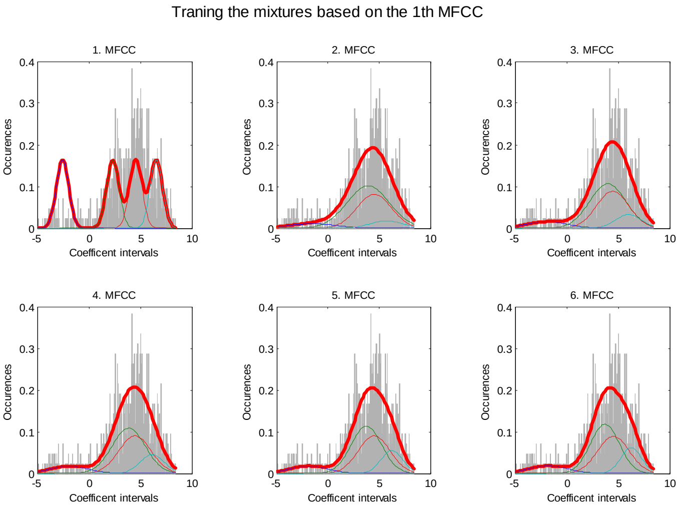


Figure 4 - Training the GMM based on the 1th MFCC

Figure 4 illustrates that just a few iterations is enough to adjust the mixture and to get better approximations of the densities.

A closer look at the 1th iteration (see Figure 4) shows that the light blue Gaussian have a mean located where just a few coefficient values exists. After a few iterations the weight, mean and variance of the light blue Gaussian have changes. The weight have decreased as few datapoints are represented in this area. Furthermore the variance have increased to describe the slope in the low end of the coefficient values.

In Figure 5 the four "optimal" mixture models for the eight MFCC are showed.

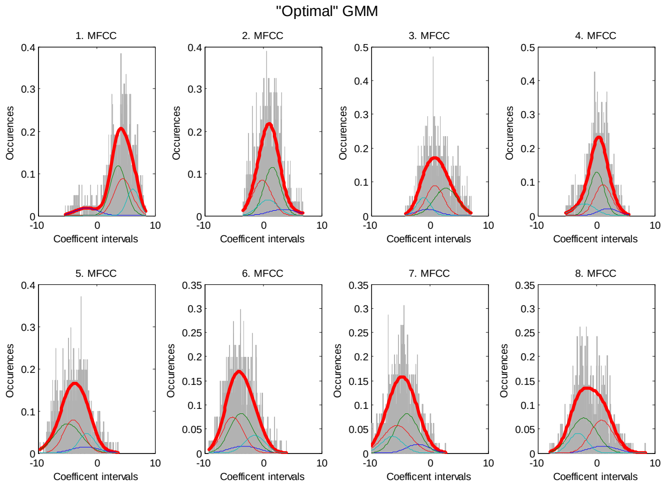


Figure 5 - "Optimal" mixtures for the 8 densities

3.1.3 *Seconds of speech needed for training*

This event only occurs at the time a new speaker needs to be included in the speaker database. As seen in Figure 6 it is important that we use enough speech samples for the training phase to achieve a relatively high recognition rate in percent. The x-axis represents seconds of "pure" speech (silence have been removed).

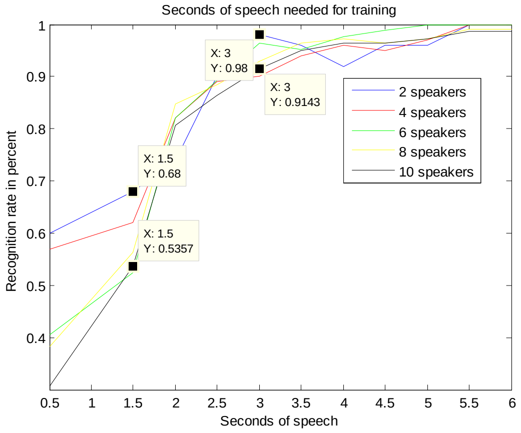


Figure 6 - Seconds of speech used for training

Figure 5 shows that using 1.5 seconds of speech in a system containing two speaker models will result in a recognition rate of 68 percent. Using the same amount of speech in a system containing 10 speaker models will result in a recognition rate of approximately 54 percent.

To achieve a recognition rate above 90 percent we need to use at least 3 seconds of speech. Using 3 seconds of speech in a system containing two speaker models will result in a recognition rate of 98 percent. In the case of a system containing 10 speaker models the result is a recognition rate of approximately 92 percent.

The above indicates that systems based on larger speaker databases require more data than systems based on smaller speaker databases to achieve the same recognition rates in percent. This is due to the fact that the models of the speakers need to be more specified.

To show that the above assumption is correct our point of reference in training two female speaker models is again the 5th MFCC. The two rows (blue = speaker 1 and red = speaker 2) in Figure 7 contain three subplots representing MFCC's of 0.5, 3 and 6 seconds of speech.

Focusing on the mean of the densities we can estimate the "true" mean based on all sentences from the two speakers contained in the ELSDSR database.

- "True" mean speaker 1 = -3.1729
- "True" mean speaker 2 = -1.5936

From the first graph in Figure 7 it is obvious that 0.5 seconds of speech is not enough data to represent the "true" mean for both speakers. Using 3 or 6 seconds of speech gives a better estimation of the "true" mean.

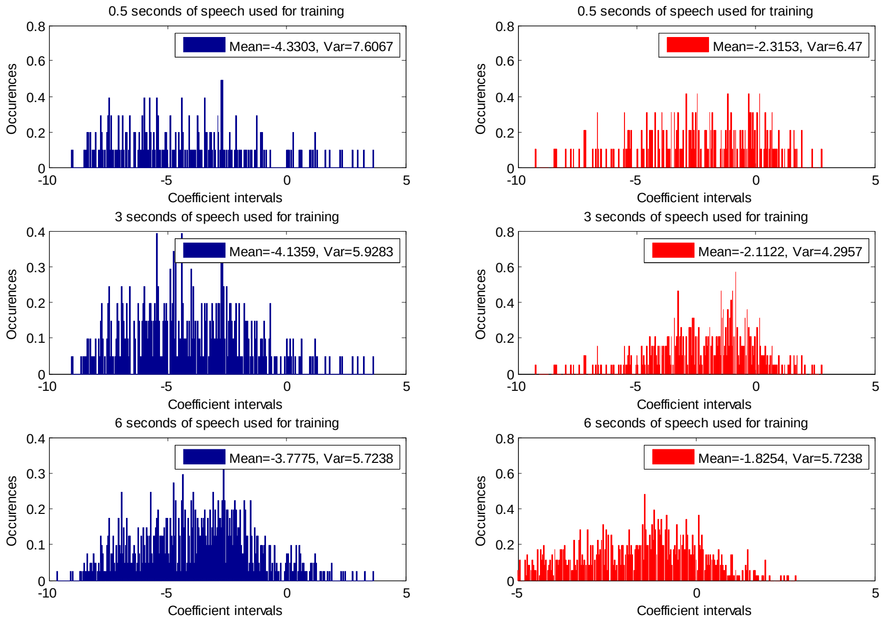


Figure 7 - Histogram showing the MFCC densities

This can also be illustrated by plotting the mean of the two densities. Figure 8Figure 7 illustrates that using more speech data for training results in better approximation of the "true" mean.

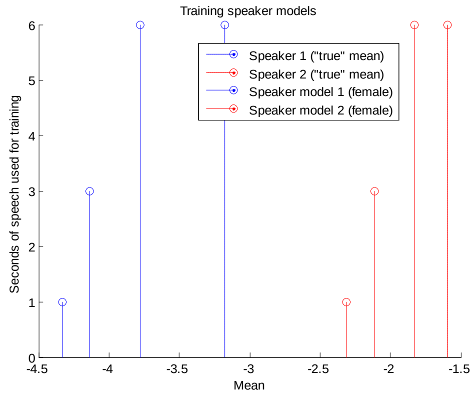


Figure 8 - – The mean of the speaker models (5th MFCC)

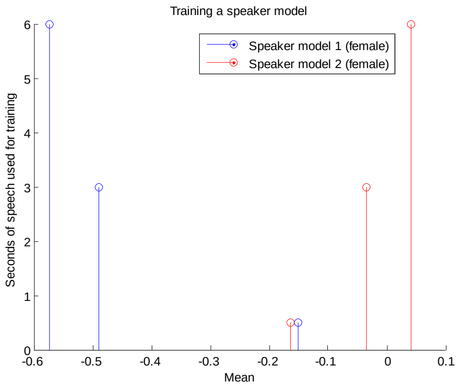


Figure 9 - The mean of the speaker models (8th MFCC)

In Figure 9 the same plot shows the 8th MFCC for the same two persons. This figure actually illustrates that using 0.5 seconds of speech result in two means not far from each other. Using 3 or 6 seconds of speech to calculate the MFCC's will give a better estimation of the "true" mean. This is seen in Figure 9 where the distance between the mean of the two speakers is perceptible larger than the mean value calculated from 0.5 seconds of speech.

The above figures illustrate the importance of using enough of speech (transformed to MFCC's) to train the individual speaker models. The more datapoints (MFCC's) the better estimates. This is an important factor in pattern recognition as using enough samples will result in better estimates of the mean and variance. It actually require more data to calculate an accurate variance than an accurate mean.

3.1.4 *Overfitting*

In pattern recognition overfitting is the concept of fitting a model to a training set. Overfitting results in an increase in the recognition rate on the training set and a decrease in recognition rate on the test set.

Different methods can be used to counter overfitting. In a GMM based speaker recognition system the following factors are important to avoid overfitting:

- Select sufficient amount of speech data for training
- Avoid large numbers of Gaussian Mixture Models
- Chose a suitable method for initialization
- Select a stopping criteria for the EM algorithm

Using to few samples will result in parameters (mean, variance) far from the parameters calculated on the "true" density. This leads to poor classification rates and may result in overfitting if the number of Gaussian Mixture Models is too large. One datapoint can in principle be represented by one mixture model - a serious overfit.

One method to counter overfitting is to initialize the GMM parameters based on the K-means algorithm. Using the K-means algorithm doesn't ensure overfitting but leads to better initialization of the GMM.

It is also important to select a suitable stopping criteria for the EM algorithm. If not using a stopping criteria the EM algorithm may converge against a low error rate on the training set which will result in an overfit – a increase in error rate on the test set. The stopping criteria for the EM algorithm can e.g. be a threshold value telling when the convergence difference is minimal.

In Figure 10 overfitting is illustrated in relation to selecting to many Gaussian Mixture Models in a systems containing few MFCC's. The figure shows that the error rate decreases the first couple of iterations whereupon the system overfit the training set. This results in an increasing error rate on the test set.

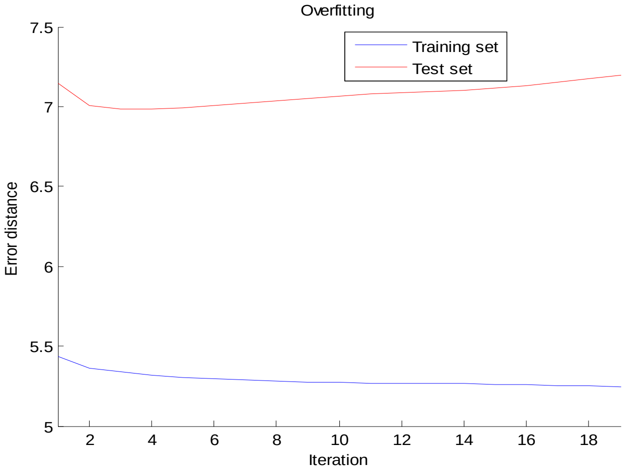


Figure 10 - Overfitting

3.2 Testing

This chapter focus on two main areas, the ms of speech needed per classification and the time used per classification. The ms of speech needed per classification describes how much data is needed to achieve a certain recognition rate in percent. The time used per classification indicates the time used for one classification in ms.

3.2.1 Ms of speech needed per classification

Figure 11 illustrates how much data is needed to obtain the recognition rates described in section(3.3). The figure indicates that 250 ms of data is needed to obtain a recognition rate of a 100 percent using 2 or 4 speaker models.

Systems containing 6 or 8 speaker models need 450 ms of speech to obtain the same recognition rate.

Systems based on 10 or more speaker models are not able to classify data with a 100 percent using less than 500ms of speech.

Other method can be used to optimize the recognition rate (see section 3.4).

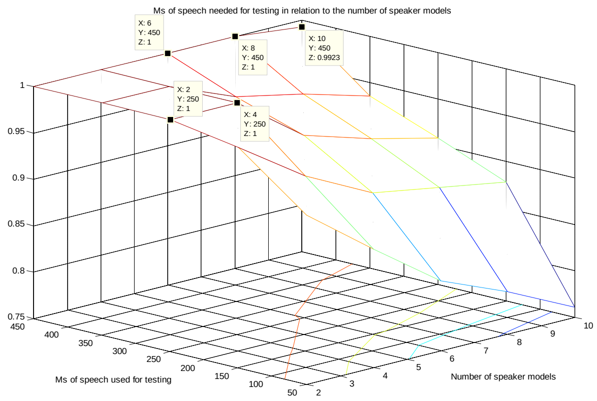


Figure 11 - Ms of speech needed per classification

The surface between the x-axis and y-axis illustrates how much data is needed per classification as the number of speaker models increases (seen by the red line).

Figure 11 and Figure 12 are identical. The difference is the "datatips" in this case showing the ms of speech needed to obtain a recognition rate above

90 percent.

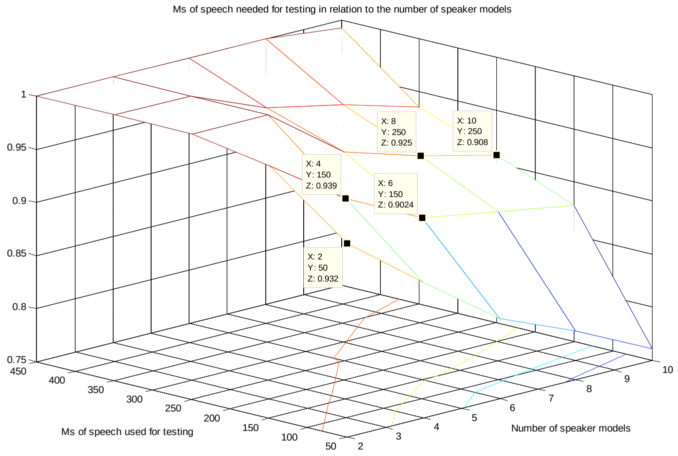


Figure 12 - Ms of speech needed per classification

The two figures support the theory that systems with larger speaker databases need more speech data than systems with smaller speaker databases for classification. This is due to the fact that the individual speaker models need to be more specified. This can be achieved by calculating more MFCC's.

3.2.2 Time used per classification

Another important factor in speaker identification systems is the time used per classification. The number of ms needed per classification is closely related to the time used per classification. The reason for focusing on the time used per classification is due to the fact that we are implementing a real time system and expect to classify speech data relatively fast.

In the previous chapter it was concluded that larger speaker databases needs more speech for classification than speaker identification systems with fewer speakers. In Figure XX the time used per classification is illustrated. The figure illustrates than an increase in the number of speaker models results in an increase in the time used per classification.

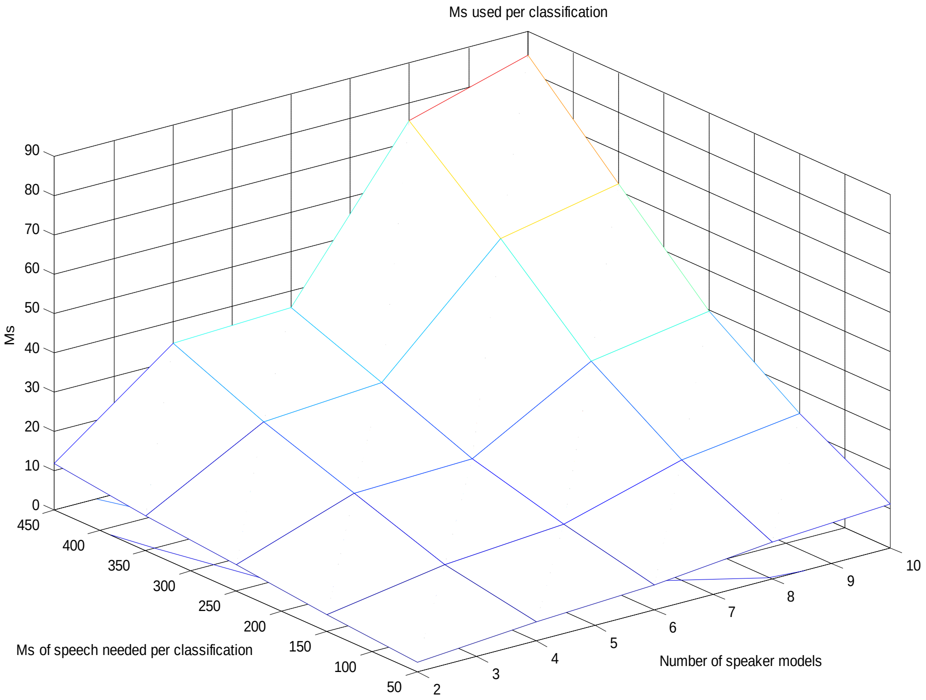


Figure 13 - Time used per classification

Figure 13 illustrates that an increase in the ms needed for classification will result in an increase in time used per classification. The total time used per classification is illustrated in Figure 14.

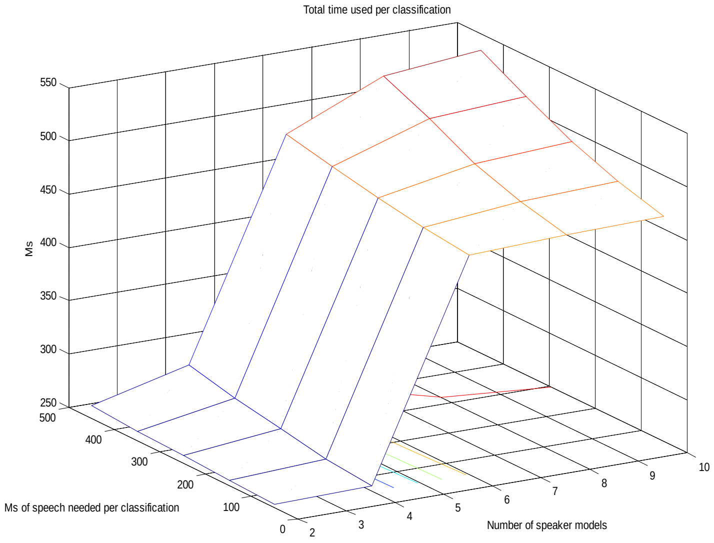


Figure 14 - Total time used per classification

From the above figures it is quite obvious that the ms of speech needed per classification is the most important factor in a real time classification system using GMM. The time used per classification aren't increasing exponential and actually only a minor factor in the total classification time.

3.2.3 Number of Gaussian Mixture models

The number of Gaussian Mixture Models is closely related to the number of MFCC's and speaker models in the speaker identification system.

Systems containing few speaker models needs few MFCC's and therefore few GMM to represent the speaker models. Systems containing larger speaker databases needs more MFCC's to describe the traits of individual speakers and thereby more GMM to represent the coefficients.

The evolution of GMM in relation to number of speaker models in a system is illustrated in Figure 15.

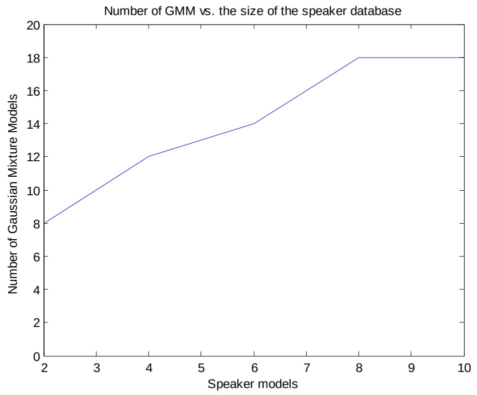


Figure 15 - Number of Gaussians plotted against the number of speaker models

Number of MFCC

The number of MFCC's have an enormous influence on the recognition rate. From Figure 16 it is obvious that recognition rates above 90 percent, can be achieved using 6 – 10 MFCC's in a system containing 2 – 6 speaker models.

Larger speaker databases (8 – 10 speaker models) needs 14 MFCC's to achieve a recognition rate above 90 percent.

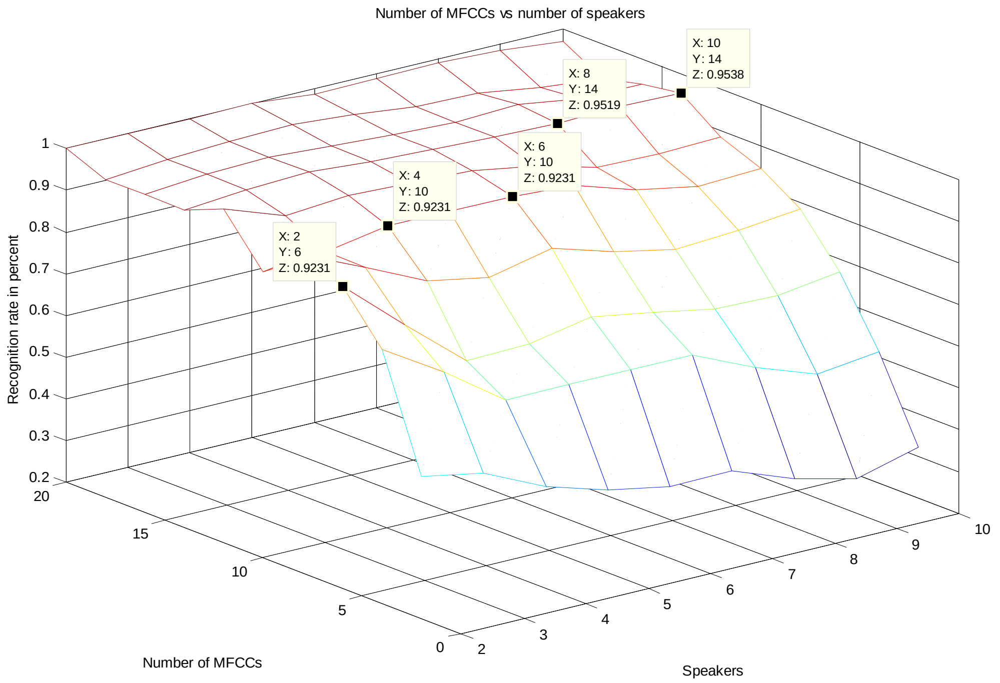


Figure 16 - Number of MFCC's plotted against the number of speaker models

3.3 Overall performance

In this chapter we describe the results achieved by implementing a speaker identification prototype based on GMM.

One sentence of speech is used to train the speaker models. The test signals are based on another sentence of speech from each speaker. To imitate a conversation the signals have been concatenating ($X = [signal_1, ..., signal_n]$) and used for the test.

Figure 17 illustrates four different classifications of speech in a system containing 10 speaker models. The figure shows that the input sentence has the best match with speaker model 1 (correct classification) due to the lowest error rate.

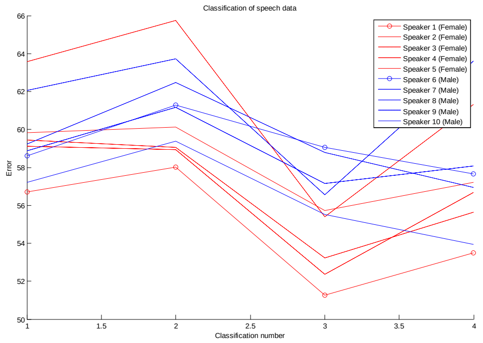


Figure 17 – Classification of speech

The following five figures illustrates the recognition rates achieved using a system based on respectively 2, 4, 6, 8 and 10 speakers. The systems have different parameters settings based on the number of models.

Figure 18 and Figure 19 illustrates the classification of speech data in a system containing respectively 2 and 4 speaker models. The system classifies the speech data with a recognition rate of a 100 percent.

