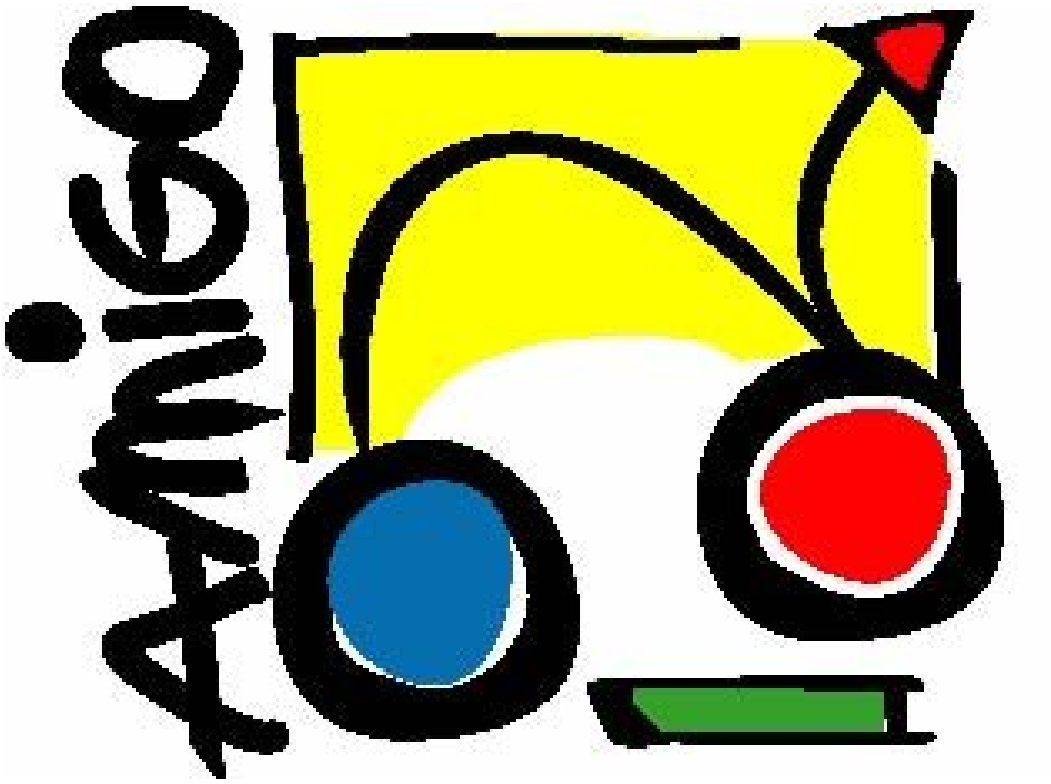

IST Amigo Project

Deliverable D10.2

Training activities University of Paderborn

Public

Project Number	:	IST-004182
Project Title	:	Amigo
Deliverable Type	:	Public

Deliverable Number	:	Deliverable D10.2
Title of Deliverable	:	Training Activities University of Paderborn
Nature of Deliverable	:	Training Report
Internal Document Number	:	Amigo_D10.2.doc
Contractual Delivery Date	:	31.08.2005
Actual Delivery Date	:	31.08.2005
Contributing WPs	:	WP 10
Author(s)	:	Reinhold Häb-Umbach, Jörg Schmalenströer

Abstract

This deliverable documents a student project on speech signal processing conducted at the Department of Communications Engineering of the University of Paderborn. It describes typical hardware equipment and software infrastructure suitable for real-time processing based on the open source operating system Linux. Experiments on speaker position tracking and acoustic echo cancellation have been set up and carried out. The gained experience serves as valuable input for the design of training courses for SMEs or other industrial parties.

Keyword list

training activities, speech signal processing, acoustic infrastructure

Table of Contents

1 Introduction

In order to help home networking make inroads in the general public, the Amigo project focuses on two key issues: First, it brings together major players of the communications, computer, consumer and home automation domain to establish a truly interoperable open middleware. Second, it develops intelligent user services to make the benefit of home networking tangible to the end user.