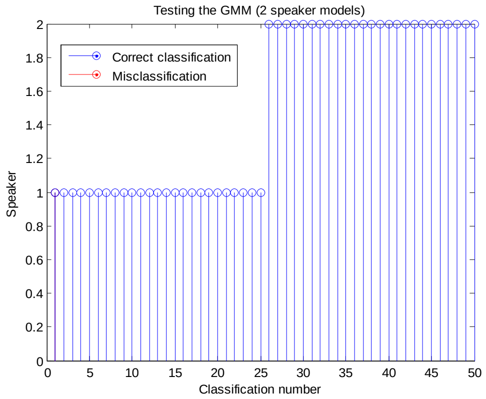


Figure 18 - 2 speaker models

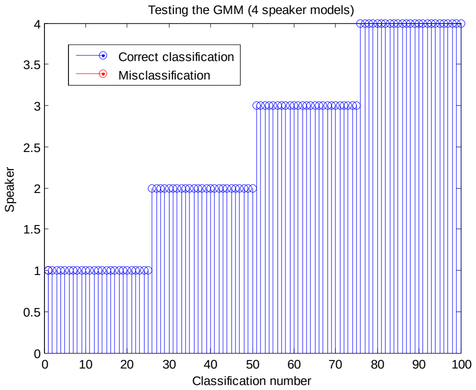


Figure 19 - 4 speaker models

Figure 20Figure 18 and Figure 21Figure 19 illustrates the classification of speech using a system based on 6 and 8 speaker models. Again the GMM classify with a recognition rate of a 100 percent. It might be noticed that the number of classifications changes in the different plots. This is due to the fact that the number of speaker models changes and that different systems are based on different parameters e.g. the frame- and step size.

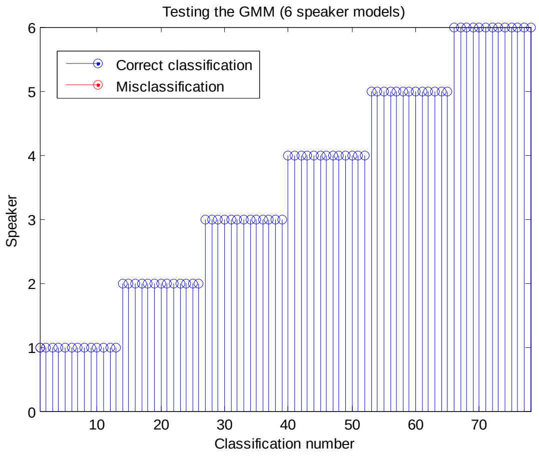


Figure 20 - 6 speaker models

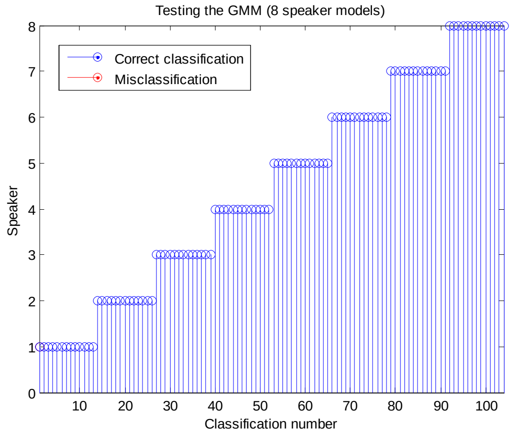


Figure 21 - 8 speaker models

In Figure 22 one of the 130 classifications is misclassified. This test is based on 10 speaker models which results in a recognition rate of $(1-\frac{1}{130})\cdot 100 = 99.23$ percent.

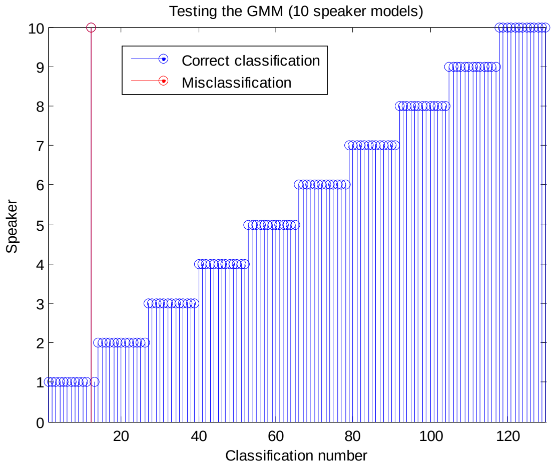


Figure 22 - 10 speaker models

Testing the GMM with different number of speaker models resulted in high recognition rates. The tests were based on different configuration of the eight parameters described in section (2.1).

The eight parameters can be combined in $3 \cdot 8 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 2 \cdot 5 = 28800$ different ways. Testing the GMM with 9 nested for-loops (the speaker models included) resulted in the following number of combinations that outputted a 100 percent recognition rate:

- 2 speakers – 332 combinations
- 4 speakers – 77 combinations
- 6 speakers – 26 combinations
- 8 Speakers – 2 combinations
- 10 speakers – 0 combinations

From the above it is obvious that systems containing few speaker models are less affected by the parameter configuration. Larger speaker databases are more affected and by using 10 speakers we cannot get a recognition rate of a 100 percent without including extra parameters like dMFCC (see section 3.4).

It needs to be mentioned that the above tests are based on perfect conditions. The speech data is recorded with almost no ambient noise. Furthermore the above systems use as much speech for training as needed. This has an enormous effect on the recognition rate.

3.4 Improvements of the GMM

Several improvements can be made to optimize the recognition rate in a GMM based speaker identification system. As mentioned earlier, once the MFCC's are derived low cost calculations like delta- and delta-delta coefficients can be computed. The pitch period can also be calculated and used for optimization.

DMFCC and DDMFCC in time

Information about the speed and acceleration of speech are speaker dependent as described in feature analysis. In Figure 23 we have illustrated the influence of including dMFCC and ddMFCC in time in the speaker identification system.

Concatenating the MFCC and dMFCC actually showed to give the best results. In Figure 23, two "datatip" illustrate the effect of using MFCC combined with dMFCC. Using MFCC, dMFCC and 250 ms of speech the system is able to classify with a recognition rate of 93.6 percent. Only using MFCC the system needs 350ms of speech to achieve the same results (93.5 percent).

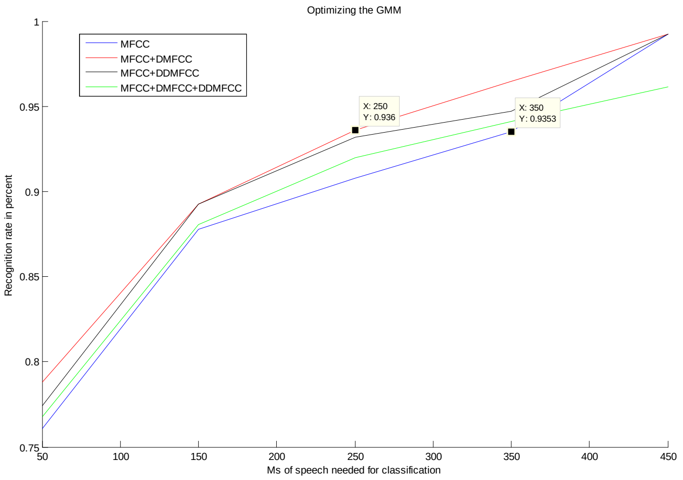


Figure 23 – Features

Other combinations like using ddMFCC together with MFCC and dMFCC won't result in better recognition rates. The effect of using ddMFCC only increase the time used for classification but not the recognition rate.

It needs to be mentioned that concatenating MFCC and dMFCC results in an increase in the time used per classification. The time used per classification is though a minor factor in the total classification time.

Pitch period

The pitch provides important information that can be used to separate genders. Using the pitch in the GMM didn't actually increase the recognition rate.

The pitch will not be neglected but used in the final implementation as a factor in the classification phase. If the log likelihood of e.g. two persons is nearly the same the pitch can be used as a final parameter in the classification.

4) Neural networks

4.1 Selecting network type(s)

In this project, we have initially considered 3 different network types. The purpose is to choose one or more network types which are feasible for speaker identification in this project.

1. Hopfield network which is a pattern regeneration network.
2. Self organizing feature map which is an unsupervised network.
3. Multilayer perceptron network (MLP) which is a back propagation network.

Hopfield network (discarded)

The Hopfield is infeasible due to its binary nature. Although we have briefly considered converting the floating point values of the speaker features into a binary format this was discarded due to the excessive network size this would lead to.

Kohonen selforganizing feature map (discarded)

The Kohonen unsupervised network needs to be used width clustering if it should be used to identify a particular speaker and not just an unlabelled speaker. This makes this SOM uninteresting for this project.

MLP network (selected)

The multiplayer perceptron supervised network is highly configurable and can be used to estimate the system function of even very complex data. This network is well suited for modeling the extracted speaker features.

This network is analyzed in this report.

4.2 Overviews of neural networks as used in this project

In this project we have chosen to use the MLP network in 2 different ways namely:

- Single MLP network
- MLP network cluster

4.2.1 Single MLP network (used method)

A Single MLP network (Figure 24) is the most common way of using the neural network. It is presented a lot of features each belonging to a known class or speaker identity in this project.

There are one output neuron per speaker identity.

The benefit is that it is easy to find the class with the highest likelihood.

The downfall is that we need to retrain the network each on all data if we want a new combination of classes. This is a problem in this project as we want to be able to add classes dynamically. The consequence is an increased amount of computation for retraining the network each time a new combination of classes is needed.

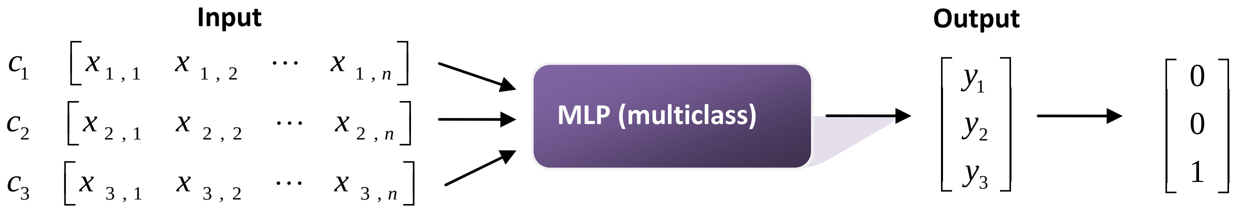

Input
c_1 $[x_{1,1} \; x_{1,2} \; \cdots \; x_{1,n}]$
c_2 $[x_{2,1} \; x_{2,2} \; \cdots \; x_{2,n}]$
c_3 $[x_{3,1} \; x_{3,2} \; \cdots \; x_{3,n}]$
MLP (multiclass)
Output
y_1 y_2 y_3
0 0 1

Figure 24 - Single MLP network

4.2.2 MLP network cluster (tried but not used)

For this method we didn't know what results to expect. The idea was to train one network per speaker as in Figure 25. This means that each network was conceptually a regression network. But used for Boolean single class decision.

It was expected that if we grossly overfitted the network to the particular class and pruned it aggressively then this network *should* be so specialized that it would only yield a result close to the target when we classified with data from the same class used for training. Other classes should yield poor results. Thus we should be able to test an unlabelled feature set against all networks and identify the network which most likely belonged to the true class.

The reason for trying this approach is that we wanted to avoid having to retrain a network each time a new combination of classes is needed. But remember that we hadn't tried this before.

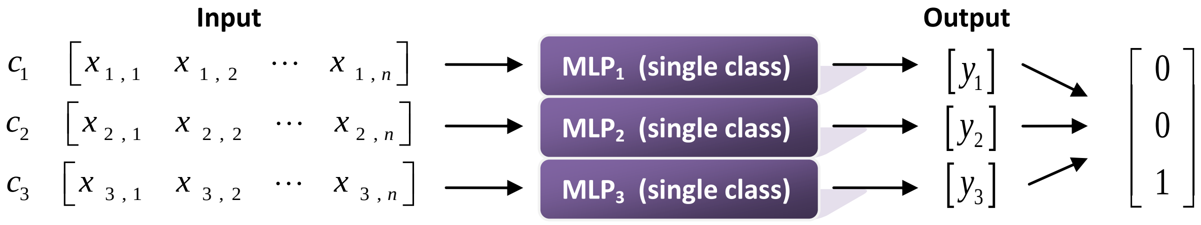

Input
Output
c_1 $[x_{1,1} \; x_{1,2} \; \cdots \; x_{1,n}]$
c_2 $[x_{2,1} \; x_{2,2} \; \cdots \; x_{2,n}]$
c_3 $[x_{3,1} \; x_{3,2} \; \cdots \; x_{3,n}]$
MLP₁ (single class)
MLP₂ (single class)
MLP₃ (single class)
$[y_1]$
$[y_2]$
$[y_3]$
0
0
1

Figure 25 - MLP network cluster

We tested it for 2 persons and 6 persons respectively but it yielded such poor results that we discarded this method.

All networks were trained in a variety of combinations. The best result achieved was with no weight decay and only PGN as training function. We used 5*20 MFCC coefficients for input (approximately 60 milliseconds) per classification for both 2 and 6 classes. The networks was configured as [100 in ; 200/0 hidden ; 2/6 out]. The best achieved results are shown in Figure 26.

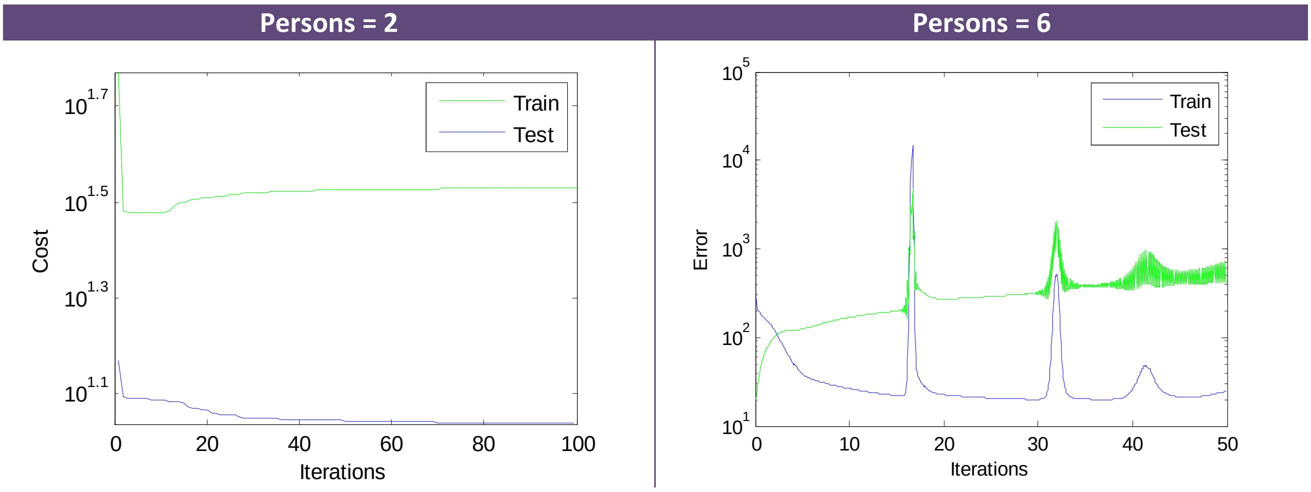


Figure 26 - Network cluster best results

4.3 Training the MLP network

One of the major benefits of this particular network type is the scalability. It can have any number of hidden layers which each can assume any size. This property makes it suitable for solving both larger tasks and model more complex surfaces (N-dimensional functions).

Another important benefit is that the network can be used both for classification with C-1 classes of output or regression (e.g. time series analysis etc.) with only one output.

The network is biased in each layer which means that it can handle input pattern scale issues internally.

The layout of a MLP network with 1 hidden layer is displayed in Figure 27.

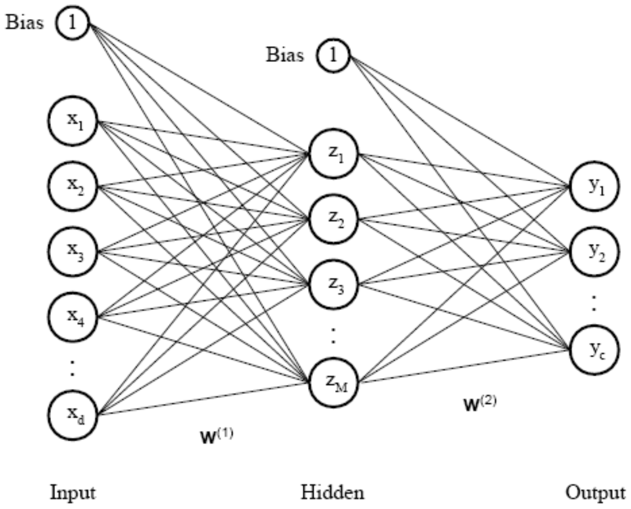


Figure 27 - MLP network layout

4.3.1 Overview of the MLP training procedure

As seen in Figure 28 we use 2 different methods for training the network.

The general idea is that Pseudo Gauss Newton (PGN) is faster at approaching the global minimum cost. It is also better at avoiding getting stuck in local minima. For these reasons we will try to use a number of PGN iterations to begin with.

Gradient descent however is better than PGN at minimizing the cost when approaching the found minimum. This is because PGN moves in an elliptic pattern whereas gradient descent steps directly at the gradient direction towards some local (or global if we are lucky) minimum.

Another point of interest is the pruning which takes place in the components "Kill non-active delta weights". Pruning is a way of avoiding that the network fits to features which has very little influence on the classification.

Finally we use line search which is a divide and conquer way of dynamically determining a good stepsize which significantly increases training time. This is done in the component "find optimal stepsize".

These and other components are elaborated later.

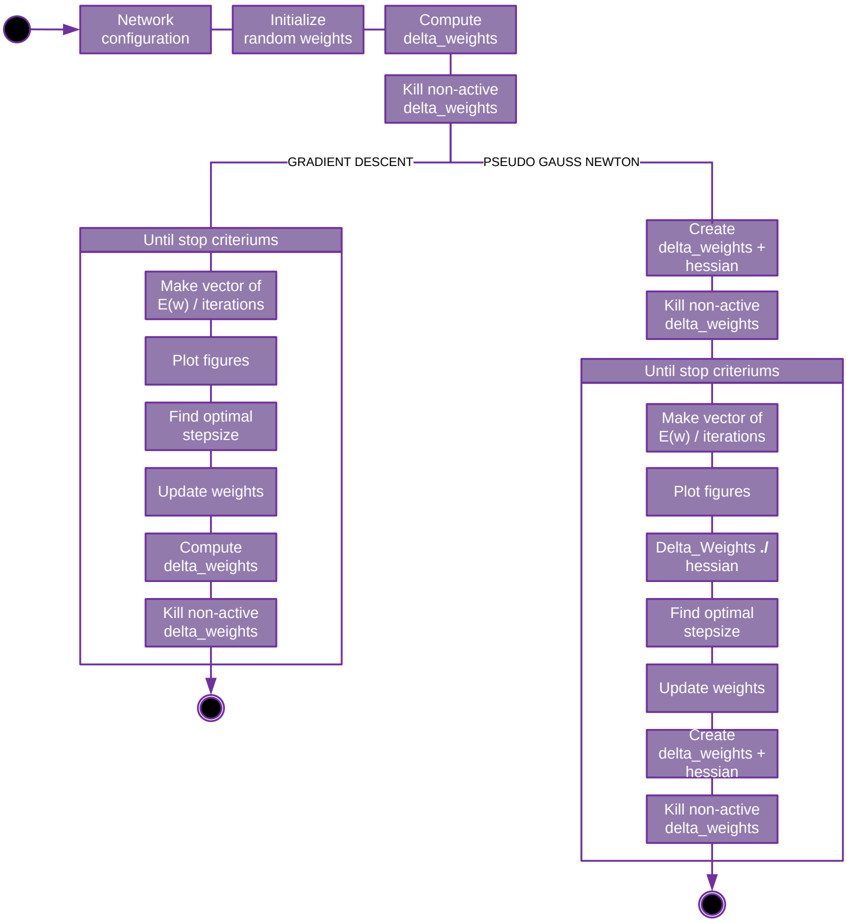

Network configuration
Initialize random weights
Compute delta_weights
Kill non-active delta_weights
GRADIENT DESCENT
PSEUDO GAUSS NEWTON
Until stop criteriums
Make vector of E(w) / iterations
Plot figures
Find optimal stepsize
Update weights
Compute delta_weights
Kill non-active delta_weights
Create delta_weights + hessian
Kill non-active delta_weights
Until stop criteriums
Make vector of E(w) / iterations
Plot figures
Delta_Weights ./ hessian
Find optimal stepsize
Update weights
Create delta_weights + hessian
Kill non-active delta_weights

Figure 28 - Network component overview

4.3.2 *Classification function*

The classification or feed forward of the MLP network is performed by passing the input pattern from the previous layer through each layer in steps.

For a network with one hidden layer as in Figure 27 we can use Lemma 5 to compute the activation of the hidden layer where $g(\cdot)$ is the activation function. This can be repeated if more hidden layers are used.

Note that the first input is always 1 which is input to the bias.

$$z_j = g\left(\sum_{i=0}^{d} w_{ji}^{(1)} x_i\right) \quad , \quad x_0 \equiv 1$$

Lemma 5

The output is computed without activation as in Lemma 6 which therefore is a linear scale function of the input from the last hidden layer.

$$y_k = \sum_{i=0}^{d} w_{kj}^{(2)} z_i \quad , \quad z_0 \equiv 1$$

Lemma 6

4.3.3 *Error function and softmax of output*

The error function (or cost function as it is also called) should compute the distance between the target values and the actual output of the network.

Speaker identification is a multi-class problem and we use the Bernoulli distribution of outputs which means that we use a position coded binary target vector. For a 4 class problem this could be: $t_n = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}$ for a pattern of class 2.

This is particularly important as we shall see shortly.

4.3.3.1 "Sum of squares" error function

From looking through various sources, it appears that the "sum of squares" error function as in Lemma 7 is the most widely used error function, also for multi-class classification problems.

$$E(w)_{square} = \frac{1}{2} \sum_{n}^{N} \sum_{k}^{c} \{ y(x^n ; w)_k - t_k^n \}^2$$

Lemma 7

This function however appears to have a problem as it is connected to the normal noise hypothesis. But as we use binary multi-class target vectors then a Gaussian error is *not* what we want. It appears to be far better suited for regression networks.

4.3.3.2 Softmax the output

As the target vector is position coded the targets are *dependent* of each other it is no longer prudent that the output is based on the logistic sigmoid activation function (Lemma 26). Instead we use a generalization of this activation function for a multinomial case (where output *is* dependant).

Softmax is such a function and is computed as in Lemma 8 where ϕ is the network output before Softmax.

$$y_k = \frac{\exp(\phi_k)}{\sum_j^C \exp(\phi_j)}$$ Lemma 8

In Figure 29 we can see how softmax and log likelihood error function penalizes wrong descisions more than using the square error function.

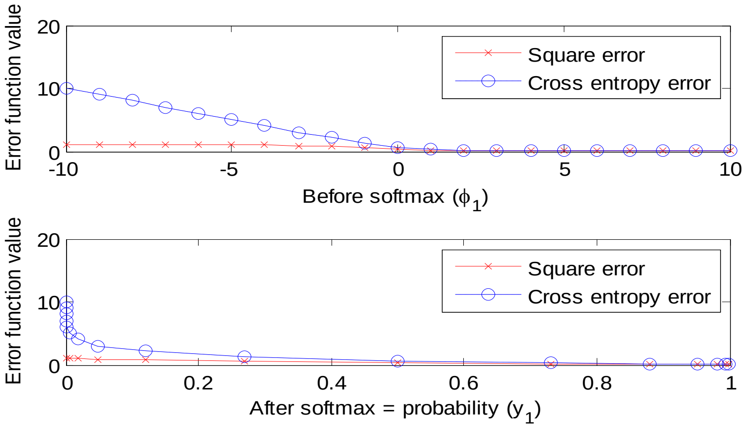


Figure 29 - Error functions and softmax relationship

4.3.3.3 Cross-entropy error function

It now appears more suitable to use a Bayesian likelihood function to model the error than the square error function. As we know the target output t and the actual output y this can be formulated as in Lemma 9.

$$p(t^n \mid x^n) = \prod_k^C \left[y_k^n \right]^{t_k^n}$$ Lemma 9

As it is not the likelihood but error we are interested in, we use the negative log likelihood as in Lemma 10.

$$E(w)_{entropy} = -\sum_n^N \sum_k^c t_k^n \ln y_k^n + \frac{1}{2}\alpha \mathrm{w}^2$$ Lemma 10

4.3.3.4 C-1 classes

There is an extra benefit of using dependant output. The Softmax function has a build-in redundancy which means that we can "save" one output neuron thus reducing the number of required network connections significantly if the second last layer is large.