This deliverable is concerned with training activities aiming at the second goal of the Amigo project. It describes a student project on speech signal processing conducted at the University of Paderborn. Speech is a major input / output modality of intelligent user services, from which on the one hand an automatic speech recognizer may extract explicit user demands, but which on the other hand, may also be used to glean information about the (social) context and thus input for implicit reasoning. Further, speech signal processing is required for unobtrusive, seamless human-to-human telecommunication, e.g. in the extended home environment.

As an example of the kind of processing required this deliverable describes the setup and execution of experiments on speaker position tracking and acoustic echo cancellation. It contains both a description of a typical hardware and software infrastructure and of application software to realize the needed functionality.

The experiments have been setup and carried out by Electrical Engineering students under supervision of members of the Department of Communication Engineering, University of Paderborn.

This served as a “dry run“ for a later training of SMEs or other industrial parties. The design and implementation of the experiments, as well as the experience gained during the actual execution of the student project are valuable input for the later SME training courses, as is documented in this deliverable.

The participating students came from different fields, such as Electrical Engineering, Computer Science, and “Economics Engineering” (“Wirtschaftsingenieur”, a combination study comprising Electrical Engineering and Economics). As such their background can be considered representative of the background of people who attend the SME training to be developed. Target groups of the SME training are:

- Project managers or other decision makers with an academic background in Electrical Engineering, Computer Science or a related field
- Application developers and engineers from research and development departments with some background in Digital Signal Processing and Communications but with no special expertise in speech and audio processing.

The goals of this student project were that the participants acquire the following skills:

- To gain a basic familiarity with software and hardware for acoustic signal processing
- To develop a solid understanding of the potentials and limitations of state-of-the-art speech processing
- To be able to assess the computational and memory demands of today’s speech signal processing algorithms

This document is organized as follows. In section 2 the hardware equipment is described, which is followed by a description of the operating system and the hardware-software interface. Section 5 describes the application software and section 6 the actual experiments. Finally, in section 7 we summarize the conclusions drawn for a later SME training, followed by some general conclusions given in section 8.

2 Hardware

A short introduction of the hardware used in the laboratories of the Department of Communication Engineering will be given in this section. More detailed information can be found on the websites of the different companies or in newsgroups. Figure 2-1 shows the basic setup of the hardware for the experiments.

The equipment should be treated with care. Amplifiers and volume controls should be used with caution to avoid acoustic feedbacks and over modulation.

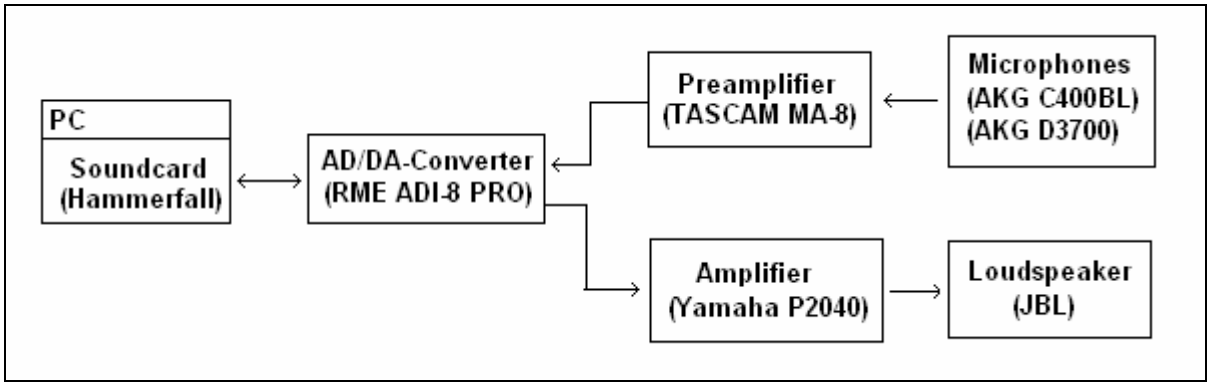

Figure 2-1: Basic setup of the hardware

2.1 ADAT

The ADAT (Alesis Digital Audio Tape) format was originally developed to record high quality music on IDE-harddisks. As the format works very efficiently, other companies adopted it and transferred it to other domains.

Our hardware uses the ADAT format to transfer the data via optical cables between the Digital-Analog-Converter and the soundcard. The transmission takes place with 24 bit per channel on 8 channels with full-duplex. If the transmission needs less bits, the least significant bits are set to zero.

The ADAT cables consist of plastic (PMMA) with a 1 mm diameter and a maximum length of 5 meters. The connectors are called „Toslink Connectors“ and they are equal to the Toshiba TOCP172 connectors, which conform with the RC-5720 standard. It is possible to use fibre optic cables for larger distances.

2.2 RME Hammerfall

The RME Hammerfall soundcard is designed for the use in the x86 computer architecture in a PCI-slot. It is able to handle 3 ADAT Input/Output ports, 2 analog I/O ports and one S/PDIF I/O port. The 3 ADAT ports offer 24 independent digital input and output ports, which must have been sampled at a frequency between 32kHz and 96kHz by a digital-analog converter (for example the RME ADI 8 PRO).

RME Hammerfall technical data

- ADAT Digital In/Out, based on RME Bitclock PLL
- 1 x SPDIF Digital In/Out, based on RME DIGI96 technology

- 1 x wordclock In/Out (BNC) on expansion board
- 1 x Breakout cabel for coaxial SPDIF
- 1 x ADAT Sync In (9-pol Sub-D) for sample exact transfer
- Zero wait state PCI-busmaster Interface with additional burst FIFO (130 MB/s full-duplex transfer rate)
- 52 Mono Channels in block mode, organized as 32 Bit ASIO doublebuffer
- Hardware ASIO; 0% CPU demand on using all 52 Channels
- 130 MB/s transfer rate using record and playback results in 9% PCI bus demand when all 52 channels are used
- Enhanced zero latency monitoring
- Hardware S/MUX: 12 Channels Record / Playback at 96 kHz/24 Bit.
- Full support of ALSA / JACK on Linux systems

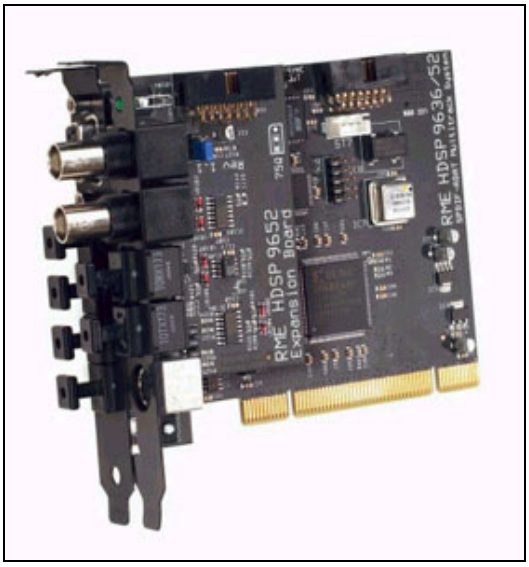

Figure 2-2: RME Hammerfall

2.3 TASCAM MA-8

The MA-8 converts balanced XLR microphone level input signals to unbalanced line level output signals without any loss in sound quality. Ultra low-noise integrated circuits guarantee pristine audio quality throughout the conversion process. The MA-8 can provide 48 V phantom power for use with high-quality studio condenser microphones and each channel's 55 dB input trim range ensures optimal level settings for a wide range of signals.

TASCAM MA-8 technical data:

- 8 high-quality microphone preamps
- 8 XLR balanced microphone inputs
- switchable 48 V phantom power
- Eight 6.3 mm direct output jacks

- 55 dB Input trim control on each channel
- Ground-Lift-Switch

Figure 2-3: TASCAM MA-8

2.4 RME ADI-8 PRO

The ADI-8 PRO is an 8 channel analog to digital and digital to analog converter. The compact 19" 1U rackmount enclosure includes several features, like Intelligent Clock Control (ICC), SyncCheck®, SyncAlign®, TDIF/ADAT converter and Bit Splitter. AD- and DA-circuit automatically operate either independently or linked. The 24 bit converters with 128 times oversampling achieve more than 110 dB real dynamic range. All digital inputs and outputs operate at full 24-bit resolution.