The modified Softmax function is computed by Lemma 11.

$$y_k = \frac{\exp(\phi_k)}{\sum_j^{C-1} \exp(\phi_j) + 1} \quad , \quad \text{for } k = 1, 2, \ldots C-1$$

Lemma 11

$$y_C = 1 - \sum_k^{C-1} y_k$$

This also influences the error out function which is then computed as in Lemma 12.

$$E_k = \ln\left(1 + \sum_k^{C-1} \exp(\phi_k)\right) - \sum_k^{C-1} t_k \phi_k$$

Lemma 12

4.3.4 *Training functions*

In this project however we have chosen to first train with the Pseudo Gauss Newton algorithm. Gradient descent is therefore used *after* we have performed a number of Newton steps (iterations).

4.3.4.1 Gradient descent functions (with fixed stepsize)

The most common way of training a MLP is to update the network by changing the weights in the direction that minimizes the error as in Lemma 13 by using gradient descent.

$$w_{new} = w_{old} + \Delta w_{new}$$

Lemma 13

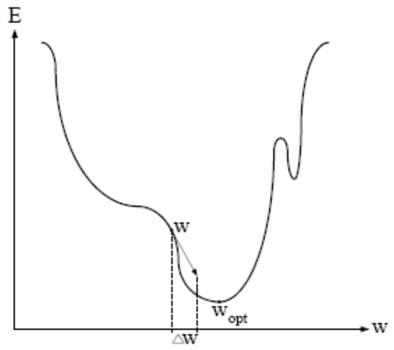


Figure 30 - Weight update

One way of determining this direction is to move the weights w in the opposite direction of the error function gradient as seen in Figure 30.

This is computed by finding the derivate of the error function with regard to the weights as in Lemma 14. Note that n is the length of movement or step size.

$$\Delta w = -n \frac{\partial E(w)}{\partial w}$$

Lemma 14

As we use the cross entropy cost function and all the output nodes thus are dependant, it is extremely difficult to find the first order derivative. Fortunately we know at forehand that it simplifies to Lemma 15 for the output layer.

$$\frac{\partial E(w)}{\partial w} = \ldots = y - t$$

Lemma 15

The function is typically augmented by a weight decay term $\frac{1}{2}\alpha \mathrm{w}^2$ which counters overfitting by preserving a preset portion of the previous network state. This leads to

$$\Delta w = -n\left(y - t\right) + \frac{1}{2}\alpha \mathrm{w}^2$$

Lemma 16

Update of the weights between the output layer and hidden layer can thereby be computed as in Lemma 17 using the activation values Z.

$$\Delta w_{kj} = -n \sum_n \left(y_k - t_k\right) z_j + \alpha w_{kj}$$

Lemma 17

Update of the weights between the hidden layer and input layer is reliant on backpropagating the error which thus is computed as in Lemma 18.

$$\Delta w_{ji} = -n \sum_n \left(\left(1 - z_j^2\right) \sum_{k=1}^{c} w_{kj} \left(y_k - t_k\right) \right) x_i + \alpha w_{ji}$$

Lemma 18

Finally the weight update can be augmented by a momentum β as in Lemma 19.

$$w_{new} = w_{old} + \Delta w_{new} + \beta \cdot \Delta w_{old}$$

Lemma 19

This momentum helps to counter the tendency of “sum of squares” error function to oscillate in local minima as in Figure 31.

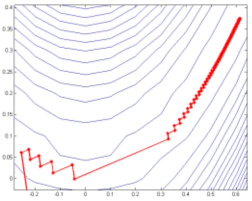


Figure 31 - Momentum and local minima

4.3.4.2 Pseudo Gauss Newton functions (with line search)

An improved method for faster convergence than gradient descent is the Pseudo Gauss Newton which is a second order algorithm.

This algorithm is building on a Tayler expansion around $\hat{w}$ of the second order derivate of the error function $E(w)$ from Lemma 7. The expansion is seen in Lemma 20.

Assume that $g_{\hat{w}}$ is the gradient of the error function and $H_{\hat{w}}$ is the second order derivative also called the Hessian.

$$E(w)=E(\hat{w})+(w-\hat{w})' g_{\hat{w}}+\frac{1}{2}(w-\hat{w})' H_{\hat{w}}(w-\hat{w})+...$$

Lemma 20

Just like in gradient descent we need the first order derivative which is seen in Lemma 21

$$\nabla E(w)=g_{\hat{w}}+H_{\hat{w}}(w-\hat{w})$$

Lemma 21

Opposite the gradient descent in which we just make a fixed step size in the negative direction of the gradient (which could result in either slow convergence or overshooting) we instead try to find the local minimum $w=w_0$ defined by $\nabla E(w_0)=0$. This means that the gradient should be zero in this spot.

The local minimum with w_0 isolated is computed as in Lemma 22.

$$w_0=\hat{w}+(-H_{\hat{w}}^{-1}g_{\hat{w}})$$

Lemma 22

The Pseudo Gauss Newton however only uses an approximation to the Hessian (hence the name PSEUDO Gauss Newton). More precisely it uses the diagonal of the Hessian. This means that the weight update for w_i can be computed by Lemma 23.

$$\Delta w_i=-\frac{\partial E}{\partial w_i}\bigg/\frac{\partial^2 E}{\partial w_i^2}$$

Lemma 23

This way an equivalent to line search is performed.

4.3.5 Activation function

The activation function is the core of making a network nonlinear. As we use binary position coded targets the activation function has to reflect this. The activation function which in this project has been denoted $g(\cdot)$ can theoretically be any nonlinear function.

In this project we use the *logistic sigmoid* function as seen in Figure 32. It is computed by Lemma 24.

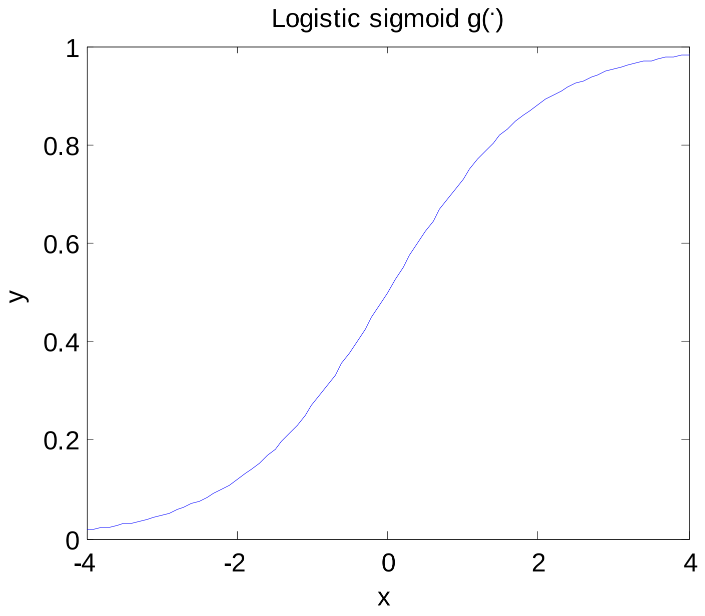


Figure 32 - Hyperbolic tangent

One of the main properties of the logistic function is of cause the nonlinear mapping between input and output values.

Another important property is the spectrum to which it maps. The activation will be saturated to one of the two possible binary target values if the input is large enough (and has the correct sign).

This property makes this particular activation function robust in handling input patterns which has very high input values but only contains little information.

$$g(x) = \frac{1}{1+e^{-x}}$$

Lemma 24

4.3.6 Pruning function

The concept of pruning a MLP network is to kill irrelevant network connections. This way the network is able to discriminate between relevant and irrelevant input values. An example is the classification of having diabetes or not. Assume the input pattern is weight, age and hair color. Then the pruning algorithm should kill the weights related to the hair color parameter as it only offers little or no information.

The idea is to find weights that have no significant contribution both in the forwarding and back propagating steps in relation to target output t_k and observed output y_k. Such weights are candidates for pruning.

The pruning is performed when the entire training set (all patterns) has been iterated.

From now on let m be the layer index and x_i be the output from neuron i in the previous layer (m-1).

4.3.6.1 Forward weight contribution

The forward weight contribution of w_{ij}^{m} to the total activation $\sum_i x_i w_{ij}^{m}$ can be computed by $x_i w_{ij}^{m}$.

4.3.6.2 Backwards weight contribution

The error contribution of w_{jk}^{m} to the total error correction $\sum_k w_{jk}^{m} \delta_k^{m}$ can be computed by:

$w_{jk}^{m} \delta_k^{m}$ for the output layer where $\delta_k^{m} = -\left(t_k - y_k\right) y_k \left(1 - y_k\right)$

or

$w_{jk}^{m} \delta_j^{m}$ for input and hidden layers where $\delta_j^{m} = \sum_k w_{jk}^{m+1} \cdot \delta_k^{m+1}$

This contribution is directed at the weight w_{jk}^{m-1} (the previous layer).

4.3.6.3 Pruning factor

Now we can look for weights which contributions only marginally to the training error. This is done when the entire training set has been iterated as mentioned earlier.

In order to decide if a weight is contribution sufficiently we use the pruning factor F_p which is defined as:

$$\left[0 < F_p < 1\right]$$

If a weight w_{ij}^{m} meets both the conditions in Lemma 25 then it is pruned. This is done for all training patterns.

$$\begin{aligned} \left|x_i w_{ij}^{m}\right| &< \left|\sum_i x_i w_{ij}^{m}\right| \cdot F_p \\ \left|w_{ij}^{m} \delta_j\right| &< \left|\sum_i w_{ij}^{m} \delta_j\right| \cdot F_p \end{aligned}$$

Lemma 25

It is clear that when $F_p \rightarrow 1$ then more connections are pruned and likewise when $F_p \rightarrow 0$ fewer connections are pruned.

4.3.6.4 When to prune decision by temperature function

There is however some obvious problem with this methods.

1. If we start to prune immediately then the network could becomes unstable without chance of recovery. The first layer(s) doesn't absorb much of the error in the first iterations which means that some connections could be subject to pruning although this would later prove to be a bad idea.
2. If we choose a fixed low pruning factor then the pruning won't have much impact.
3. If we choose a fixed high pruning factor then problem 1 occurs. Also we risk removing important connections.

To solve this we use a temperature function which gradually increases the pruning factor. The idea is that the pruning factor is very low or zero in the first iterations. When the network begins to stabilize or converge towards the final weights, then the pruning factor increases alongside. The temperature function of choice is seen in Lemma 26 where n is the iteration number.

$$T(n) = \frac{C}{\log(1+n)}$$

Lemma 26

The C parameter is the initial temperature factor. It is very important because it allows us to control the behavior of the temperature function.

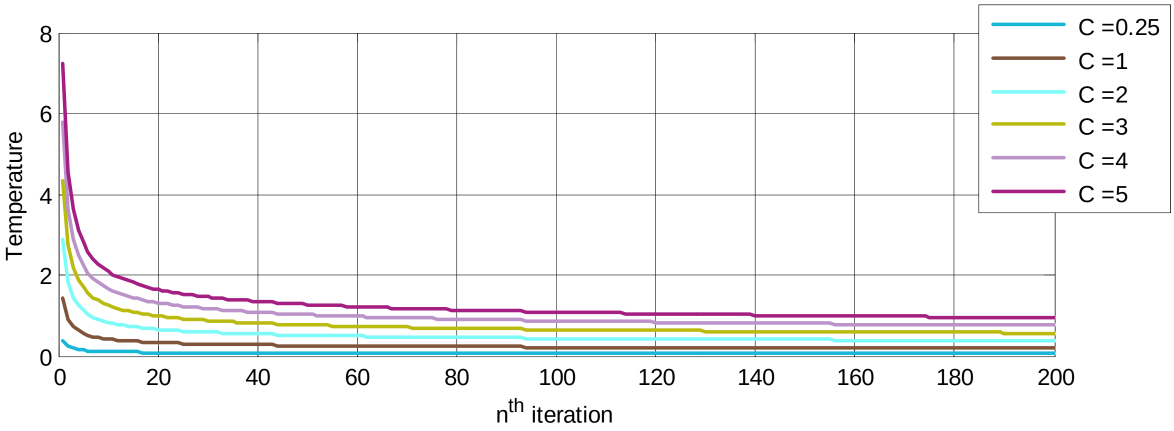


Figure 33 - Temperature as function of iterations

We are however interested in the opposite scheme where the temperature starts low and rises over time. Also the temperature $T(n)$ *can* be higher than 1 as seen in Figure 33 and would thus violate the definition of F_p if assigned to it. Although Lemma 25 would still work for $\left[0 < F_p < \infty\right]$ we assign it as in Lemma 27.

$$F_p = 1 - \begin{cases} T(n), & \text{if } T(n) \le 1 \\ 1 \quad , & \text{otherwise} \end{cases} \quad ; \text{ for all iterations } \{\text{n=1...N}\}$$

Lemma 27

This produces the modified temperature function as seen in Figure 34. Notice the mutual relation between $T(n) > 0$ and θT . We expect that networks which takes many iterations to stabilize also needs a softer transition in the pruning factor increment. Thus the lower angle of T and choice of this temperature function.

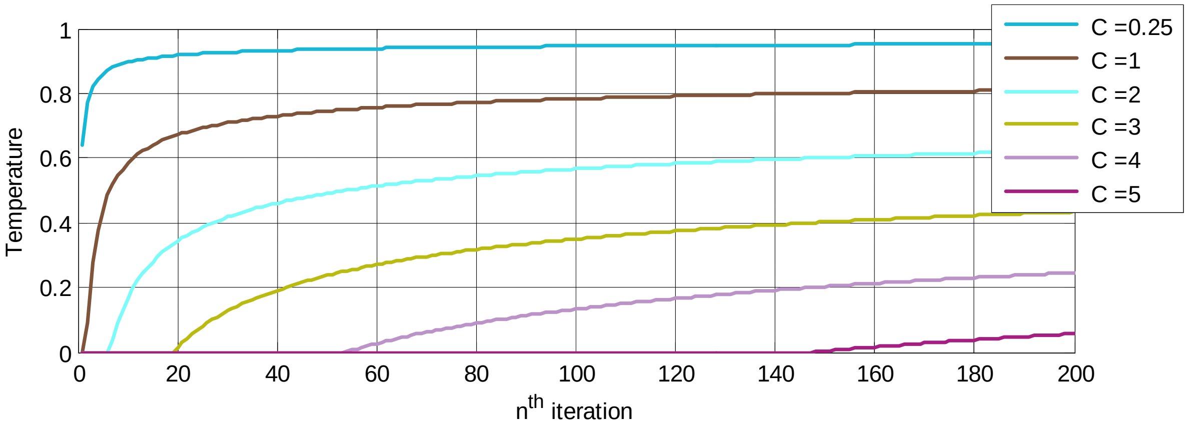


Figure 34 - Modified temperature as function of iterations

This way it is possible to adjust when the pruning takes effect and how steep the increase is.

4.3.7 A comment on the network implementation we use

Parallel in time to this project, the authors have developed a Win32 implementation of the multilayer feed forward neural network with pruning, linesearch and momentum as key components.

The network can be training using both Pseudo Gauss Newton and Gradient Descent. Also the network has a built in shuffle routine for the training set. Finally the network is able to train on a user defined number of training set samples which greatly increases speed and also reduces the problem of fitting to certain training set samples (typically the last trained samples).

This network was used for a Win32 application for optical number recognition where the user can draw a “handwritten” number in the application and the program then classifies it.

An important feature of the implemented network is that the layer sizes can be chosen simply based on a vector containing the desired sizes e.g. [10 50 30 2] for a network with 10 inputs and 2 outputs with two hidden layers of size 50 and 30.

4.4 Test of the MLP networks

In this section we try to test the network under different configurations. As the network we use has an excessive amount of tweaks, we have chosen to test on the combinations we deem important. Finally we put it all together to find the "true" capabilities of the nn.

As mentioned before, the possible parameter space is extremely large so it is not possible to test all combinations of parameters. In the following sections we use the default setup defined in section (2.2) unless other is stated. The default setup is chosen such that it produces empirically good results.

Some parameters are expected to be mutual dependent (covariance). For these parameters we test them against each other and use the default setup for the rest of the parameters.

The other parameters are tested individually using the default setup for the rest of the parameters.

4.4.1 MFCC count ↔ Sliding windows size

In Figure 35 we can see how the pattern size relates to the number of patterns used for NN input at a time. The idea is to use the context to improve the success rate. Remember that at MFCC count = 20 and 200 ms sliding window we actually uses $200/10\cdot 20-1$ because we compute the 20 MFCC each 10th millisecond.

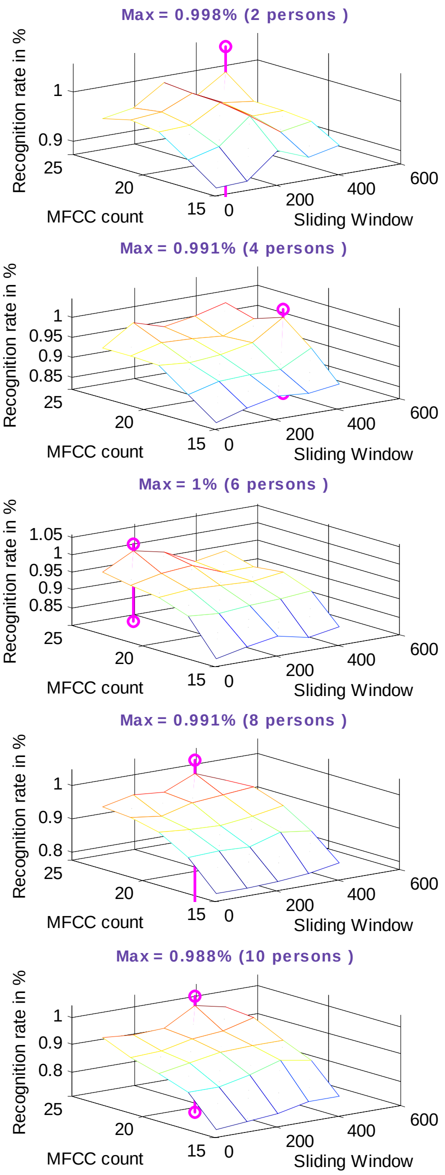


Figure 35 - MFCC count vs. Sliding windows size

Opposed to the GMM results, the graphs indicate that the network is less dependent on how large a context we. It is clear that it is the MFCC count that dominates the recognition rate. We expect this is because a NN can model far more complex functions and thus require less context. There are however a trend that too large input window, decreases recognition rate slightly. Also notice that the pruning had a far smaller impact on recognition rate than the previous example. This is likely due to how the network weights converged. But in essence it is unpredictable as we don't know at what temperature weights are updated.

4.4.2 MFCC count ↔ Hidden layer size

In Figure 36. we look into the relationship between MFCC count and the size of the first hidden layer (we only use one hidden layer in this project as two layers haven't improved results. The hypothesis is that the spectral features of an MFCC vector can be more precisely modeled when using a network capable of estimating more complex functions. We have chosen to mark recognition rates which differ up to 10% of the best recognition rate.

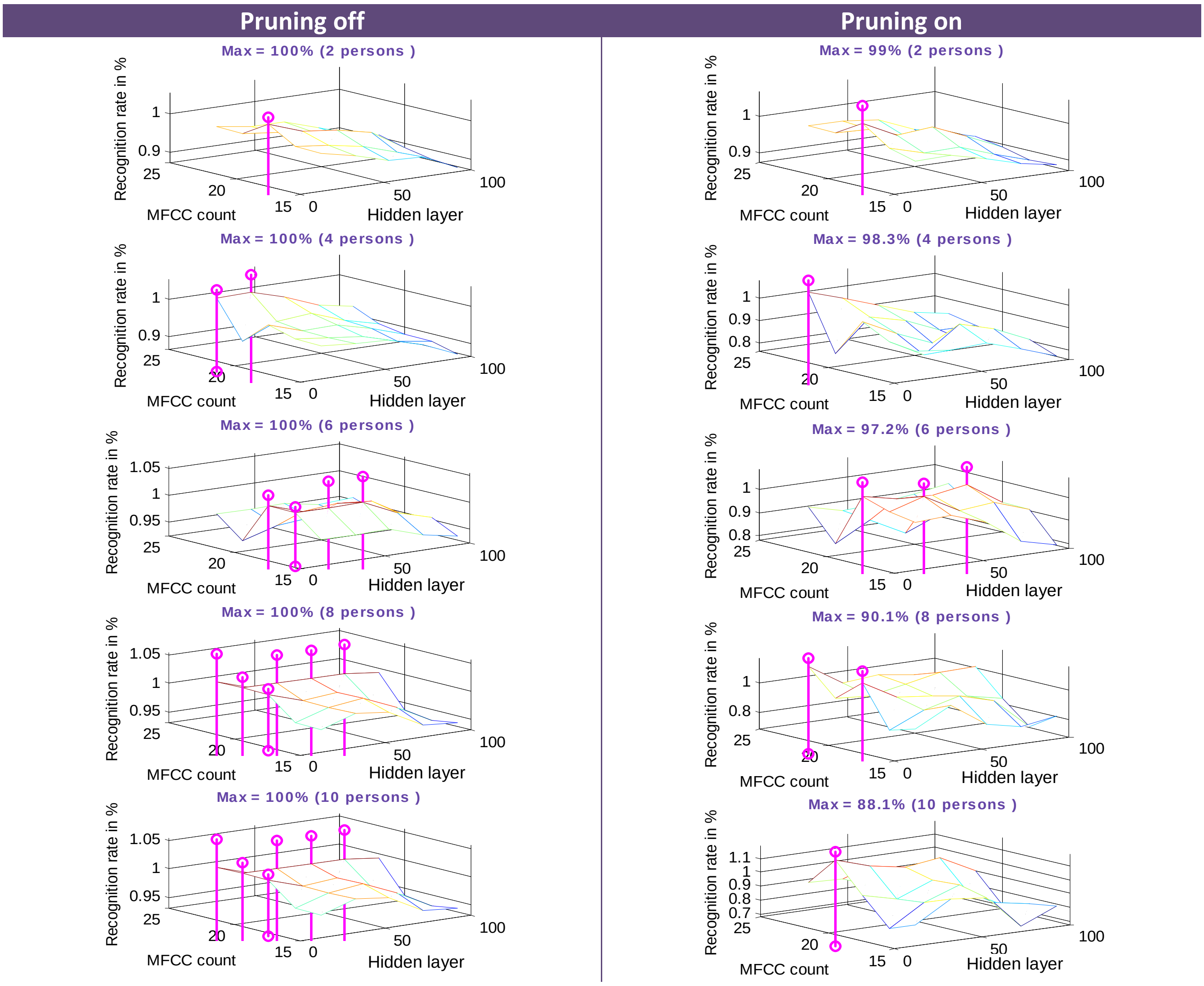


Figure 36 – MFCC count vs. Hidden layer size

From the graphs, it is clear that the MFCC count and hidden layer size is connected. When we use few MFCC coefficients we get the best results when using no hidden layer. Opposite, when the MFCC count grows, the recognition rate is still high when using a hidden layer. If the hidden layer size is to large then the recognition rate drops, except when using many MFCC coeffcients. This is due to overfitting. The recognition rate drops to near 25% in worst configurations. Also note that pruning appears to reduces some redundancy in the network but at a loss of recognition rate. This can be seen from the recognition rates. There are fewer combinations that yield good results. The overall recognition rate in this example was very poor when using pruning.

4.4.3 Initial weights

The activation function is relevant when choosing the initial weights. It is desireable that the initial weights are such that they can be easily updated with regard to the speed in which they converge toward optimal weights.