Technical data RME ADI-8 PRO

- 8 channel AD-Converter, completely symmetrical and DC-coupled audio path, 116 dBA AD
- 8 channel DA-Converter, DC-coupled audio path, balanced output, 112 dBA DA
- ADAT optical inputs, 24 bit, based on RME's bitclock PLL for sample accurate lock
- ADAT optical outputs, 24 bit, fully compatible to all ADAT optical inputs
- TDIF-1 interface, 24 bit, Low Jitter PLL, Emphasis support, DA-88 compatible
- Bit Split/Combine, Yamaha 02R compatible technique to record 24 bit data onto 16 bit machines
- Copy Mode adds a unique 24 bit ADAT to/from TDIF converter
- Bit Split/Combine, Copy Mode, ADAT and TDIF usable in all combinations
- Digital Patchbay operation, allows to copy, duplicate and distribute the digital input signals
- SyncCheck, unique technology to check clock synchronization
- Virtual Sample Buffer, allows to use the internal Low Jitter Clock (quarz crystal) even for DA-conversion
- Automatic storage of all settings

Figure 2-4: RME ADI-8 PRO

2.5 Microphones

During the experiments two types of microphones are used. These are boundary layer microphones AKG C 400 BL and hand microphones AKG D 3700. Both microphone types are useful for different tasks.

2.5.1 AKG C 400 BL

The AKG C 400 BL boundary layer microphones are typically used in conference rooms and theatres, where distant talk microphones are needed. Working on a condenser converter with permanent charge, the microphone needs a supply voltage of 9V to 52V. Integrated in the XLR-connectors of the microphones are the phantom power converters, which reduce the voltage to the required value. The distance between the speaker and the microphone can be up to 5 meters and the frequency range of the microphone is optimized for the human speech.

Figure 2-5:AKG C 400BL

2.5.2 AKG D 3700

The AKG D 3700 microphone is a close-talk microphone designed for singers and optimized for the human voice. So it has a broader frequency range than the AKG C400BL. Its concept achieves the best results at short distances. A hypercardioid characteristic reduces acoustic feedbacks, and thus improves the signal quality.

The microphone does not need a phantom power, but it is less sensitive than the boundary layer microphone.

Figure 2-6: AKG D 3700

2.5.3 Technical data

	C 400 BL	D 3700
Transducer Principle	Condenser prepolarised	Dynamic pressure gradient
Polar pattern	Hypercardioid	Hypercardioid
Frequency Range	150 - 10000	60 - 18000
Sensitivity	13,5 mV/PA (bei 1kHz)	2,5 mV/PA (bei 1kHz)
Electr. Impedance	200 Ohm	< 600 Ohm
Load Impedance	2000 Ohm	2000 Ohm
Sound Pressure Level THD 1% (3%)	95 (107) dB SPL	147 (156) db SPL

Boundary layer microphones (AKG C 400 BL) show a higher sensitivity than "normal" hand microphones (AKG D 3700), so they can be used to build up microphone arrays.

2.5.4 Usage of microphones

Boundary layer microphones are used, if the distance between the speaker and the microphone is larger than 0.5m. They are well suited for varying distances and normally used in microphone arrays.

The hand microphones should only be used for short distances and if the speaker can carry the microphone around.

The close-up range effect can be observed for all acoustic sources at short distances with all types of dynamical microphones. It affects a strong accentuation of low frequencies, however at high power levels the signal can be strongly modified, which results in intelligibility problems. A short distance between speaker and microphone leads to a full and soft voice. If the distance is increased the voice has more echoes.

If a microphone is used for short distances, the speaker should never speak directly into the microphone. Otherwise breathing sounds are absorbed and the quality of the recording is worse. This can be avoided by speaking just above the microphone, to receive a natural sounding and balanced voice.

2.6 JBL Control 1C

The JBL Control 1C is a high quality loudspeaker for speech signals with a recommended power amplifier range of 150 watts. It consists of a 5.25" woofer and a 0.75" tweeter and has a nominal impedance of 4 ohms. Frequencies between 120Hz and 20 kHz can be played back.

Figure 2-7:JBL Control 1C

2.7 Yamaha P2040

The Yamaha P2040 is a high performance amplifier, which can be used in 4-channel and 2-channel mode. For the experiments the 4-channel mode must be selected (switch on the rear side).