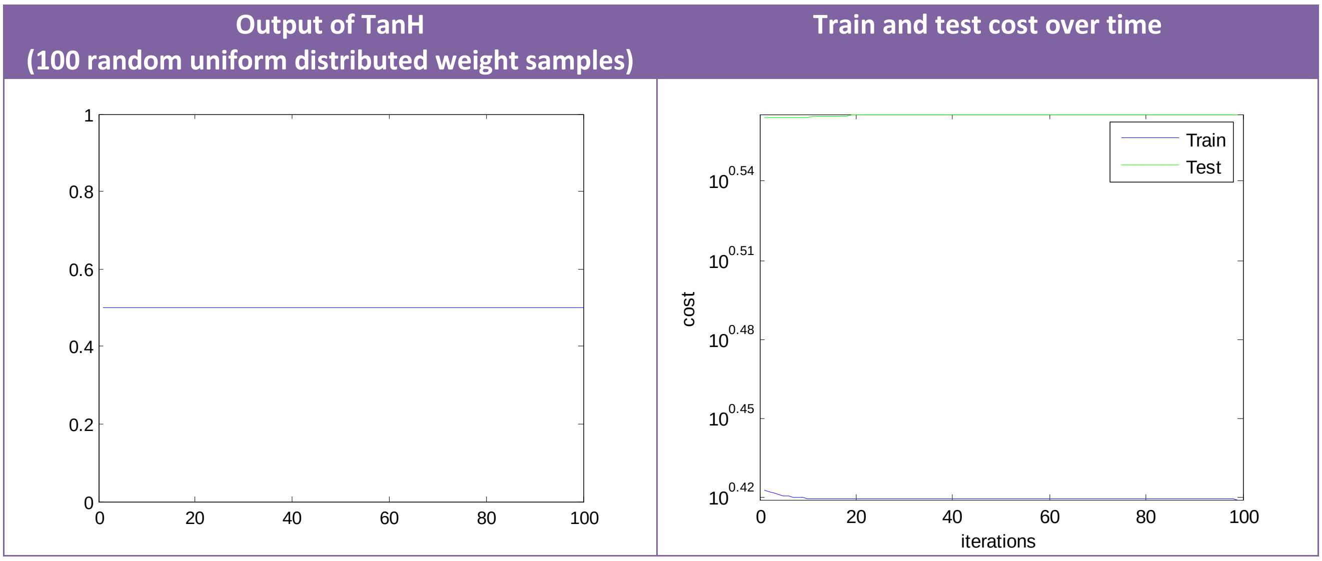
For this purpose we use a range which defines the variance of a random uniform distribution of the weights initial value. The range is the diversity from the center which we define as 0.5 in correspondence to $g(\cdot)$.

Let's look at how the range relates to the activation function $g(\cdot)$ by testing on the default network with a small hidden layer. The network is size: [3 windows * 18 input ; 5 hidden ; 3 (4) output].

Range = 0+µ

As in Figure 37 where the range is too small the weights are so alike that the activation function can't map them into a reasonable interval.

This creates problems when multiplying the input and weights because the consequence is an almost linear mapning between input and output.


Output of TanH (100 random uniform distributed weight samples)

Train and test cost over time

Figure 37 - Weight initialization range = 0

The properties are:

- Only bias has influence on the output of the network
- The low cost is not based on actual input due to only bias influence
- The network output is unaffected by the input and is thus not relevant.

Range = 10000

As in Figure 38 where the range is too large the weights are so excessive that the logistic activation function maps them into a pulse train (or almost).

This creates problems when multiplying the input and weights because the consequence is that it in reality is only the output layer weights that are updated.

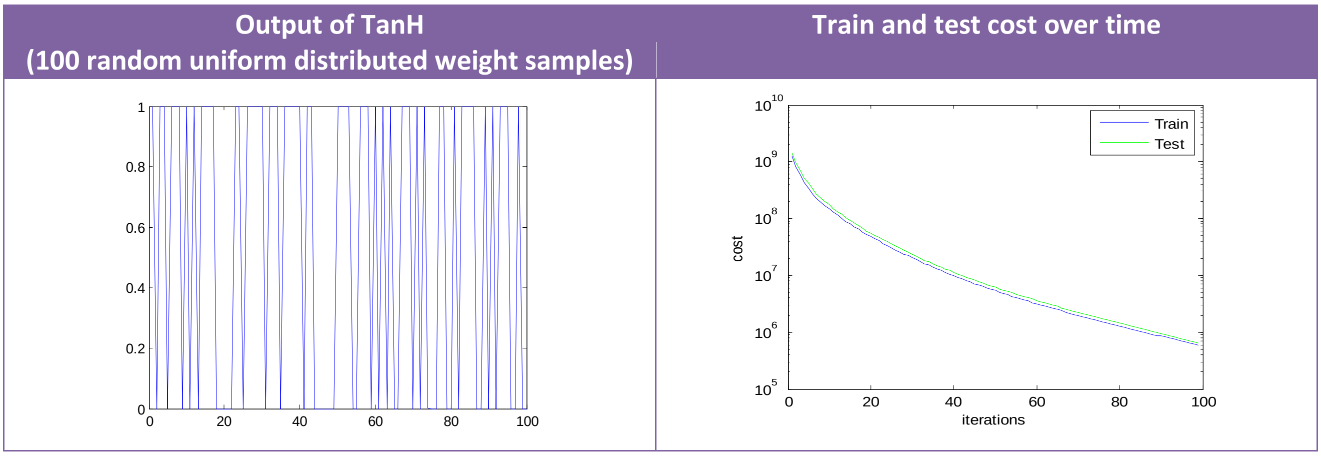


Figure 38 - Weight initialization range = 10K

The properties are:

- It takes a lot of iterations to optimize the weights.
- Range is so large that the activation is almost 1 or 0
- Activation function $g(\cdot)$ is in effect almost a step function rather than nonlinear.

Range = 0.5

As in Figure 39 where the range is suitable. Neither a linear or pulse train like output is created.

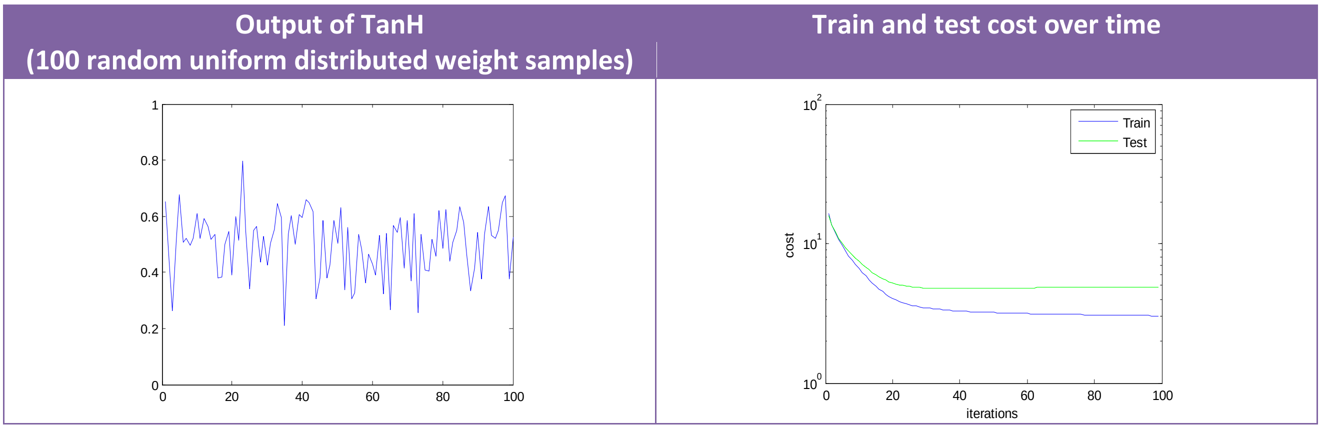


Figure 39 – Weight initialization range = 0.5

The properties are:

- Range is within the outer extremes of the activation function.
- It is faster to update the weights towards achieving a low cost.

4.4.4 Pseudo Gauss Newton steps and Gradient Descent stepsize

Here we look at how many Pseudo Gauss Newton steps should be performed before switching to Gradient Descent. We try with 2 and 6 classes (persons) respectively.

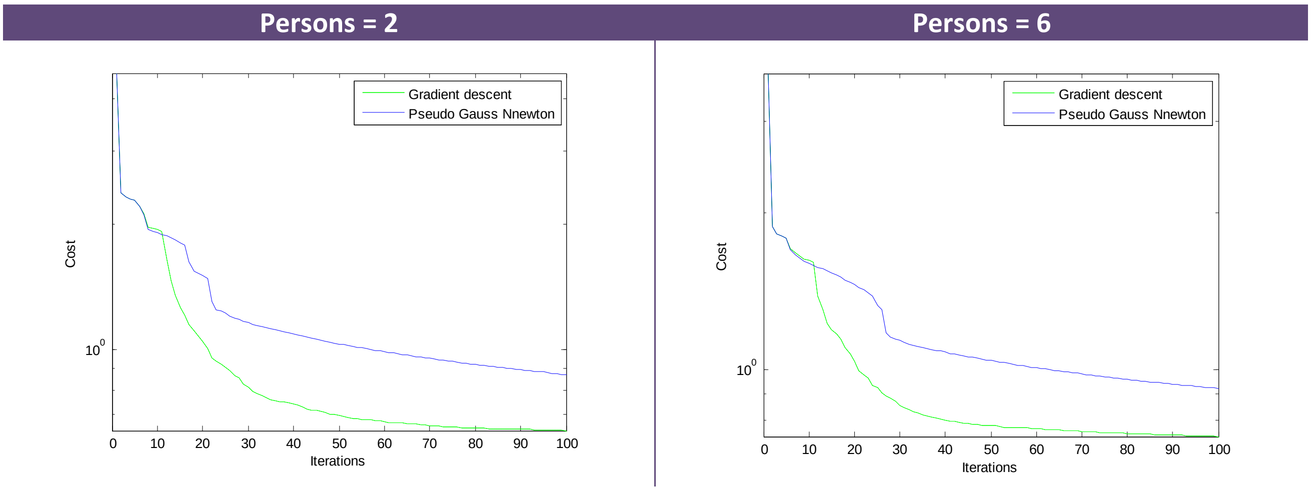


Figure 40 - Netwon and GD steps

To find out when to switch from PGN to GD, we initialize GD after 10 PGN steps and clone the network weights. Then we train each of the clones continuing from iteration 10. This is shown in Figure 40.

PGN decreases the error a lot faster than GD in the first 10 iterations. But afterwards, it is more suitable to use GD as it is better at converging towards local minima. PGN is better at detecting the approximate area of such local minima than the GD but doesn't converge towards the minima nearly as fast as GD.

We can see that after 20-30 PGN steps, it the PGN flattens out. This is important as we use the second order derivative of the gradient to estimate when to switch between PGN and GD. We also use it to estimate when to stop iterating based on predefined minimum gradient norms. The gradient norms corresponding to Figure 40 is shown in Figure 41. Also notice how similar the training events occur. The network is almost invariant to using 2 or 6 classes. This is surprising, but is likely due to the pruning and weight decay. More connections are pruned for 2 persons than for 6 persons.

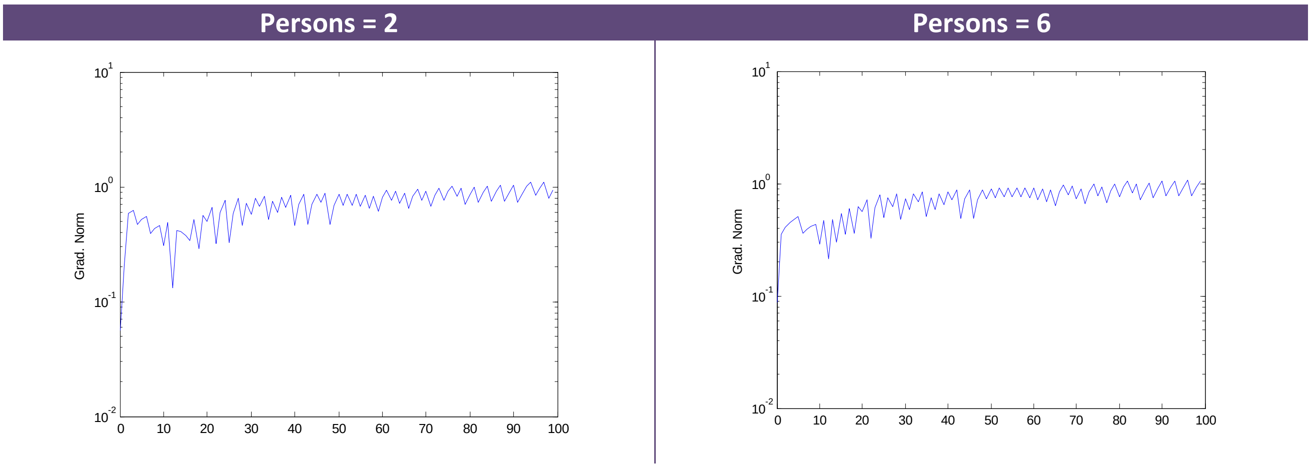


Figure 41 - Newton and GD grad norm

4.4.5 Weight decay

Let's look at how the weight decay influences the error. The theory is that the decision boundary should flatten when weight decay α increases. This is because the weight decay function limits the convergence to minimum by maintaining a part of the previous state. This way the network can't model the hyper-function of the newest input pattern(s). This is illustrated in Figure 42.

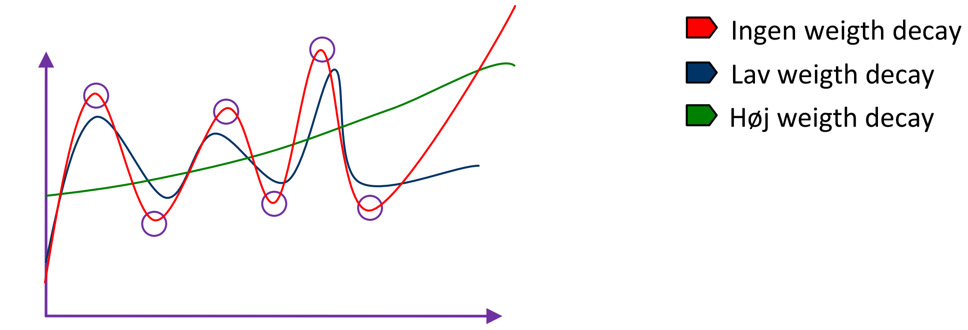


Figure 42 - Weight decay princip

Using the default setup on 2 persons, we can investigate the decision boundary. Figure 43 illustrates how the decision boundary is not fitted to the particular training patterns (blue x'es). This way we counter overfitting to the training set.

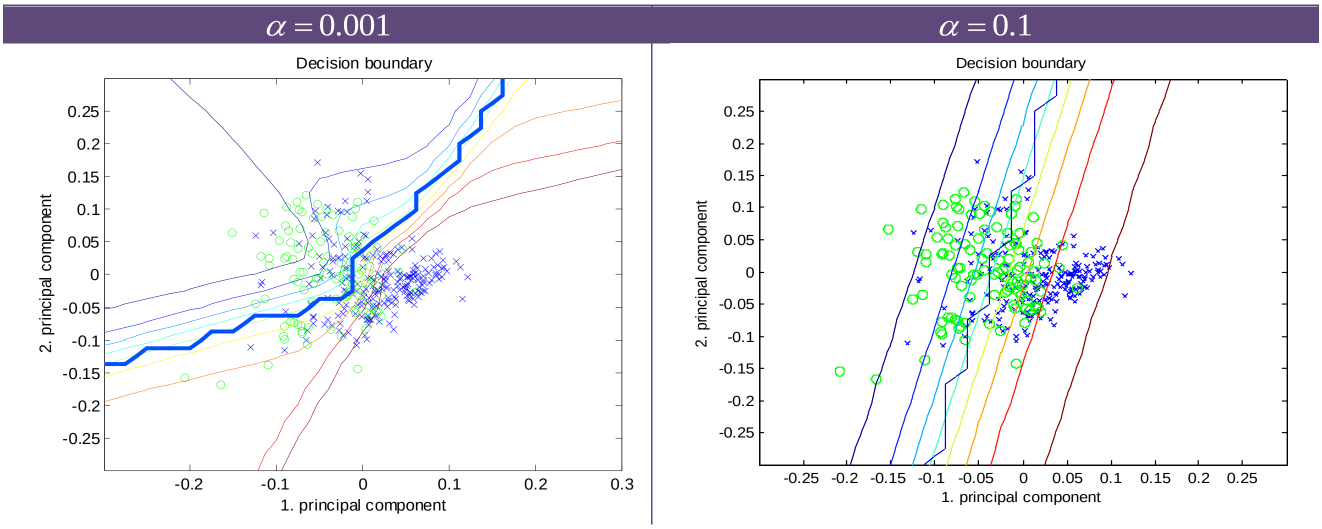
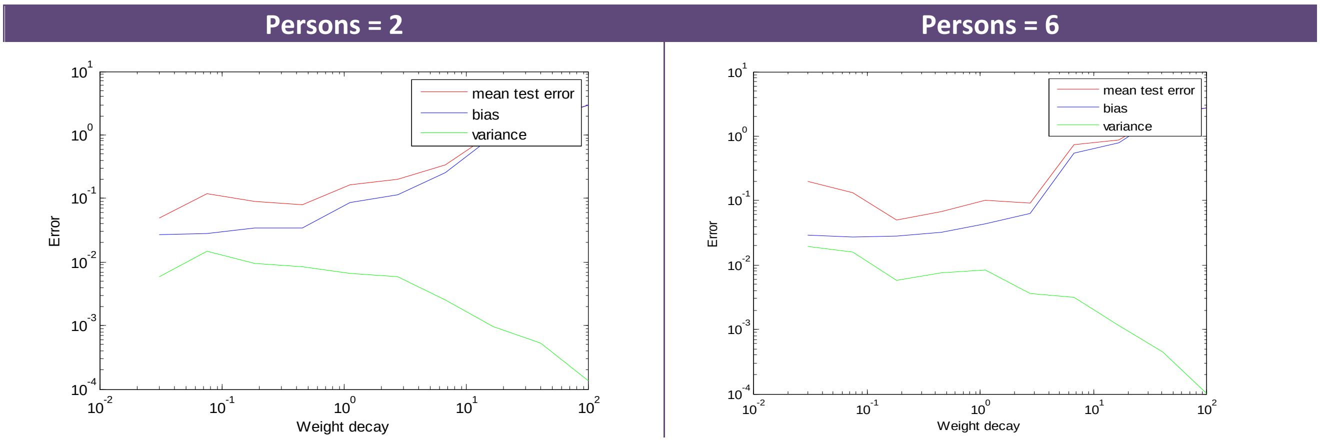


Figure 43 - Weight decay and decision boundaries

The error is closely linked to the weight decay as seen in Figure 44. But the weigh decay is closely linked to both the bias and variance. E.g. before $\alpha = 10^0$ the error mainly dependant on the variance as the bias is nearly stationary. But as the bias increases the error is more dependent at the bias. This indicates that the weight decay should be found in the region where bias and variance is lowest. As these results are quite time consuming to compute we have only tested on 2 and 6 class network respectively.



Persons = 2

Persons = 6

Figure 44 - Weight decay with bias and variance

From Figure 44 it appears that the best results are achieved with $\alpha = 0.2$

4.4.6 Generel overfitting

Overfitting most often an issue when training on datasets that are very specifik in nature. Typically for regression networks. In this project the patterns (features) are more generalized, so overfitting has not caused big troubles so far. This is also true for the GMM model. There are however some issues that can couse the network to overfit. Namely: too many neurons in hidden layer and training too many iterations.

To illustrate some overfitting, we first train the network using default setup and 40 Pseudo Gauss Netwon steps which give a reasonable approach to a local minimum. We then use a very small stepsize for Gradient Descent ($eta = 0.001$).

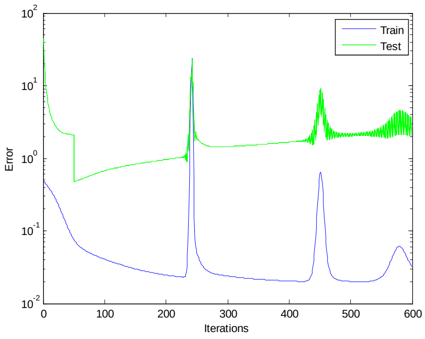


Figure 45 – Overfitting

From Figure 45 we can see that after the 40 PGN steps the test error drops significant. This is likely because the first GD step is able to minimize the error significantly by moving weights in the opposite direction of error.

The reason why PGN doesn't do this is that it moves in an elliptic like direction around the minimum valley. An example of how PGN and GD updates in weight space is shown in Figure 46.

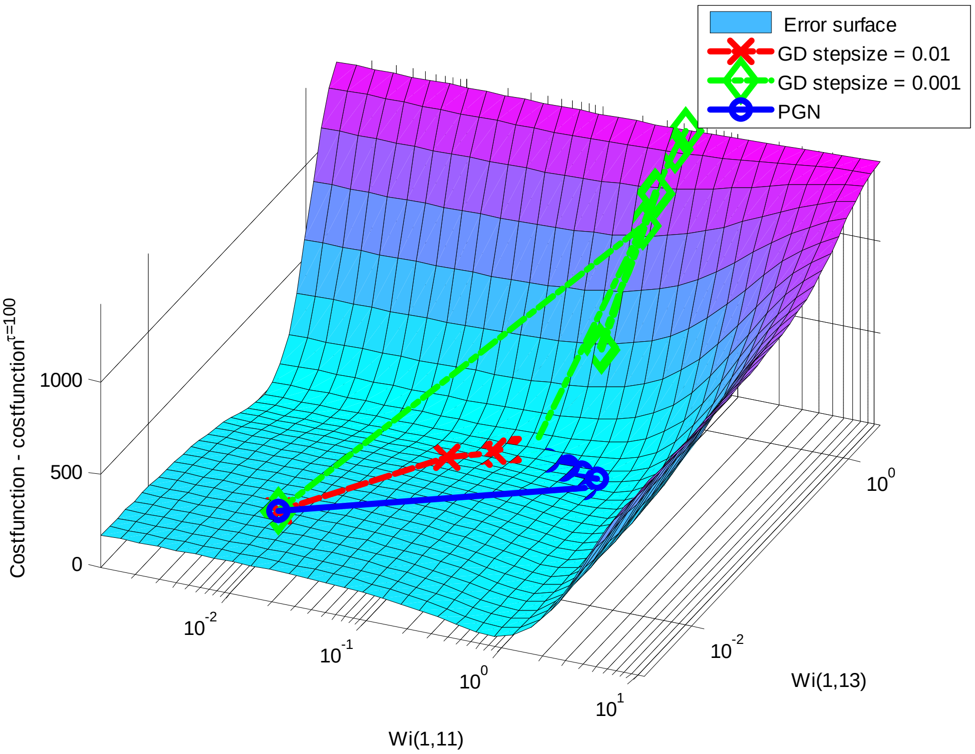


Figure 46 - Gradient directions of GD and PGN

4.4.7 Pruning

Pruning involves the pruning factor which we are mapping from a temperature function. The temperature function is controlled by an initial temperature T that controls the behavior of temperature and thus also the pruning factor. The interesting part is therefore to evaluate how different values of T affects the error. For this we use the default setup.

We train the network 5 times with 2…10 classes (persons). Each network is stopped when the gradient norm changes less than 1e-001 (trained on different number of iterations).

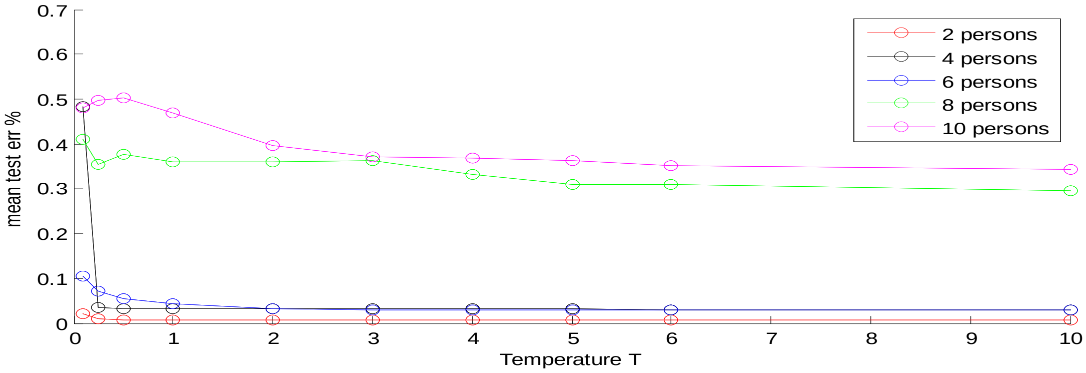


Figure 47 - Temperature vs. Error

From the results in Figure 47 we can see that a low temperature gives a little higher error rates. This is because the network is pruned earlier when the temperature is low. Early pruning tends to remove connections indiscriminant of how important they are, as the network is unstable in this phase. In Figure 48 we show an example of how pruning works and affect a neural network.

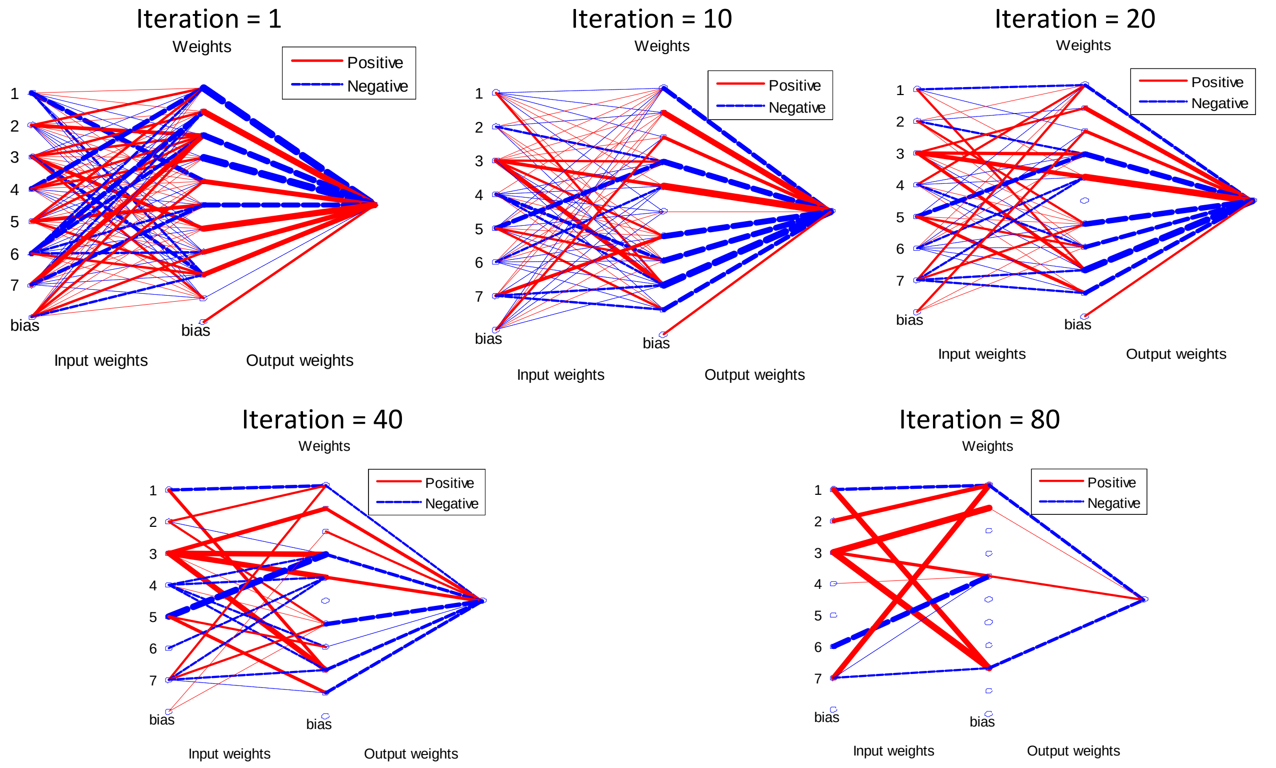


Figure 48 - Connections being pruned

4.5 Overall performance

This section describes the overall best performance of the MLP network. The recognition rates are based on the best combination of parameters compared to the number of speaker models. These parameters are synonymous with the ones used in section (4.4.2) where we got a recognition rate of 100%.

Although the best parameters differ from 2-10 class problems, there are some general trends, namely:

- Temperature = 2
- No pruning
- No hidden layer
- 400 ms sliding windows
- 24 MFCC
- Weight decay = 0.001;

The rest of the parameters vary and can be read from the previous sections.

The signals used for training and test are equal to the sentences used in the overall performance chapter in GMM.

Report:

Implementation

Figure list:

1) Introduction

In this report we present the Win32 application developed for speaker identification.

The focus is on the structure and how processes are connected during runtime.

Just to give an impression of the program developed for this project we include a screen dump in Figure 1.

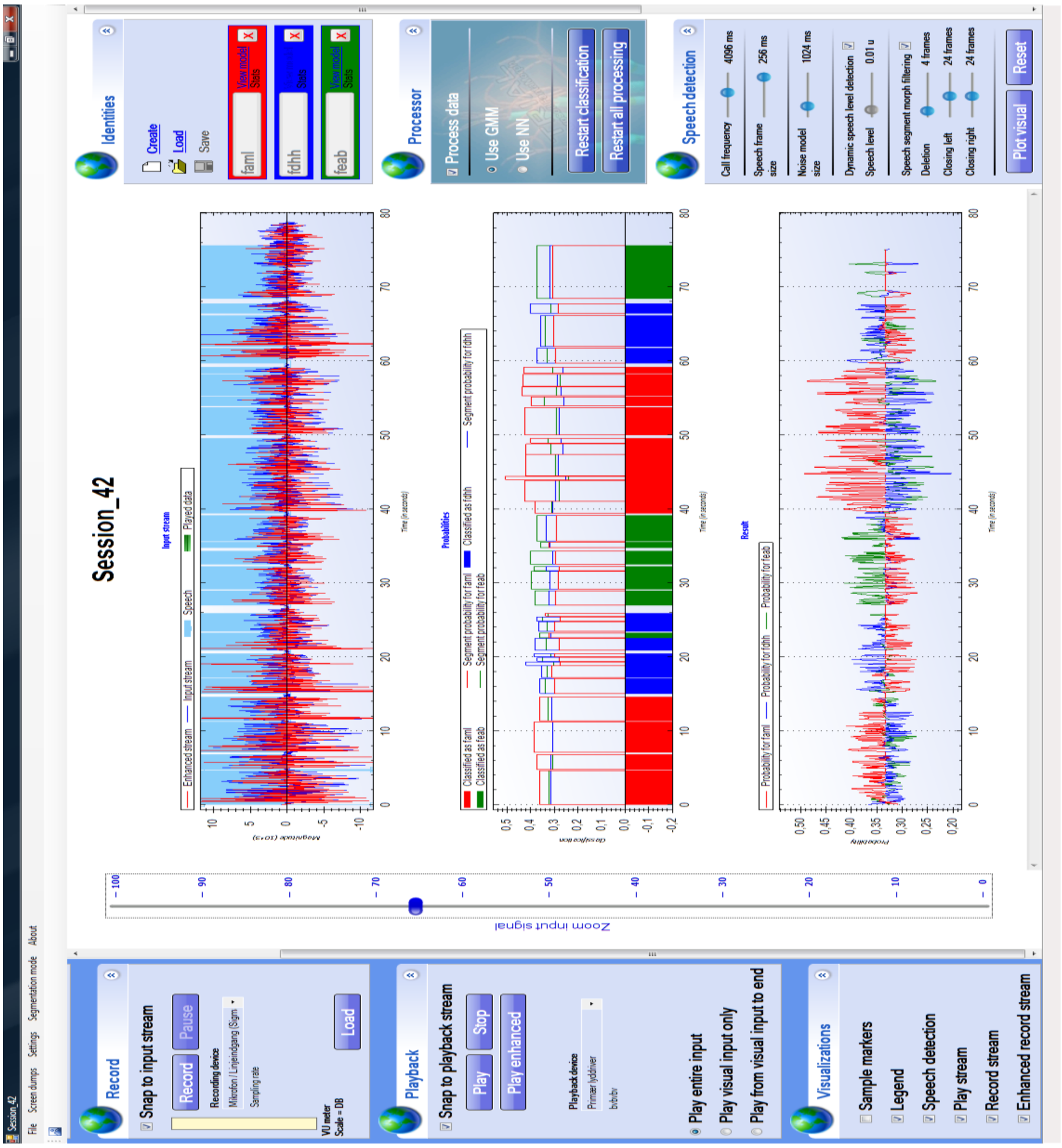


Figure 1 - Overview of Signal & Feature analysis

2) Component overview

In Figure 2 we show the primary clusters in the program and how they are tied together.

As the software is multithreaded we use a thread manager to control access rights to memory items. It is also responsible for managing the sequence in which data is processed.

The user can via the user interface send requests to the thread manager which then perform these by forwarding commands to the appropriate clusters.

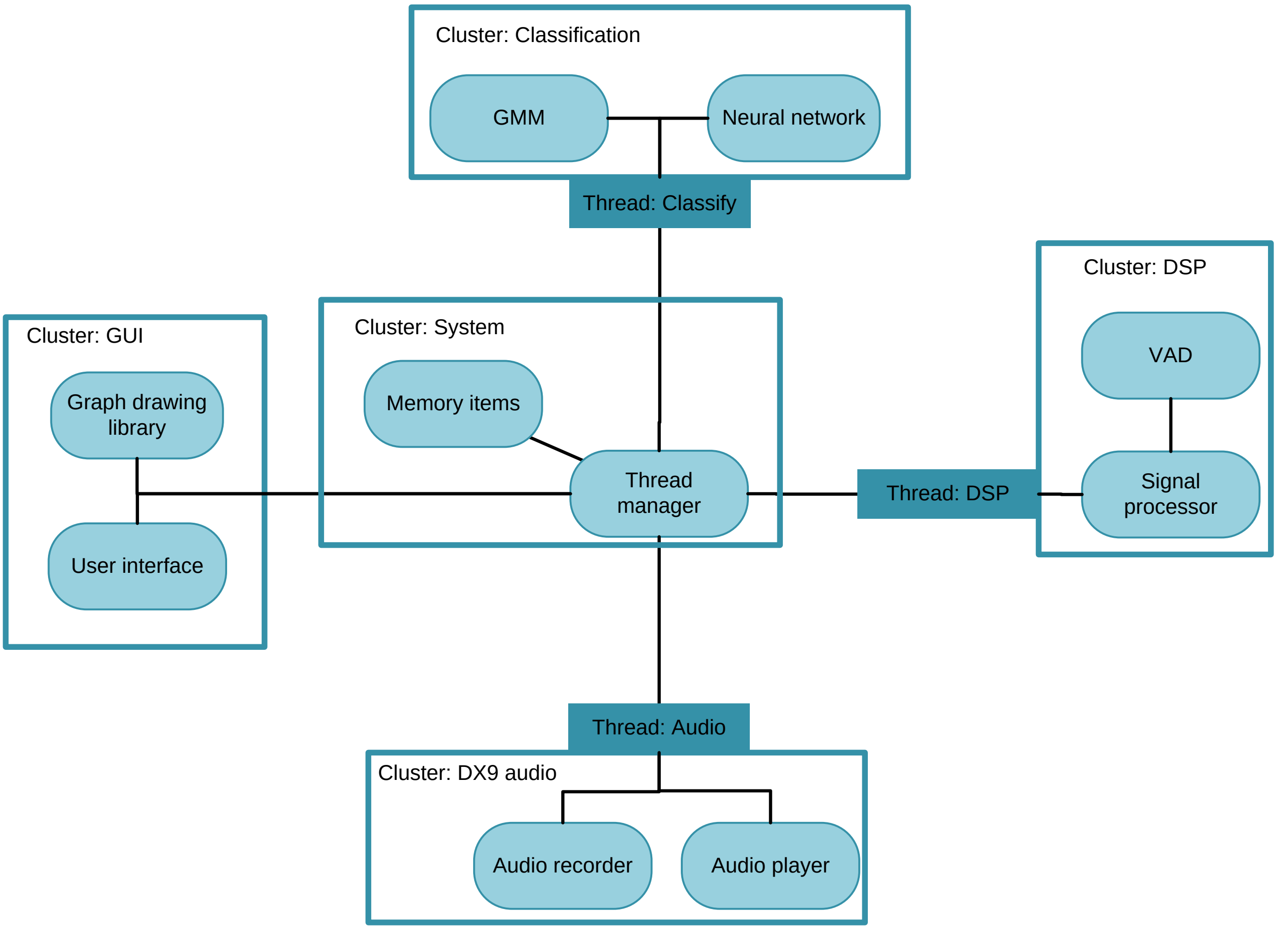


Figure 2 – Clusters

As seen in Figure 1 we use 3 additional threads (the GUI and thread manager runs on the system thread).

3) Clusters

3.1 GUI

The GUI is primarily build of two components: The traditional user controls on screen, and a graph drawer.

The graph drawer is responsible for creating graphs that can be updated at real time as in Figure 3. Plotting all samples at real-time while they continuously is recorded is very time demanding. For instance do we need to update 16000 * 20, if we have zoom such that 20 seconds is visible on screen. As the screen resolution doesn't allow to show all 320000 samples anyway, we down sample it prior to drawing. This greatly increases performance and allow any number of time-range to be viewed. We also use a numerical trick to prioritice the samples which has the highest deviations to give a more realistic looking graph.

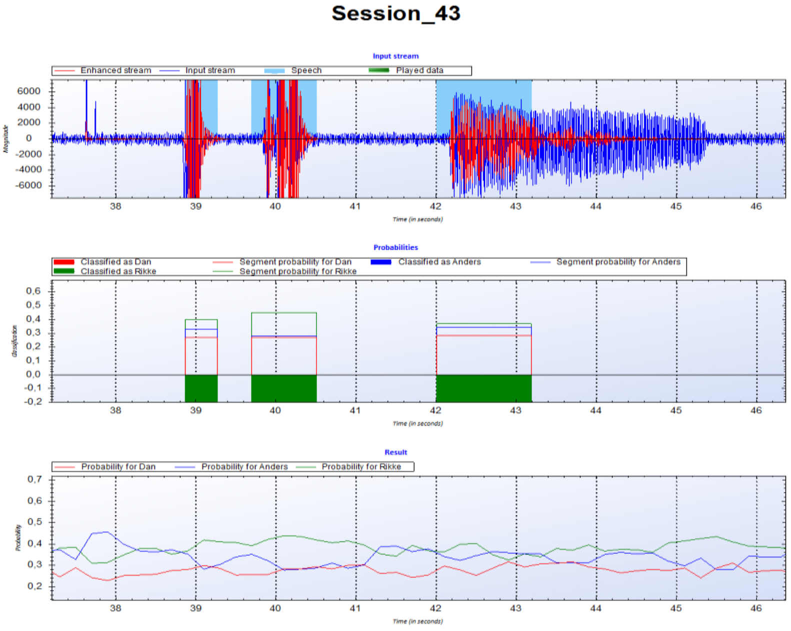


Figure 3 – drawing library example

An area of special interest is how the graphs store data. To minimize storage, the graph library data is a reference to the same lists used for signal processing, classification etc. This way we avoid storing redundant data. The access to the memory items is granted by the thread manager.

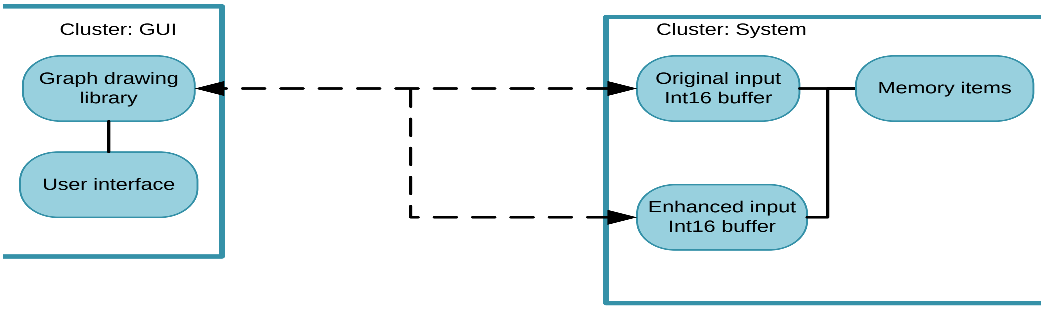
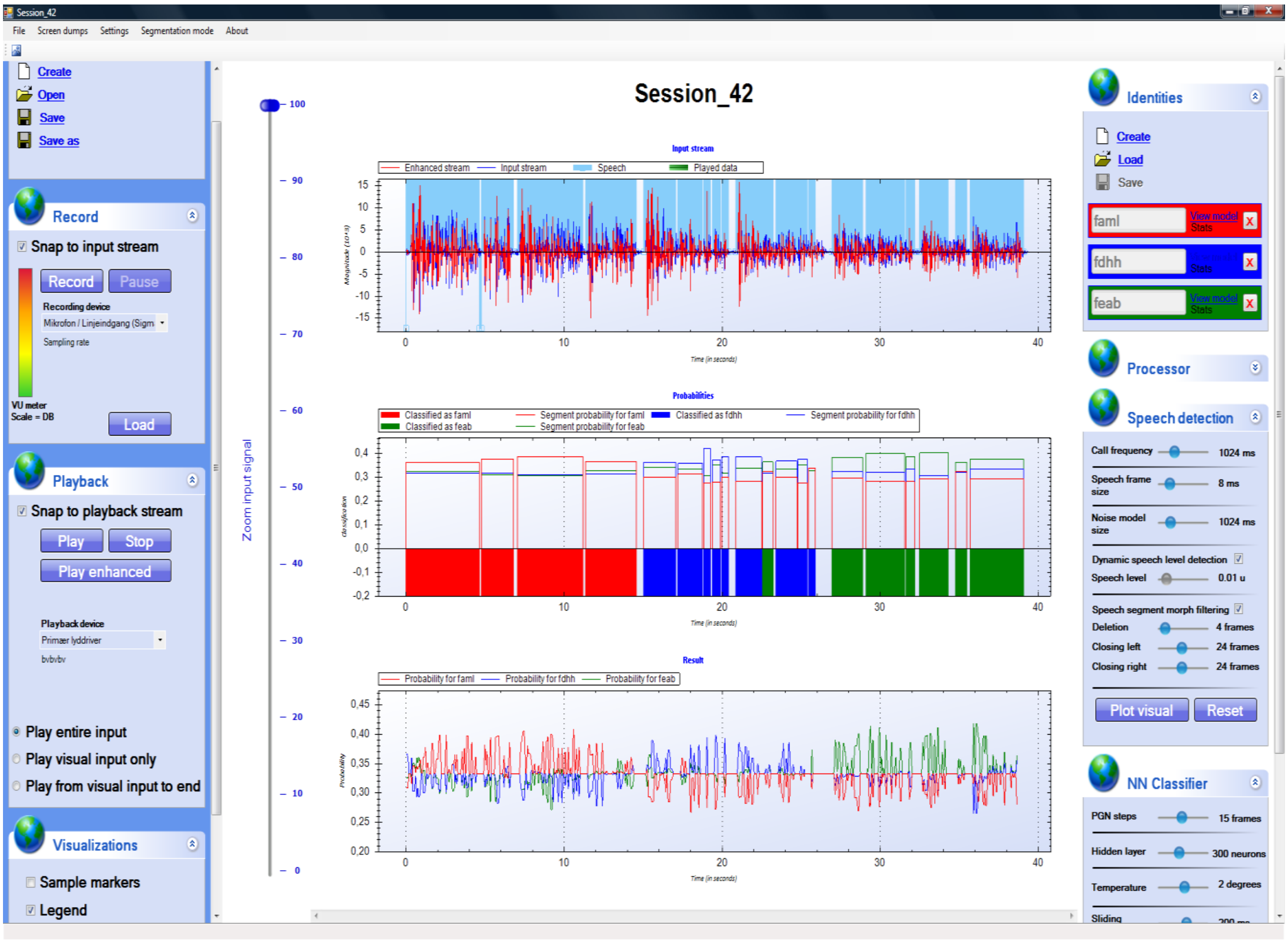


Figure 4 - Shared memory items of graphs and processing

The GUI controls are used to send messages to the Thread manager component. From the interface we have options to control most parameters related to the various other components. This is for example the behavior of front-end signal processing and the voice activity detector.

Furthermore we can play and record data from an external microphone or load data from wave files. Finally there are options to train new models and configure the classification components.



☑ Snap to input stream
Recording device: Mikrofon / Linjeindgang (Sigm
Sampling rate

☑ Snap to playback stream
Playback device: Primær lyddriver
bvbvbv

● Play entire input
○ Play visual input only
○ Play from visual input to end

□ Sample markers
☑ Legend

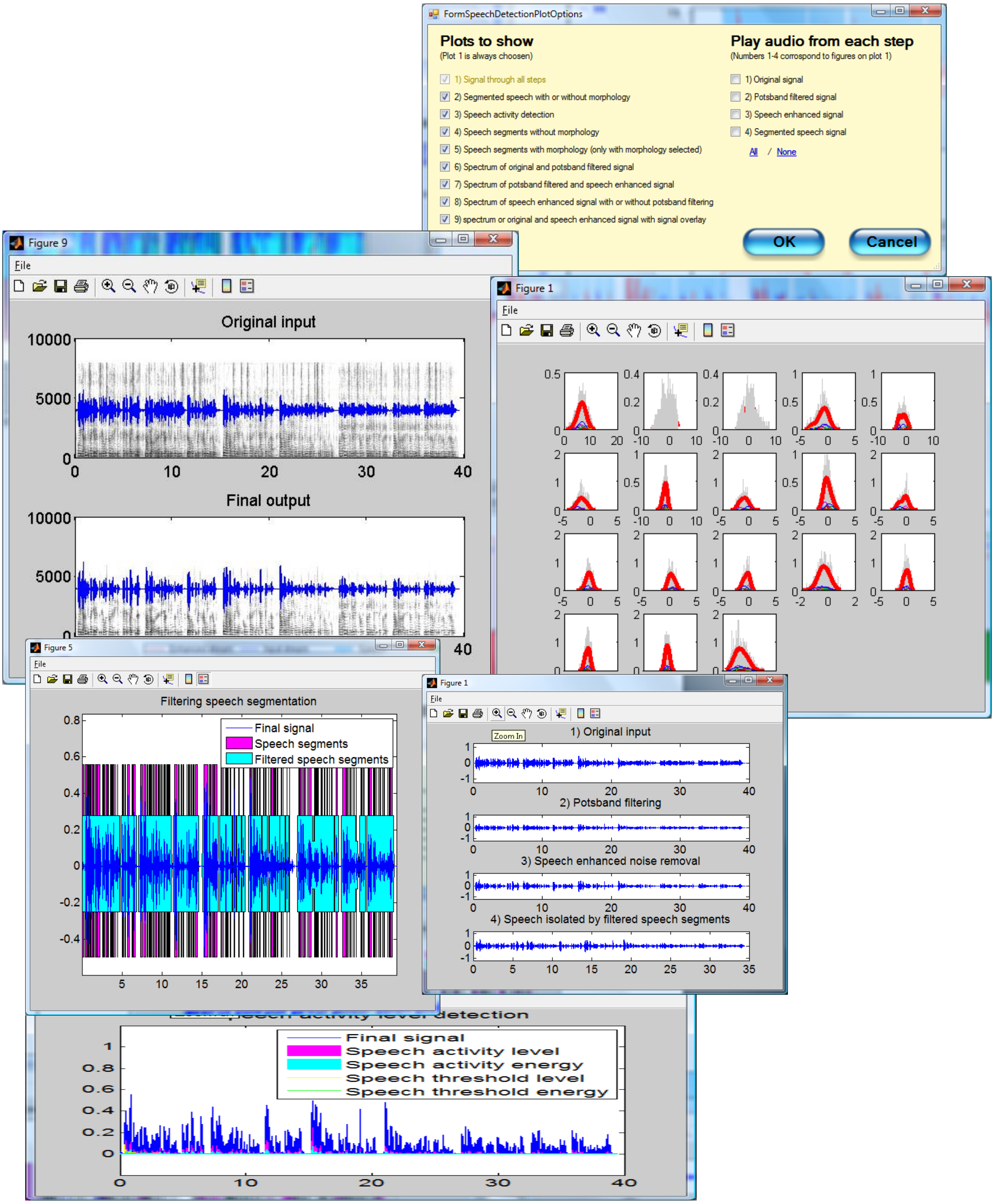
faml
fdhh
feab

Call frequency 1024 ms
Speech frame size 8 ms
Noise model size 1024 ms
Dynamic speech level detection ☑
Speech level 0.01 u
Speech segment morph filtering ☑
Deletion 4 frames
Closing left 24 frames
Closing right 24 frames

PGN steps 15 frames
Hidden layer 300 neurons
Temperature 2 degrees
Sliding 200 ms

Training of new models is performed by a 2 step procedure which involves the training, and a verification of the model.

Finally the user interface have options to access more detailed plots such as the GMM models or VAD related figures. Some of the more specialized figures are plotted via a Matlab C# wrapper. Examples of the figures the program can produce is showed in Figure 5.