Figure 2-8: Yamaha P2040

2.8 Acoustic feedbacks

Acoustic feedbacks occur, if the direct path between the microphones and the loudspeakers is too short. An acoustic loop can be set up by the microphone, the loudspeakers and the amplifiers, which causes an amplification of the recorded signals and a playback of these. The power level increases autonomously and the loop can only be broken by reducing the microphone amplifier level.

Pay attention that the distance between the microphones and the loudspeakers is well chosen and that the microphones are not directed towards the loudspeakers, if the played signal is also the recorded one.

3 Operating system

The operating system needs real-time performance to handle audio signals with low-latencies. Additionally enough reserves are necessary in order to provide real-time processing for subsequent processes like speech recognizers. The operating system described in the following is running on a Dual Xeon Workstation with 1GB Ram and a Hammerfall soundcard and is based on Linux and Open Source Software.

Suse Linux 9.2 with a linux-2.6.8-24.11 kernel is the basic operating system. All kernel patches should be installed to minimize the risk of security holes. You can use YOU (Yast Online Update) from the SuSe Linux AG for example. The kernel sources and the gcc compiler with all libraries must be installed.

3.1 Real-time module processing

The standard kernel from most of the distributions has no real-time capabilities. This leads to high latencies and major problems with real-time audio signal processing. The Jack Server for example displays “xruns” to inform the user that samples got lost, because the Jack process did not get enough CPU time. Other problems might be buffer overruns or underruns in the Spark software. These problems must be minimized to get a stable audio processing system. Spark (Speech Processing and Recognition Kit) is the proprietary speech processing software developed at the Department of Communications Engineering, University of Paderborn.

First real-time capabilities for the kernel have been installed. For this the kernel got a new module (realtime-lsm):

The following steps have been done:

- Download realtime-lsm-xxx.tar.gz from "http://sourceforge.net/projects/realtime-lsm" and unzip it in /root/realtime-lsm-xxx/ for example
- Become root
- cd /usr/src/linux
- zcat /proc/config.gz > .config
- make oldconfig
- make bzImage
- cd /root/realtime-lsm-xxx/
- make
- make install
- cd /usr/src/linux
- make modules
- cp System.map /boot
- Search for bzImage and

  cp bzImage /boot/RealtimeKernel
- Add boot option to Bootloader

If everything works well, the kernel should be compiled and installed.

After rebooting the system the kernel has the ability to load the realtime-lsm module. The user root has to enter the following commands, or these commands can be added to /etc/init.d/boot.local

- modprobe realtime any=1
- modprobe realtime gid=29
- modprobe realtime mlock=0

Every user from the group 29 can now start processes with real-time privileges. On our system this step improves the performance of the audio signal processing noticeably.

3.2 Installation of Jack

The sound server Jack is a Open Source Project and can be downloaded from "http://jackit.sourceforge.net/download/". It is also possible to install Jack directly from the CVS system to get the newest version.

Installation from CVS is done by:

- cvs -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/jackit login

  Hit <enter> if it asks you for a password
- cvs -z3 -d : pserver:anonymous@cvs.sourceforge.net: /cvsroot/jackit co jack
- cd jack
- ./autogen.sh
- make
- make install

If stability problems are obvious, please install the latest stable version from the available rpm-packages.

3.3 Mixer setup

Figure 3-1 shows the HDSP mixer panel. A detailed description of the software can be found on the homepage of RME Audio. The basic fader options are (description from RME Homepage):

- Top row: Hardware input signal level i. e. Fader. Per fader and routing window, any input channel can be routed and mixed to any hardware output (third row.)
- Middle row: Playback channels (playback tracks of the software). Per fader and routing window, any playback channel can be routed and mixed to any hardware output (third row).
- Bottom row: Hardware outputs. Because they refer to the output of a subgroup, the level can only be attenuated here (in order to avoid overloads), routing is not possible. This row has two additional channels, the analog outputs.

Figure 3-1: HDSP mixer panel

For the experiments the faders should be placed as shown in the picture. In the top row the faders must be zero and in the other rows the faders must be up. Sometimes it will be necessary to reduce the input or output level to avoid overmodulation. If it occurs, a red light above the fader flashes. In this case the signal has been truncated.

4 Hardware–Software Interface

The audio server Jack offers the opportunity to use the RME Hammerfall soundcard in real-time and to work sample synchronous. Furthermore, Jack provides an abstract interface, which can be used with every supported hardware. This increases the compatibility to upcoming hardware and reduces the implementation effort.