FormSpeechDetectionPlotOptions

Plots to show
(Plot 1 is always choosen)

- ☑ 1) Signal through all steps
- ☑ 2) Segmented speech with or without morphology
- ☑ 3) Speech activity detection
- ☑ 4) Speech segments without morphology
- ☑ 5) Speech segments with morphology (only with morphology selected)
- ☑ 6) Spectrum of original and potsband filtered signal
- ☑ 7) Spectrum of potsband filtered and speech enhanced signal
- ☑ 8) Spectrum of speech enhanced signal with or without potsband filtering
- ☑ 9) spectrum or original and speech enhanced signal with signal overlay

Play audio from each step
(Numbers 1-4 corrospond to figures on plot 1)

- □ 1) Original signal
- □ 2) Potsband filtered signal
- □ 3) Speech enhanced signal
- □ 4) Segmented speech signal

All / None

OK Cancel

Figure 5 - Graphs from GUI

3.2 DX9 audio

The DX9 audio cluster has 3 responsibilites:

- Record input signal bytes
- Convert bytes to int16 and back
- Play selected output signal Int16 values

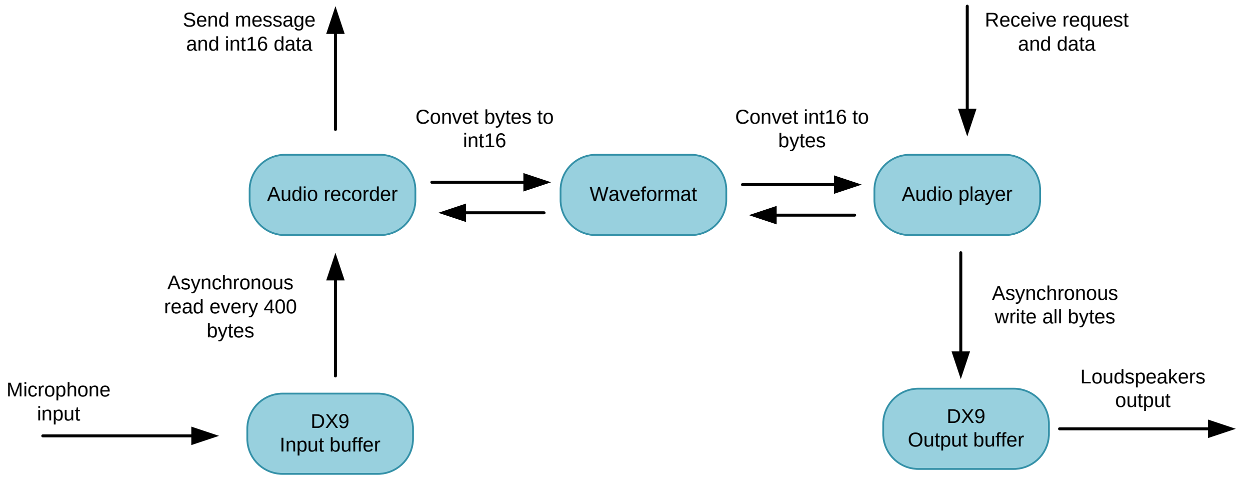


Figure 6 . DX9 audio

From Figure 6 we can see that the audio recorder converts the recorded input from byte to int16. This is due to easier data processing later on. But when the recorded data is to be played, then we convert it back.

It would be obvious to just store the recorded bytes, but as we are able to read data from files and we also want to play the enhanced recording we need to make this convertion.

The wave format component contains a definition of the wave format we use (16kHz 16 bit mono). It also contains the related conversion functions.

The DX9 input and output buffers are a part of the DirectX library for audio recording and playback. Thus this program needs DirectX to be present but we include it embedded in the program so no problem is related to this.

Authors:
Anders Havnsø Rasmussen
Dan Bakmand-Mikalski

Contact:
Mail bakmand@gmail.com
Phone +45 28761007

3.3 DSP

The function of this component is primarily to perform speech enhancement, noise removal and voice activity detection. This function performs directly on the memory items in such a way that it reads from the memory item containing the original recorded input and write to the memory items which contains the enhanced input and detected speech as seen in Figure 7.

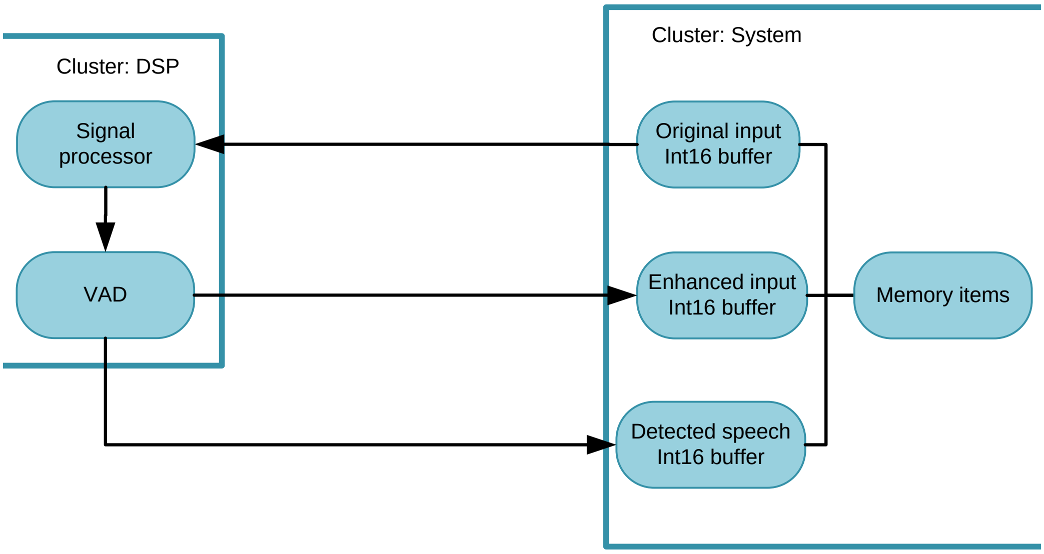

Cluster: DSP
Signal processor
VAD
Cluster: System
Original input Int16 buffer
Enhanced input Int16 buffer
Detected speech Int16 buffer
Memory items

Figure 7 - DSP and memory item relations

Actually the detected speech Int16 buffer only contains a set of indexes into the Enhanced input. Also notice how the enhanced buffer will typically be a little shorter than the original buffer. This is due to the asynchronous update as the DSP is running on a separate thread. The delay is not noticeable for the user so we consider it real-time. This is illustrated in Figure 8 where the red line is the enhanced input and the blue line is the original input. Figure 9 shows the main menu for tweaking the VAD. More options exists though.

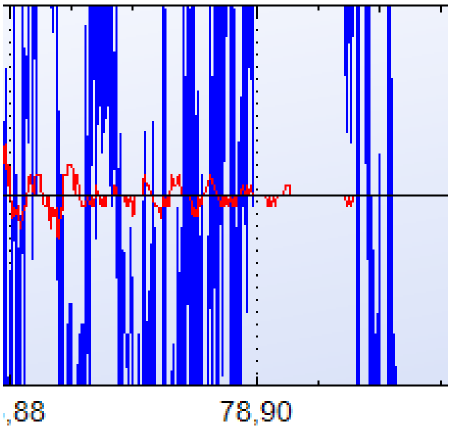
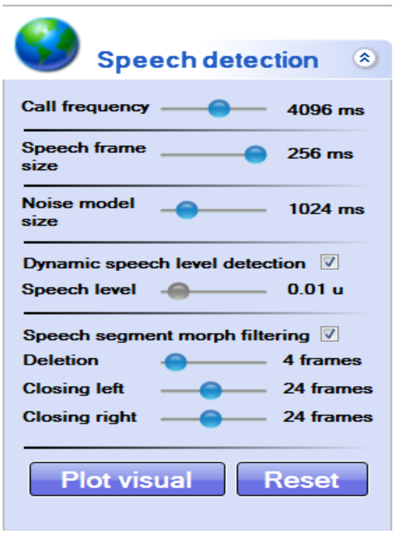


Figure 8 - DSP delay


Speech detection

Call frequency 4096 ms
Speech frame size 256 ms
Noise model size 1024 ms
Dynamic speech level detection ☑
Speech level 0.01 u
Speech segment morph filtering ☑
Deletion 4 frames
Closing left 24 frames
Closing right 24 frames

Figure 9 - DSP options from the program

3.4 Classification

Classification works solely on the detected and enhanced speech. There is an option in the program to use either a neural network or a Gaussian mixture model for classification.

As seen in Figure 10 a feature extraction component extracts the selected features. They are typically MFCC and/or d(d)MFCC. The results are stored in a buffer of data type doube (for precision issues). This buffer is far smaller than the input signal buffers, as we don't classify each sample, but each speech segment in steps of 100-400 seconds depending on user selection.

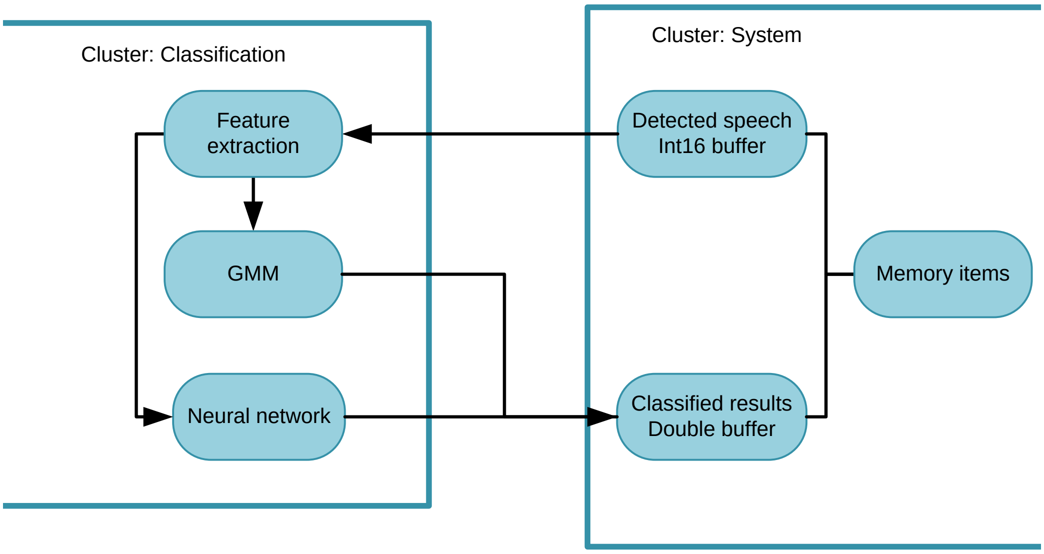
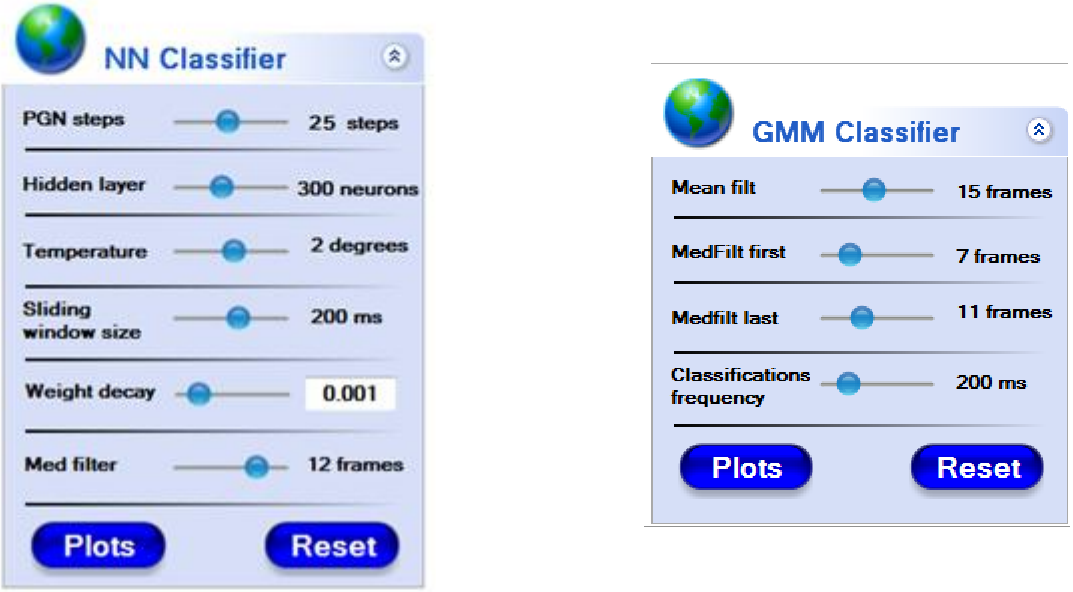


Figure 10 - Classification interaction with memory items

The primary properties that the user can adjust are seen in Figure 11.



NN Classifier

PGN steps 25 steps
Hidden layer 300 neurons
Temperature 2 degrees
Sliding window size 200 ms
Weight decay 0.001
Med filter 12 frames

GMM Classifier

Mean filt 15 frames
MedFilt first 7 frames
Medfilt last 11 frames
Classifications frequency 200 ms

Figure 11 - Classification tweaks

3.5 System

The system clusters central component is the thread manager which is responsible for maintaining the access rights to the memory items. This is important because they are shared by multiple threads and thus are critical regions. We use the .NET Monitor class to control access which is a semaphore.

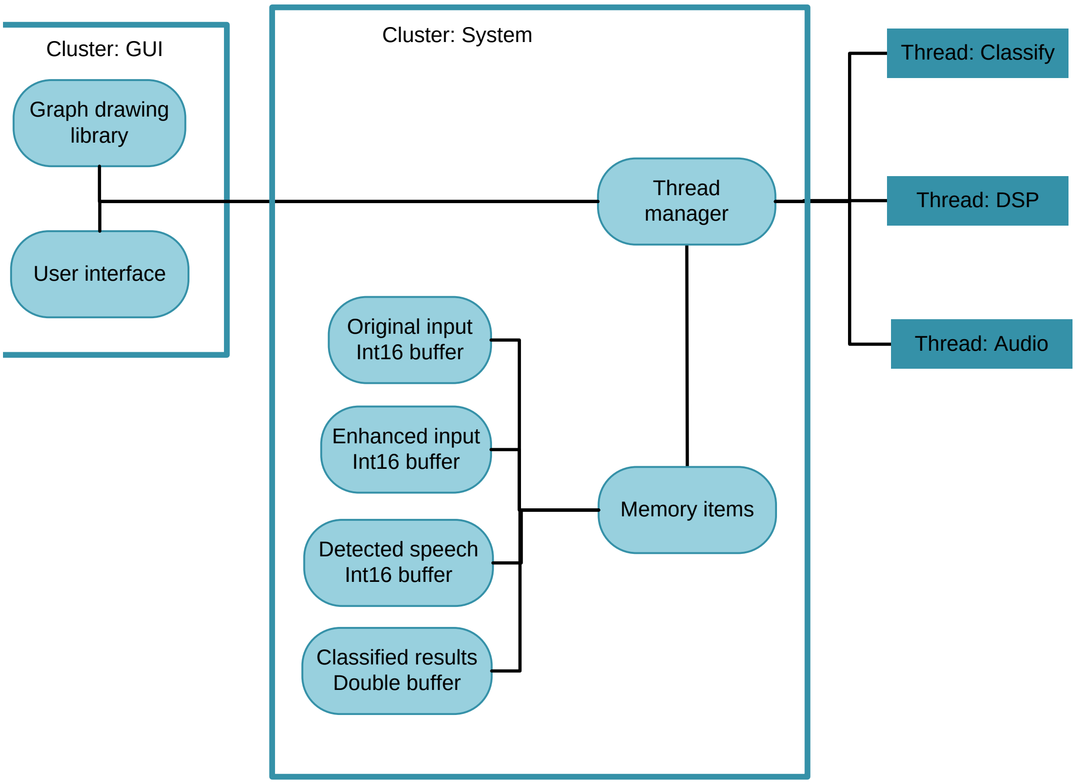


Figure 12 - Memory and thread management

From Figure 12 we can see that the thread manager is connected to the 3 worker threads in the program. The tread manager itself runs on the applications main thread.

The thread manager is also responsible for delegating requests and messages between the 3 threads and the user interface. This way the user can e.g. call for a new classification using another GMM or NN. The user can also pause processing or change the parameters that the 3 worker threads use. On the other hand, the 3 threads send messages and data back to the thread manager (again via a critical region of temporary objects). The thread manager then stores them in the correct memory items and signals the user interface and graph library to update.

This is performed at real-time with only a few milliseconds of delay.

4) Memory items issue

The most significant memory items are as showed in Figure 13. The reason why they are especially important is that they are continuously growing when input is recorded/loaded and processed.

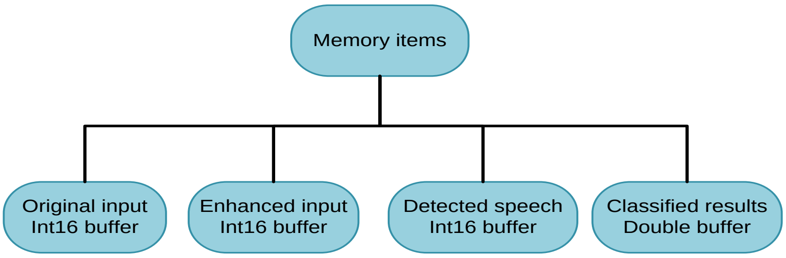

Memory items
Original input Int16 buffer
Enhanced input Int16 buffer
Detected speech Int16 buffer
Classified results Double buffer

Figure 13 - Memory items

Initially we experienced “out of memory” problems even on a laptop with 2 gigabyte ram.

This was a side effect of using the XP method. As we implemented the components in a step-by-step procedure as seen in the process report, we didn’t handle the redundancy issues that occurred. Each component had its own clones of the buffers. This is the downside of not modeling the system thoroughly before implementation. As a result, we ended up doing a lot of work on updating all components, in the debug phases of XP.

The result is that no more than one copy of each of the buffers exist at any time. The memory issues become more obvious if we do some calculations as to how much memory the buffers consume on up to 2 hour of recording. This is seen in Figure 14.

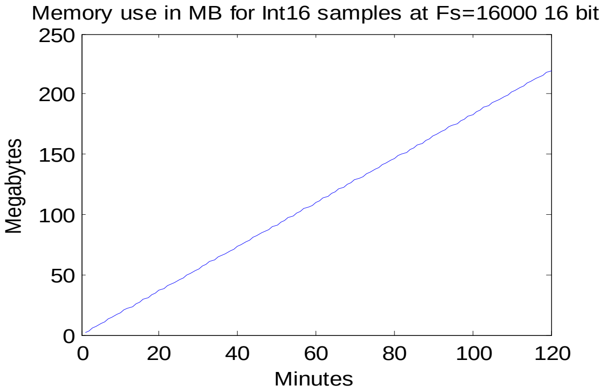


Figure 14 - Memory use over time

As we use at least 2 buffers of Int 16, and possible more, the memory use is more than 200 Mb for 1 hour of recording. On top of this, comes the locally allocated memory that the functions use during computation. This is still within reasonable range, but that wasn’t true before we redesigned the memory blocks. At that time we used more than 5 local clones of the buffers, which cause “out of memory” after approximately ½ hour.

Appendix A

Contract for “Speaker Identification” DTU 2007

This contract is a binding agreement between the parties of the master project: ”Speaker identification”, DTU 2007.

The agreement is between Dan Bakmand-Mikalski & Anders Havnsø Rasmussen, both civil engineering students at the IMM institute at Denmarks Technical University (DTU) in Kgs. Lyngby.

Project superviser is Niels-Ole Christensen

___________________________	___________________________
ANDERS HAVNSØ RASMUSSEN	DAN BAKMAND-MIKALSKI

Contents

1) Management and work environment

1.1 Management

All parties of the project team has equal rights. Noone can make strategic descisions without the approval of a majority of team members. In teams of even count. The member posting a suggestion is not voting. The responsibility is thus divided equally between the project team members.

1.2 Work environment

It is *within reason* allowed to engage activities of personal character (e.g. reading mails, browsing homepages…). Social activities are promoted base don the philosophy of happiness encourages creativity and repells stress.

It is vital that problems and internal conflicts is surfaced at an early state. Positive communications should be used as a way of mutual inspiration and encouragement. Mutual respect for each team meber and their work must be maintained.

It is compulsory that each team member can claim an explanation of project related subjects he or she doesn't understand.

The most vital goal of this project is to expand the knowledge of all participants.

1.3 Level of ambitions

The most important ambition of this master thesis is that all team members pass the exam.

It is assumed more important that all participants gain knowledge of all main subjects of this master thesis than it is to aquire a high grade.

An example of this is that a person with little or no insight into a particular problem should be involved in this problem thus gaining knowledge. This is opposed to only involving team members than has the most knowledge of the subject.

1.4 Participation

Participation is mandatory. If a team member claims the need to continue working later than normal working hours, this has higher priority than other members who want to go home. Considerations has to be made for those who has planned family related or social activities after normal working hours.

Special activities not related to this project which need to be carried out within working hours (e.g. doctors appointments) must have been presented no later than the previous work day. The implicated team member is obliged to handle complactions which occurs as a consequence to his or hers absence.

The room for private activites during working hour is expected be larger at the earlier states of the project and tighten towards the final deadline.

Sickness and other unforseen situations must be respected. If the participation of a particular team member drops to a level which has significant effect on the execution of the project or the participation gives rise to doubts about the seriousness of a particular team member this should be treated as a conflict (look at section: “Conflict handling”).

2) Workflow

2.1 Decision base for solving work task

It is accepted that every team member can't participate in all activities. It is the responsibility for each team member to engage the rest of the team in major decisions related to all tasks. It is however the responsibility of each team member to engage him or her in fields which the team member is uncertain about.

It is both expected and the responsibility of each team member to offer constructive criticism of solved tasks.

2.2 Information

After solving a task it is the duty of the task responsible to revise the task solution with the rest of the team. This allows the other team members to suggest improvements and correct errors.

The team should be informed of major events related to individual areas of responsibility.

It is particularly important to inform or request help if a team member has doubts as how to solve a particular task. The same is true if a team member is unsure of how a particular task relates to the goal of the project etc.

3) Scheduling

3.1 Progress control

It is the team member who is responsible for the scheduling and milestones that is also responsible for verifying the progress of the project. If this team member experience any conflict in the schedule or anticipate any based on feedback from the rest of the team a progress meeting should take place. The meeting should be focus on how to handle the delay and how to avoid future delays based on this new experience.

3.2 Progress meetings

A progress meeting is planned at least every 2 weeks and no less than 8 times during the entire project timespan.

At the progress meeting all team members must prepare a short informal speech containing new key aspects of their field of responsibilities.

3.3 Evaluation of team engagement

At progress meetings the team member responsible for scheduling and milestones must inform on the progress of the project.

This team member must ask questions to the entire project team in such a way to estimate if the current tasks are within schedule or should be considered a risk. This is expected to prevent unrecoverable problems related to scheduling etc.

4) Conflicts

4.1 Options for changing this contract

If a majortity of the team members experience a section of this contract as causing problems related to finishing the project in a satisfactory manner, it is possible to change or add to this contract. This however is only possible with a 3-1 majority (2-1 for project groups with 3 members. All must agree in project teams with less than 3 members).

If a change or addition cannot be accepted by a team member this team member has an option of veto. If the veto cannot be accepted by the rest of the project team the person stating the veto must leave the project team.

4.2 Solving conflicts

All team members must offer their view of the particular conlict. It is assumed that a conflict is best handled at ealiest opportunity.

It is required that each team member must be 100% open and honest. All point of view must be accepted and respected.

A conflict based on breach of this contract must be attempted solved as quickly as possible within the limits of this contract.

If this nature of a conflict is such that the conflict can be stated but not solved a reevaluation of the exlusion of related team members must be discussed.

4.3 Exclusion

Exlusion is based on the assumption that the majority is not necessarily corect. As a consequence of this assumption, considerations must be payed to both the remaining and excluded member(s) opportunity to complete their education.

Is this not possible the projet team must be dissolved and new teams build.

All members of the project team before the exclusion has equal right to all materials and software developed until the exclusion is in effect.

Appendix B

Milestones

ID	Milestone Name	Milestone Description	Milestone Type
1	**Project & development strategies**	The foundation for the project must be complete. The report layouts are ready. Work on speaker identification can begin now.	text
2	**Process analysis**	The document of the process report are finalized	text
3	**Front-end signal processing analysis**	Issues related to basic input signal handling including noise/speech enhancement are examined and solutions are now created.	text
4	**Front-end signal processing architectural components**	Matlab versions of the software are prepared for conversion to C#. Interfaces for the wrapper functions are defined.	software
5	**Voice activity detection analysis**	Issues related to detecting speech from the input signal are now examined and solutions created.	text
6	**Voice activity detection architectural components**	Matlab versions of the software are prepared for conversion to C#. Interfaces for the wrapper functions are defined.	software
7	**Feature analysis analysis**	Analysis of what and how features are to be extracted are now complete.	text
8	**Feature analysis architectural components**	Matlab versions of the software are prepared for conversion to C#. Interfaces for the wrapper functions are defined.	software
9	**Classification analysis**	Models for classification has now been suggested and adapted to fit the feature sets. Parameters influence on performance has been evaluated now.	text
10	**Classification architectural components**	Matlab versions of the software are prepared for conversion to C#. Interfaces for the wrapper functions are defined.	software
11	**Win32 Speaker identification application**	GUI, drawing libraries and DirectX Audio has been implemented in C#. All converted software components are now wrapped into a C# based Win32 application. The application has been debugged at prototype level.	software
12	**Implementation report**	A degraded documentation of the Win32 application is now complete.	text
13	**All documentation**	A final correction reading and software check has now been performed prior to printout and project closure.	text

	Milestone Type	Complete	2007								
			feb	mar	apr	maj	jun	jul	aug	sep	okt
st be eady. Work n now.	text	01-03-2007		☆							
rt are	text	15-03-2007		☆							
handling ent are reated.	text	01-05-2007				☆					
e prepared r the	software	15-05-2007				☆					
from the l solutions	text	18-06-2007					☆				
e prepared r the	software	02-07-2007						☆			
are to be	text	16-07-2007						☆			
e prepared r the	software	31-07-2007						☆			
been eature sets. nce has	text	14-08-2007							☆		
e prepared r the	software	17-09-2007								☆	
Audio has ts are now pplication. d at	software	15-10-2007									☆
Win32	text	22-10-2007									☆
ware check printout	text	29-10-2007									☆

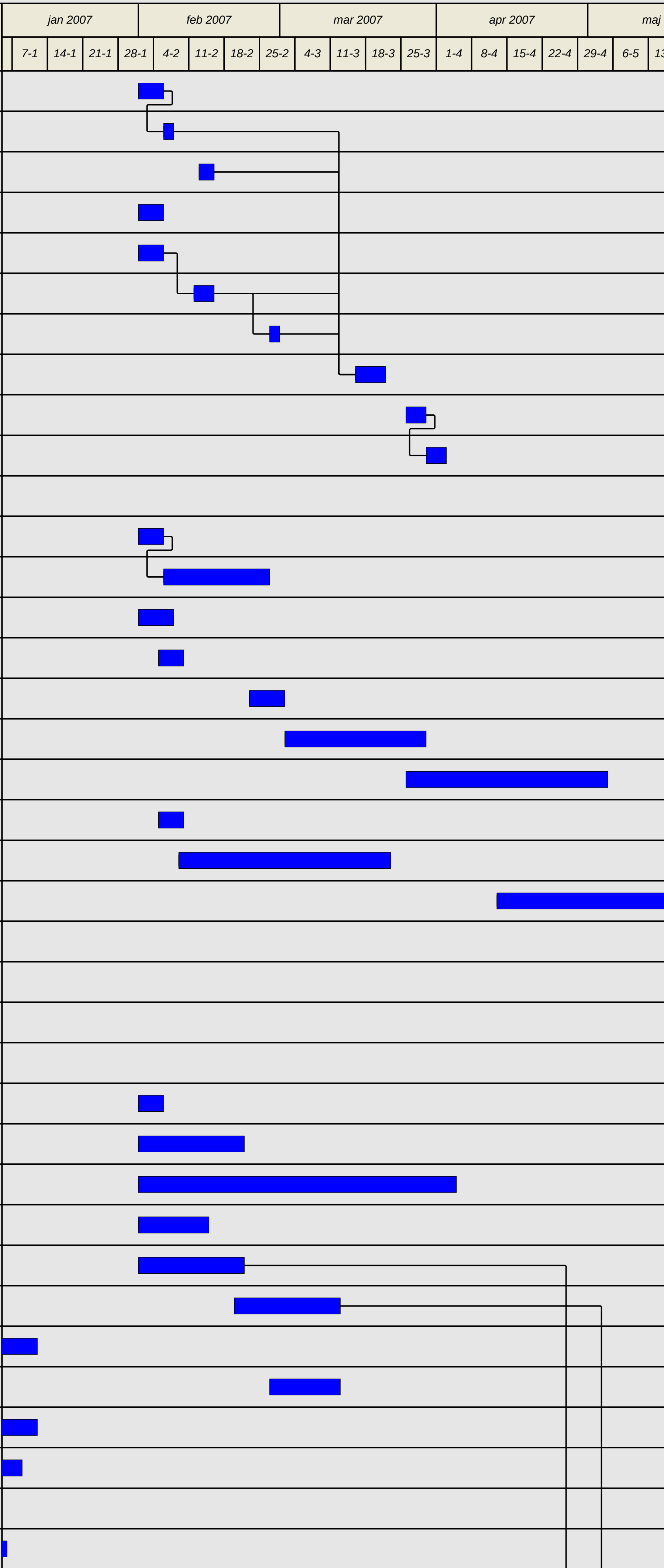
Appendix C

Schedule

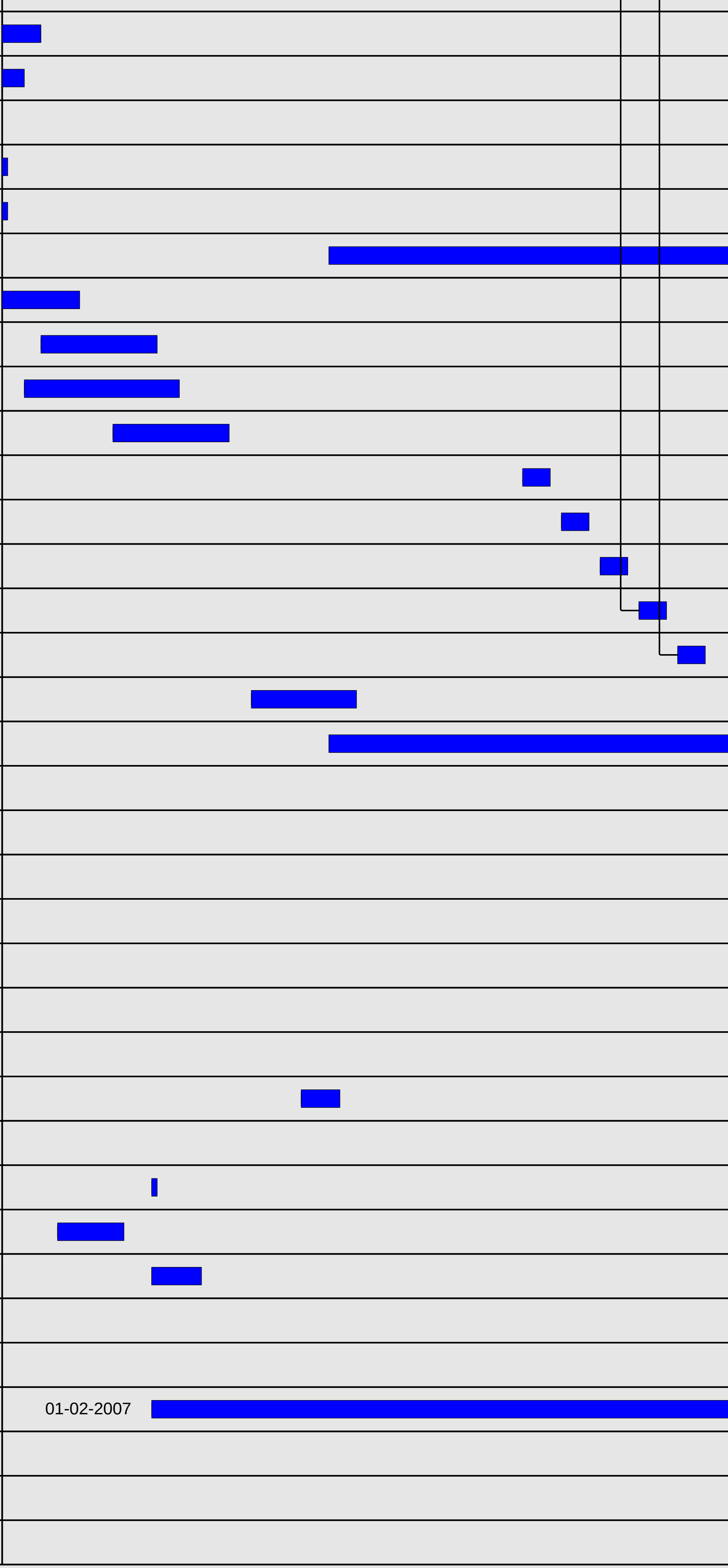
ID	FIELD	Task Name	Task Type	Responsible Autor	Start
1	Process report	Problem statement	text	DBM / AHR	01-02-2007
2		Goals	text	DBM / AHR	06-02-2007
3		Participants strengths/weeknesses	text	DBM / AHR	13-02-2007
4		Contract	text	DBM / AHR	01-02-2007
5		Risc analysis	text	DBM / AHR	01-02-2007
6		Development strategies	text	DBM	12-02-2007
7		Milestones	text	DBM / AHR	27-02-2007
8		Scheduling	text	DBM / AHR	16-03-2007
9		Various documentation	text	DBM / AHR	26-03-2007
10		Correction reading	text	DBM / AHR	30-03-2007
11					
12	Singal & feature analysis report	Basic audio related issues	text	DBM	01-02-2007
13		Front-end processing analysis	text	DBM	06-02-2007
14		Matlab implementation : DC removal	software	DBM	01-02-2007
15		Matlab implementation : Speech enhancement filtering	software	DBM	05-02-2007
16		Matlab implementation : Noise removal	software	DBM	23-02-2007
17		Voice Activity Detection analysis	text	DBM	02-03-2007
18		Matlab implementation : Voice Activity Detection	software	DBM	26-03-2007
19		Human speech production	text	DBM / AHR	05-02-2007
20		Feature extraction analysis	text	AHR	09-02-2007
21		Matlab implementation : Feature extraction	software	AHR	13-04-2007
22		DSP results	text	AHR	25-05-2007
23		Various documentation	text	DBM / AHR	30-05-2007
24		Correction reading	text	DBM / AHR	07-06-2007
25					
26	Classification report	Choose suitable models	text	DBM / AHR	01-02-2007
27		Matlab implementation : Agile GMM	software	AHR	01-02-2007
28		Matlab implementation : Agile NN	software	DBM	01-02-2007
29		Matlab implementation : Parameter optimization component	software	AHR	01-02-2007
30		GMM analysis	text	AHR	01-02-2007
31		NN analysis	text	DBM	20-02-2007
32		Solution design	text	DBM / AHR	05-01-2007
33		Classification results	text	DBM / AHR	27-02-2007
34		Various documentation	text	DBM / AHR	05-01-2007
35		Correction reading	text	DBM / AHR	05-01-2007
36					
37		Structural design	text	DBM / AHR	05-01-2007

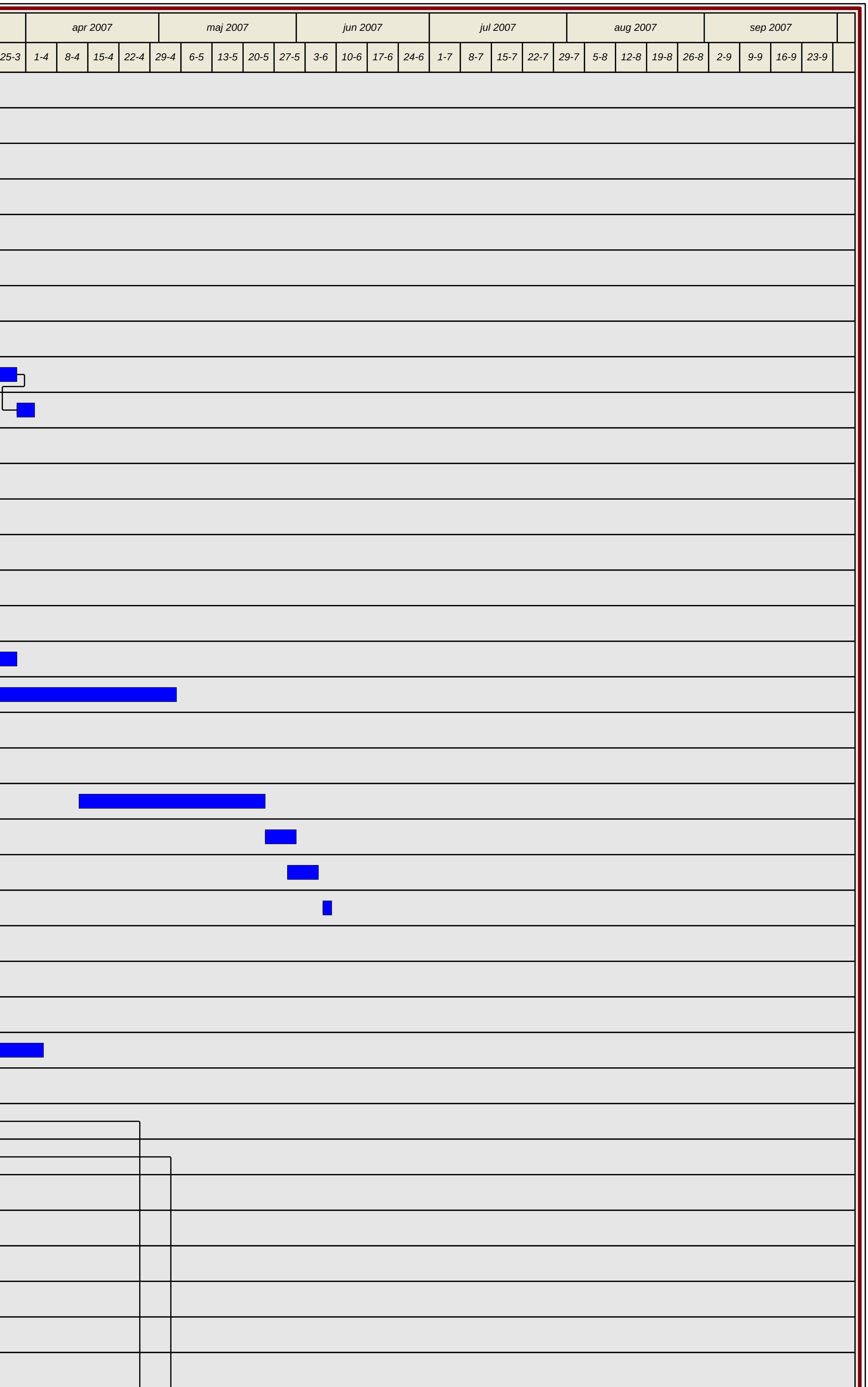
34		Various documentation	text	DBM / AHR	05-01-2007
35		Correction reading	text	DBM / AHR	05-01-2007
36					
37	Implementation report	Structural design	text	DBM / AHR	05-01-2007
38		Parallel programming design	text	DBM	05-01-2007
39		Prepare architectural spikes	software	DBM / AHR	05-03-2007
40		Graphical interface design	text	DBM	05-01-2007
41		DirectX implementation : Audio I/O components	software		12-01-2007
42		C# implementation : GUI controls	software	DBM	09-01-2007
43		C# implementation : Graphical drawing components	software	DBM	25-01-2007
44		C# implementation : Signal processing Matlab wrappers	software	DBM	09-04-2007
45		C# implementation : Feature extraction Matlab wrappers	software	AHR	16-04-2007
46		C# implementation : Voice activity detection Matlab wrappers	software	DBM	23-04-2007
47		C# implementation : GMM Matlab wrappers	software	AHR	30-04-2007
48		C# implementation : Neural Networks Matlab wrappers	software	DBM	07-05-2007
49		C# implementation : Various coding	software	DBM / AHR	19-02-2007
50		XP phase : Release plannning design	text	DBM / AHR	05-03-2007
51		XP phase : Components coding	software	DBM / AHR	21-05-2007
52		XP phase : Acceptance tests	text	DBM / AHR	21-05-2007
53		XP phase : Bug solving	software	DBM / AHR	21-05-2007
54		XP phase : Iterations of small releases	software	DBM / AHR	03-07-2007
55		Various documentation	text	DBM / AHR	23-07-2007
56		Correction reading	text	DBM / AHR	10-09-2007
57					
58	Others	Project strategy	text	DBM / AHR	28-02-2007
59		Major challenges	text	DBM / AHR	21-09-2007
60		Limitations	text	DBM / AHR	01-02-2007
61		Collect/test development tools	software	DBM / AHR	15-01-2007
62		Documentation layout	text	DBM / AHR	01-02-2007
63		Buffer for events related to risks	Mixed unknown	DBM / AHR	01-10-2007
64		Planed hollidays	fun	DBM / AHR	06-08-2007
65		Parallel course on DTU	Not so fun	DBM / AHR	01-02-2007
66		Various documentation	text	DBM / AHR	13-09-2007
67		Correction reading	text	DBM / AHR	24-09-2007
68		Finalizing all documentation	text	DBM / AHR	27-09-2007

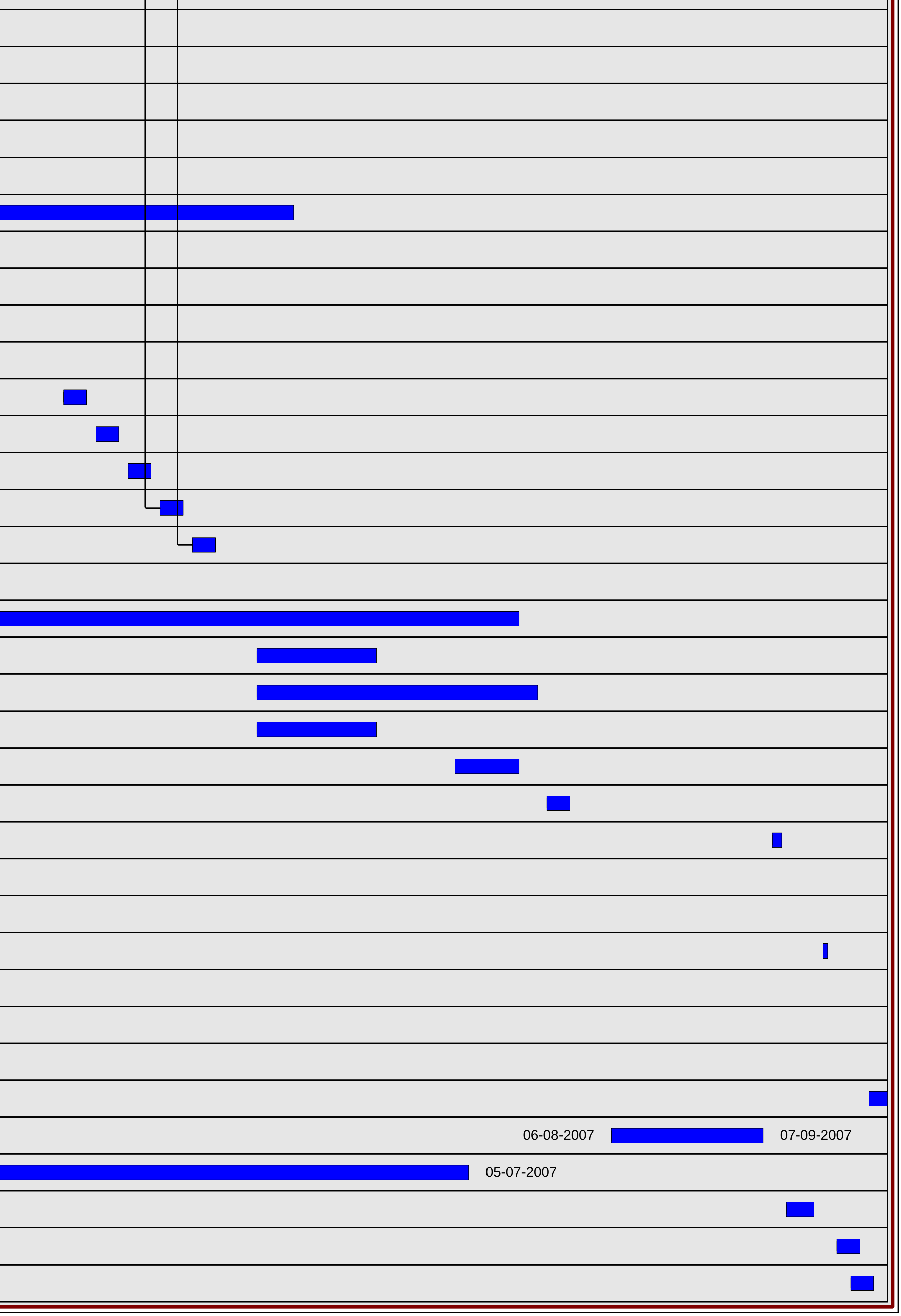
onsible Autor	Start	Finish	Duration
AHR	01-02-2007	05-02-2007	3d
AHR	06-02-2007	07-02-2007	2d
AHR	13-02-2007	15-02-2007	3d
AHR	01-02-2007	05-02-2007	3d
AHR	01-02-2007	05-02-2007	3d
	12-02-2007	15-02-2007	4d
AHR	27-02-2007	28-02-2007	2d
AHR	16-03-2007	21-03-2007	4d
AHR	26-03-2007	29-03-2007	4d
AHR	30-03-2007	02-04-2007	2d
	01-02-2007	05-02-2007	3d
	06-02-2007	26-02-2007	15d
	01-02-2007	07-02-2007	5d
	05-02-2007	09-02-2007	5d
	23-02-2007	01-03-2007	5d
	02-03-2007	29-03-2007	20d
	26-03-2007	04-05-2007	30d
AHR	05-02-2007	09-02-2007	5d
	09-02-2007	22-03-2007	30d
	13-04-2007	24-05-2007	30d
	25-05-2007	31-05-2007	5d
AHR	30-05-2007	05-06-2007	5d
AHR	07-06-2007	08-06-2007	2d
AHR	01-02-2007	05-02-2007	3d
	01-02-2007	21-02-2007	15d
	01-02-2007	04-04-2007	45d
	01-02-2007	14-02-2007	10d
	01-02-2007	21-02-2007	15d
	20-02-2007	12-03-2007	15d
AHR	05-01-2007	11-01-2007	5d
AHR	27-02-2007	12-03-2007	10d
AHR	05-01-2007	11-01-2007	5d
AHR	05-01-2007	08-01-2007	2d
AHR	05-01-2007	05-01-2007	1d



AHR	05-01-2007	11-01-2007	5d
AHR	05-01-2007	08-01-2007	2d
AHR	05-01-2007	05-01-2007	1d
	05-01-2007	05-01-2007	1d
AHR	05-03-2007	28-05-2007	61d
	05-01-2007	18-01-2007	10d
	12-01-2007	01-02-2007	15d
	09-01-2007	05-02-2007	20d
	25-01-2007	14-02-2007	15d
	09-04-2007	13-04-2007	5d
	16-04-2007	20-04-2007	5d
	23-04-2007	27-04-2007	5d
	30-04-2007	04-05-2007	5d
	07-05-2007	11-05-2007	5d
AHR	19-02-2007	09-03-2007	15d
AHR	05-03-2007	16-07-2007	96d
AHR	21-05-2007	15-06-2007	20d
AHR	21-05-2007	20-07-2007	45d
AHR	21-05-2007	15-06-2007	20d
AHR	03-07-2007	16-07-2007	10d
AHR	23-07-2007	27-07-2007	5d
AHR	10-09-2007	11-09-2007	2d
AHR	28-02-2007	06-03-2007	5d
AHR	21-09-2007	21-09-2007	1d
AHR	01-02-2007	01-02-2007	1d
AHR	15-01-2007	26-01-2007	10d
AHR	01-02-2007	09-02-2007	7d
AHR	01-10-2007	19-10-2007	15d
AHR	06-08-2007	07-09-2007	25d
AHR	01-02-2007	05-07-2007	111d
AHR	13-09-2007	18-09-2007	4d
AHR	24-09-2007	28-09-2007	5d
AHR	27-09-2007	01-10-2007	3d




apr 2007
maj 2007
jun 2007
jul 2007
aug 2007
sep 2007
25-3
1-4
8-4
15-4
22-4
29-4
6-5
13-5
20-5
27-5
3-6
10-6
17-6
24-6
1-7
8-7
15-7
22-7
29-7
5-8
12-8
19-8
26-8
2-9
9-9
16-9
23-9


06-08-2007
07-09-2007
05-07-2007

Appendix D1

DC removal by running average filtering

1.1.1 DC-component removal using filtering

The running average filter (or mean filter as it is sometimes called) is one of the simplest filters. All coefficients are equal and normalized.

1.1.1.1 Filter layout

The filter layout in Figure 1 illustrates the 1-dimensional discrete filter.