We will give a short overview of Jack and Linux, as a starting point and after that the Spark implementation of the jack frontend is briefly explained.

4.1 Jack Audio Connection Kit

Jack is a low-latency audio server, written for POSIX conformant operating systems such as Linux. It can connect a number of different applications to an audio device, as well as allowing them to share audio between them. Its clients can run in their own processes (ie. as normal applications), or they can run within the Jack server (ie. as a "plugin").

Jack was designed basically for professional audio work, and its design focuses on two key areas: synchronous execution of all clients, and low latency operation.

If special demands on real-time abilities exist, a special real-time kernel or a kernel with real-time abilities must be installed. This allows programs to execute in a real-time environment. Wrong implementations of loops can cause system crashes and some kernel patches have security holes.

4.1.1 Start parameters Jack Daemon

After installing Jack and all demanded packages on the system, the Jack Server must be started through the Jack Daemon process.

The system call is:

```
user@Linux:> jackd   --realtime -d alsa -d hw:0 -p 128 –r 48000
```

This call starts the Jack Server with ALSA (Advanced Linux Sound Architecture) compatibility. The sampling rate should be selected according to the hardware and if a downsampling step is necessary, the ratio between the sample frequency and the desired frequency should be an integer value. Through selecting the frames per period (parameter -p), the streaming rate of jack and with this the delay is chosen. A small value (-p 64) results in a short latency and high computational effort. So the value for the parameter “-p” should be also chosen accordingly to the available computational power.

4.1.2 C++ Implementation of a Jack Client

The complete implementation of a Jack client can be seen on the Jack website. There are simple examples for clients, which help to implement a client on your own. Here we will present the principal process of creating and registering a Spark client on the Jack server.

The following steps must be done:

- client = jack_client_new ("SparkPlay")

  Create a new client named “SparkPlay” and register it to the Spark Server

- jack_set_process_callback (client, process, &thread_info)

  The call-back function “process“ is chosen for the client, which is executed if the hardware sends an interrupt. This is a hardware driven function call of a thread and the “process” function should be small and fast, in order to minimize the latency. If the function is too slow, the Jack Server will crash.

- jack_on_shutdown (client, jack_shutdown, &thread_info)

  This call defines the function ("jack_shutdown") executed after the determination of the client.

- jack_activate (client)

  The Jack client is activated and registered, but no types are defined.

- jack_port_register(client, name, JACK_DEFAULT_AUDIO_TYPE,JackPortIsOutput, 0)

  This function call defines the client as an output and the client becomes visible in the Jack connection manager. Before connecting the new client to other clients, ports must be declared.

- jack_connect (client, jack_port_name(ports_name) ,dest_name)

  This creates the port “dest_name” and a connection between the port and the port "ports_name" saved in “jack_port_name“.

This brief overview of the basic jack functions shows the necessary steps for registering a jack client to the server. You can say, that the hardware is assigned to call selected functions as a response to interrupts. So it is not necessary to wait for new data from the hardware, or to ask for new data. The call-back functions are able to activate the processing functions of a system after new data is available. If no data is available, the processing functions fall asleep and, by doing this, minimize the computational effort. In combination with a real-time kernel a sound processing with low-latency is possible.

4.2 Spark Modules

The software Spark consists of several modules for recording, playback and processing of data. First we will present the implemented modules for recording and playback, which are used in the experiments. Then we will discuss the advantages and disadvantages of the chosen solution.

The main target of the implementation is a fast and failure-free processing of data. Easier solutions for addressing a soundcard are known, but most of these approaches have the problem of buffer underruns or overruns.

Think of a system, which runs normally in real-time, but for some time periods it is slower. An online speech recognizer for example demands in speech pauses more computational power than during speech sequences. In the mean it might run in real-time, but if the sound architecture has not enough memory to save the samples, the speech recognizer will crash or samples will be lost. Normally a buffer overrun is signalled and the speech recognizer is stopped. This behaviour can be accepted on a test system, but never on a system for an end-user. Imagine a home system crashing every few hours, because nobody gave a command.