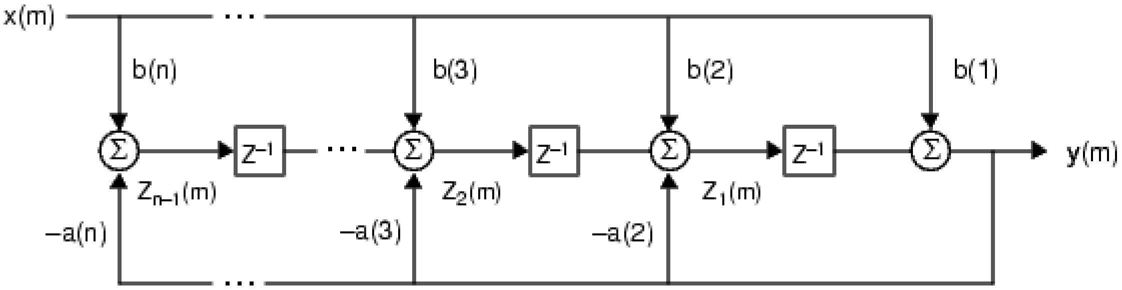


Figure 1 - Running average filter layout

This allows for the following filter to work:

$$H(Z) = \frac{1}{k}[1 \ldots k]\, Z^{-1} \qquad Where\ k\ =\ filter\ size\ (order)$$

Figure 2 shows the filter with filter order *k=20*

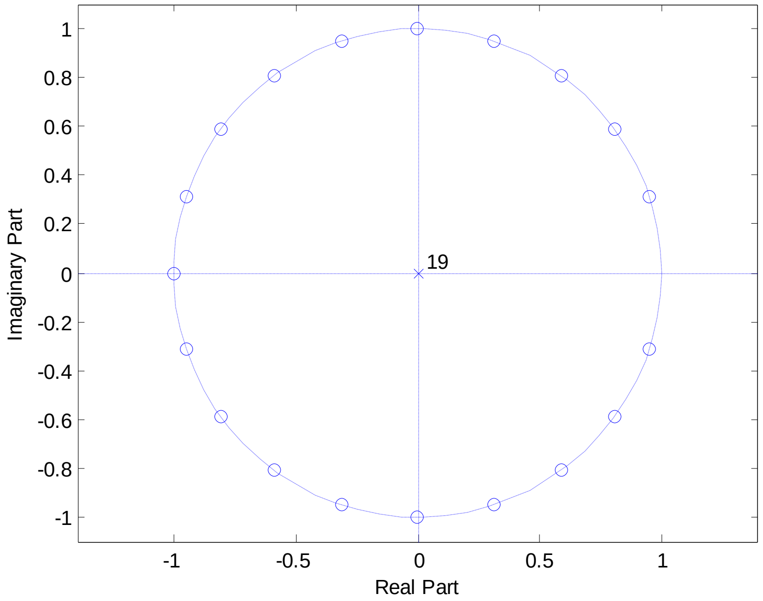


Figure 2 - Zero-pole of 20. Order running average filter

1.1.1.2 Filter response

The filter response is illustrated in Figure 3 using 20 equal coefficients of value 0.05

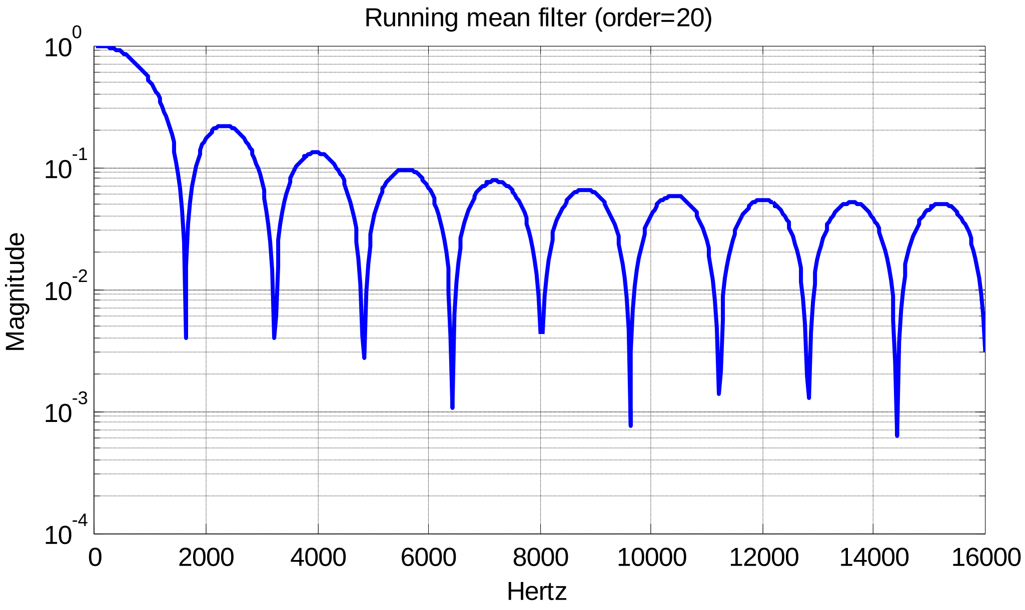


Figure 3 - Running average filter response

1.1.1.3 Influence of the filter order

The precision of the mean estimate based on 10 signals of length 11-15 seconds at a sampling frequency of 16 kHz is shown in Figure 4.

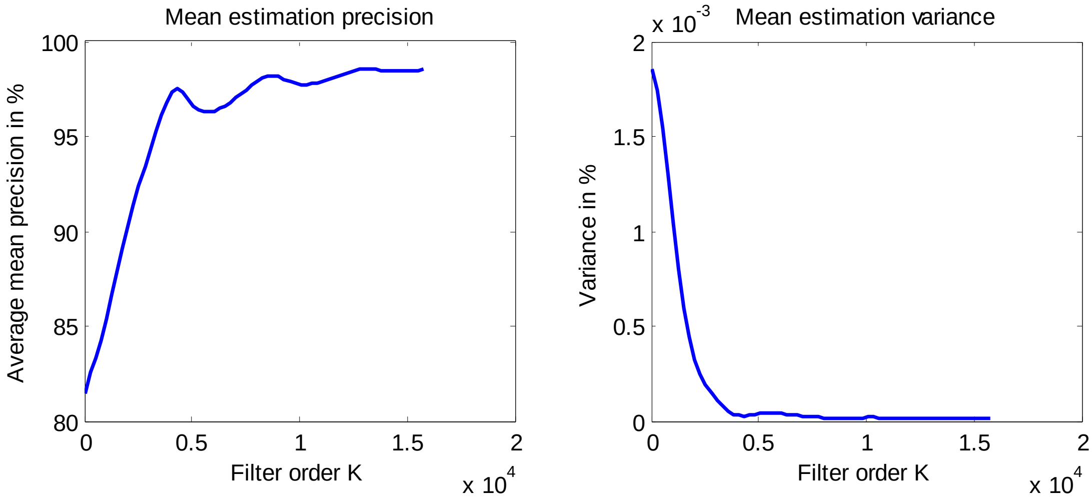


Figure 4 - Relation between filter order and estimate precision

From the Figure 4, it can be seen that the filter order greatly influences the precision of the mean estimate. This can be evaluated because we use known signals in this example.

From Figure 4 it can also be seen that the estimation variance decreases rapidly when increasing the filter order up to 15000 filter coefficients. This means that if 15000 samples are used for estimation the result is gradually beginning to stabilize.

As an authors comment we may add, that it is not until the filter order is larger than 32000 (2 seconds) that the precision consistently rises to above 98 % which is an acceptable precision for the context of use (RMS computation for speech activity detection).

When the variance between the estimates and the true mean decreases, it is also obvious that the running average converges towards the true mean.

The downside to this approach however is that the estimate is not reliable until the filter buffer is filled. That means when it goes from transient state to steady state.

1.1.1.4 Transient and steady state of running average estimates

Figure 5 shows the mean estimate of a streaming input signal using two different filter orders. It is zoomed in on the beginning of the estimate to visualize the effect of filter order and its transient state.

It can be seen from the figure that a filter order of 20 decreases the transient state but it also makes the mean estimate more subtle to changes in the streamed input. The opposite is true for a filter order of 100.

Also note that the variance is smaller and therefore the precision is greater for the larger filter order.

As seen in Figure 5, the mean estimate is still not very precise when using filter order = 100. To obtain an estimate with an error less than 2. % it requires a filter order of at least 32.000 (2 seconds) as mentioned earlier.

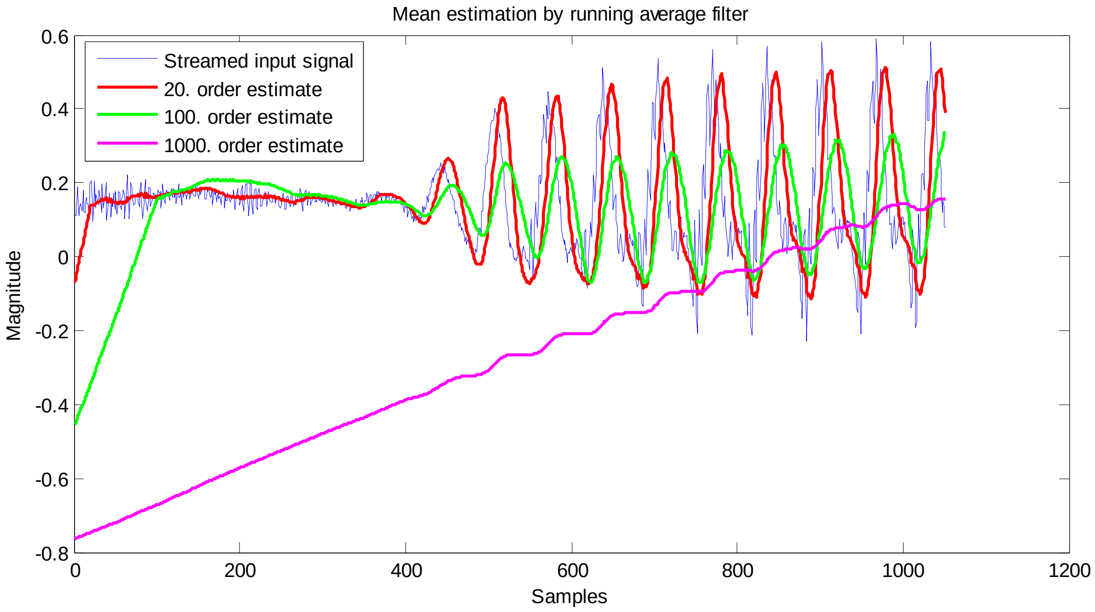


Figure 5 - Visualization of filter order influence

1.1.1.5 Example of streamed signal with mean removed by running average filtering

When applying the running average filter of order 100 to an input stream, the mean can be removed based on the running average estimate. This is illustrated in Figure 6.

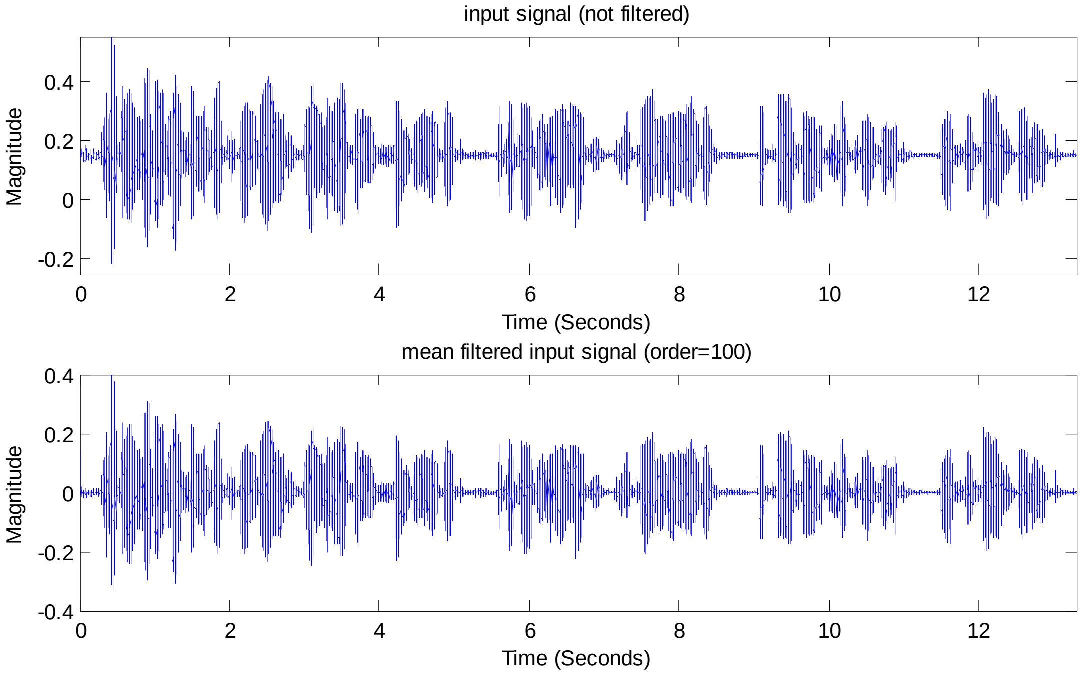


Figure 6 - DC component removed from streaming input

Appendix D2

DC removal extended results

1.1.1 Results of DC removal on streaming signal

1.1.1.1 Choice of method for DC component removal

If we compare the memory and computational cost difference using the configuration in Figure 1, the benefits of cache based estimation are clear.

	Cache based estimation	*Running average filter*
Samples cache size (s)	320 (20 milliseconds)	
Mean cache size (h)	1000 (2 seconds in 20 ms intervals)	
Filter size (n)		32000 (2 seconds of samples)
Long term memory	2 seconds	2 seconds
Computational cost pr. 20 ms	O(s)	O(n)
Total memory use	4640 bytes (320 uint16 + 1000 uint32)	64000 bytes (32000 uint16)

Figure 1 - Cost table of running average filter vs. cache based estimates

Although the performance increase is dependent on the 2 cache sizes it nevertheless reduces memory consumption to less than 10% and reduce computational cost to approximately 1% in general.

It is further more possible to adjust cache sizes to fit the need for accuracy and speed tradeoffs.

Based on these results, we have chosen to use cache based estimation as the real-time perspective is one of the primary goals.

1.1.1.2 Accuracy of chosen method for DC component removal

To evaluate the accuracy of cache based estimation, the DC component is detected dynamically from a streamed input signal as in Figure 2 using the setup shown in Figure 1.

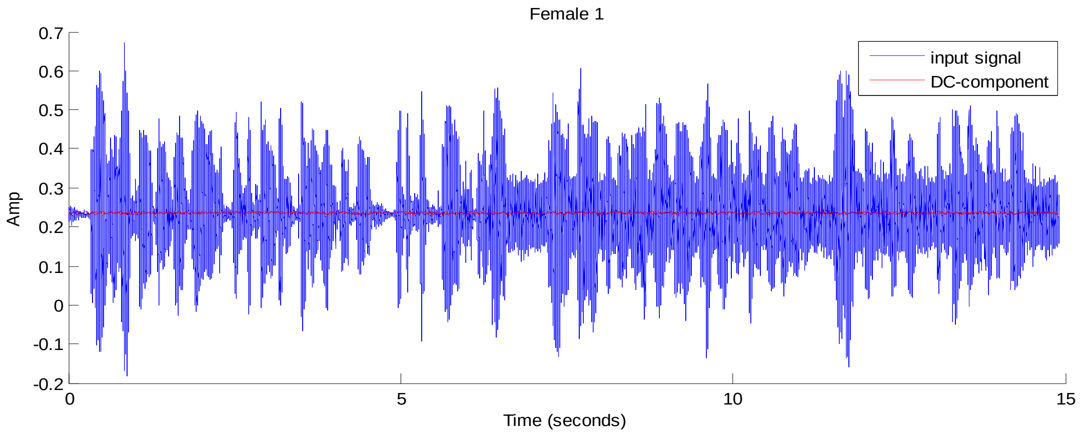


Figure 2 - DC component estimate using cache based estimation

After removing the DC component the alteration to the signal is minor. This alteration (look at Figure 3) is an effect of the dynamic DC removal as the estimate is constantly updated for increased precision using a growing mean cache. If the signal was not streamed or it could be assumed that the DC was known and constant, this effect would of cause not occur.

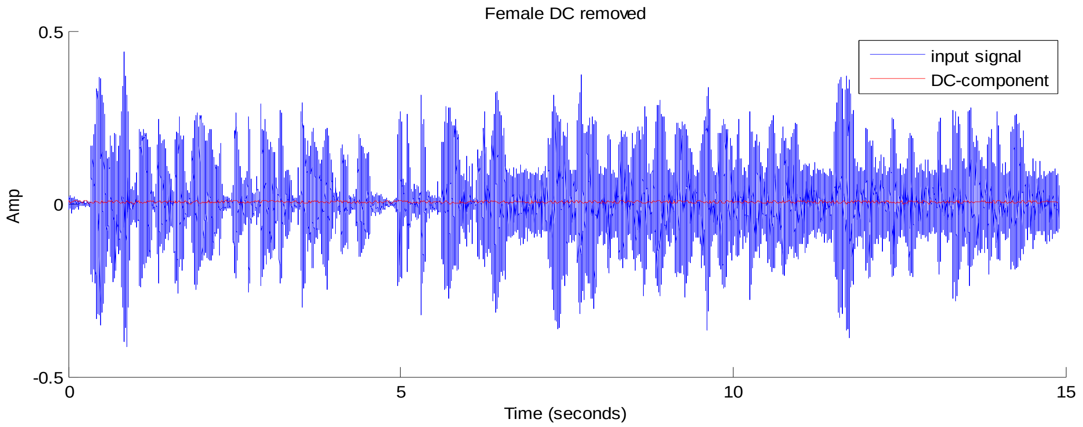


Figure 3 - DC component removed based on updated estimates

In a perfect information world, as this test case is, we have the opportunity of computing the "true" DC component from the entire signal using Lemma 1.

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \approx 0.23$$ Lemma 1

Using the true mean value we can now remove it from input signal x which gives us $\hat{x}$ by Lemma 2.

$$\hat{x}_i = x_i - \mu \quad for\ i = 1 \dots n$$ Lemma 2

By comparing results of cache based estimates of DC component from a streamed signal against the “true” DC component we can get an error estimate on the cache based DC removal method. This is done by computing the sum of the absolute difference of the two signals as shown in Lemma 3:

$$error = \frac{1}{n}\sum_{i=1}^{n}|\hat{x}_i, \tilde{x}_i| \approx 0.0037$$

Lemma 3

This error can then be considered as a relative compared to the original DC component which was approximately 0.23.

The approximation accuracy of the DC component is thus computed by Lemma 4:

$$\frac{100}{\mu + error}\mu = \frac{100}{0.2337}0.23 \approx 98.4\ \%$$

Lemma 4

Testing on 5 different males and females, this yields a relative accuracy of the “true” mean value in the range of: *98%* - *99%* based on 2 seconds mean cache size.

This is true when testing on many different types of signals.

Based on observations, the difference is mainly dependent on how fluctuant the magnitude of the test signal is. The more speech or noise the less precise.

1.1.1.3 Influence of mean cache size

The influence of the mean cache size has been estimated by fixing the sample cache size to 320 samples and varying the mean cache size between 10 (200 ms) and 2000 (4 seconds).

The result in Figure 4 is an average over 4 examples from the ELSDSR database.

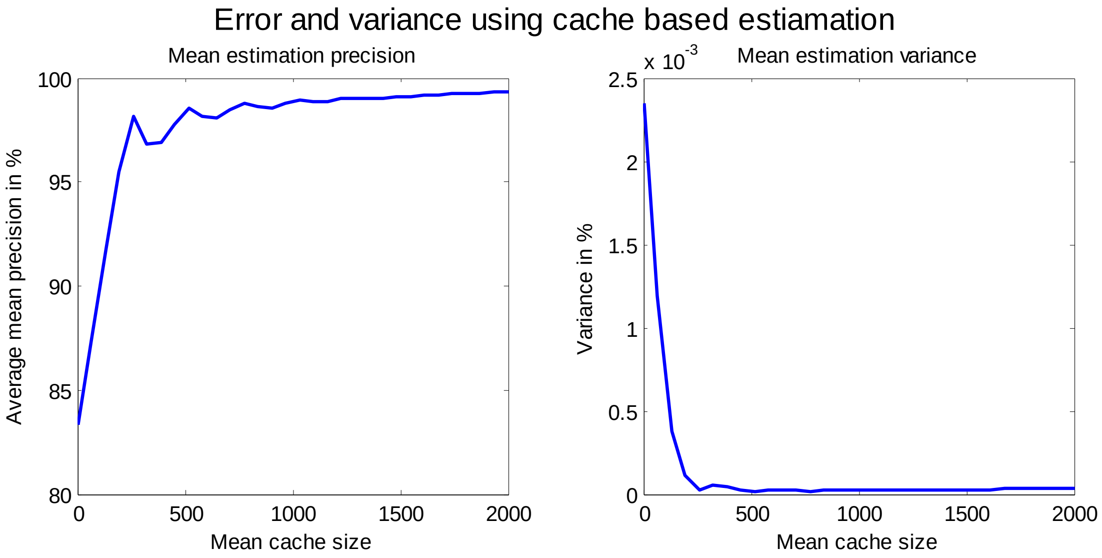


Figure 4 - Relation between mean cache size and precision

Appendix E

Example on how RMS based

voice detection adapts to changning SNR

1.1.1.1 Why changes in noise/speaker magnitude are handled

Consider the following 2 situations which describe a time jump in a stream of input.

- The streaming input signal is divided into frames of 20 milliseconds.
- A speech segment is found
- Each frame is offset by a stepsize of 10 milliseconds from the previous frames' start point.
- Each frame results in a Root mean square value.
- The RMS energy histogram is based on RMS values from 100 frames.

Time now = frame 100 (1 second have elapsed)

This histogram will give a threshold which can be compared to the RMS value of frame 101. This threshold is based on information from frame 1-100.

Assume the threshold equals 0.1

Time now = frame 500 (5 seconds have elapsed)

The streaming input signal has now streamed 500 frames of 20 milliseconds. A threshold based on a histogram of RMS values from the past 100 frames is compared to the RMS value of frame 501.

Assume the threshold now equals 0.2

Comparing the two situations

Suppose a sudden change in the speakers' position relative to the recording device or some noise occurring during the 4 seconds time difference. This could cause the noise level or speaker power to change.

This would be handled correctly since the histogram adapts only to noise and amplitude levels for a predetermined time of the past signal and not the entire signal as illustrated by the assumed values.

Therefore variations are caught and handled by this method.

Appendix F

Brief walkthrough of PCA

PCA is a technique used to reduce multidimensional datasets. The method is very useful in analysing larger dataset as it is possible to reduce data onto e.g. the two or three most important dimensions which can be plotted in Matlab.

To show the effect of PCA analysis, the feature matrix (C) containing coefficients is illustrated below.

$$C = \begin{bmatrix} c_{1,1} & \cdots & c_{1,m} \\ \vdots & \ddots & \vdots \\ c_{n,1} & \cdots & c_{n,m} \end{bmatrix}, \text{ where n equals frames and m equals coefficients.}$$

The first st ep is to calculate the mean of C (see Lemma 1).

$$\tilde{C}_n = \frac{\sum_{m=1}^{M} c_{n,m}}{M}, \textit{ where } N = \textit{number of frames and } M = \textit{number of coefficients} \qquad \textbf{Lemma 1}$$

The next step is calculating the covariance between different dimensions. The covariance between dimension n and m is showed in Lemma 2. The covariance between dimension n and n equals σ^2.

$$\mathrm{c}_{n,m} = \frac{\sum_{i=1}^{N} (c_{i,n} - \tilde{C}_n)(c_{i,m} - \tilde{C}_m)^T}{N-1} \qquad \textbf{Lemma 2}$$

Because the above covariance is measured between 2 dimensions and most of our data are n-dimensional, we use the covariance matrix to represent the different covariance's as showed in Lemma 3.

$$\hat{C} = \begin{pmatrix} \mathrm{cov}_{1,1} & \cdots & \mathrm{cov}_{1,m}) \\ \vdots & \ddots & \vdots \\ \mathrm{cov}_{n,1} & \cdots & \mathrm{cov}_{n,m} \end{pmatrix} \qquad \textbf{Lemma 3}$$

Next step is to find the eigenvalues and eigenvectors. The eigenvalues are used for finding the d (d equals the number of dimensions) most important dimensions. The eigenvectors are used for projection of $C - \tilde{C}$ onto the d most important dimensions. The equation for projection of the data is showed in Lemma 4.

$$\bar{C} = U \cdot C - \tilde{C}, \textit{ where } U \textit{ is an eigenvector matrix} \qquad \textbf{Lemma 4}$$