The Spark architecture avoids these problems by using several ringbuffers in the main memory. Actually the Jack Server is running as a asynchronous thread, starting the call-back function to write the samples in the first ringbuffer. This is always possible and the call-back function has the possibility to write over the read pointer of the ringbuffer. Loosing some samples is better than crashing the whole system. The call-back function is a small one, in order to guarantee a fast execution.

From the ringbuffer the thread “Mic_Thread” reads the data, resamples them from 48kHz to 16kHz and writes them to the ringbuffer “SparkBuffer”. This step is resource demanding, so a multi-processor system spreading the tasks on several CPUs improves the system recognizably. The Spark JackMic module can read the data from the ringbuffer and send it to the other modules.

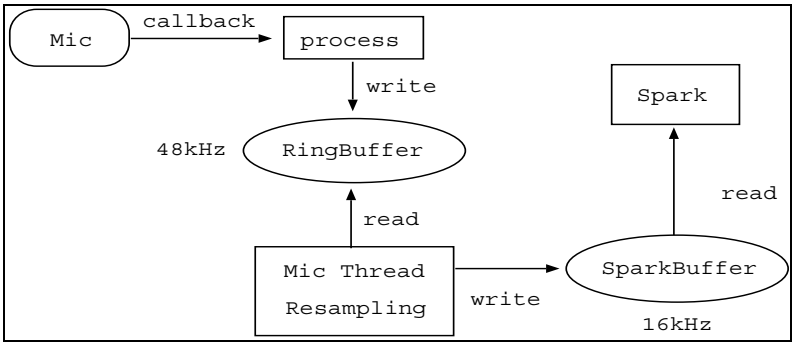

Figure 4-1: SparkMic module for Jack

If Spark is slower than real-time, the memory “SparkBuffer” is overwritten and samples are lost. No interrupt will occur or error message will be displayed, to avoid irritated end-users. Spark assumes that only a few samples are lost and that the impact is minimal for the running system. Exceptional loss of data will lead to severe problems with speech recognizers and other software modules, so the internal counter for buffer overruns should be checked while developing the system.

Optimally a system should run faster than real-time, so that the basic system has resources for exceptional tasks. The worst case is a system running real-time only in the mean, because this might cause casually data loss.

In the above figure the schematic description of the Spark microphone module is given. Clearly the three independent threads (process, Mic_Thread, Spark) are identifiable. On a multiprocessor system this asynchronous implementation achieves good performance.

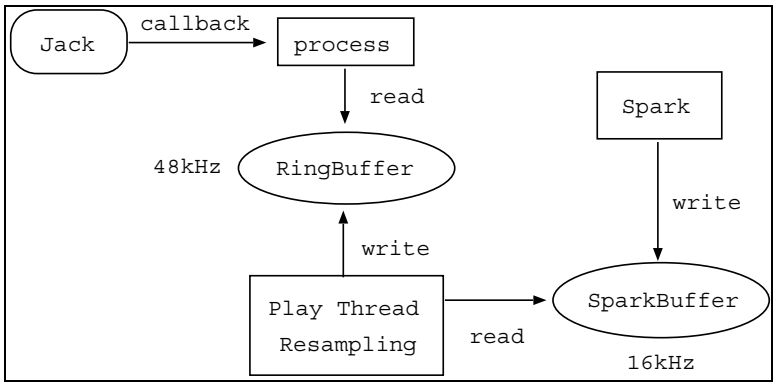

Figure 4-2: SparkPlay module for Jack

In figure 4-2 the module for playback is depicted. The architecture is equal to the recording module, with two ringbuffers and three asynchronous threads. For playback the sampling rate must be changed from 16 kHz to 48 kHz, as all internal filters work with 16 kHz and the hardware needs 48 kHz. The high resource demanding step of resampling is done by Play_Thread.

4.3 View on the system

In this chapter some aspects of the system will be discussed. Each subsystem produces delays, which have to be estimated by experiments. An optimal system would be sample synchronous and delay free. Such a system is not realizable, because every system component, like the lowpass filter for the resampling steps or the D/A-A/D converter delays the signal. Minimizing these delays improves the system, but it is impossible to eliminate all delays.

An efficient software design helps to build an approximately real-time system. But a personal computer will never be a real-time system, because other tasks are running in parallel and demand also the CPU power. Before building the real-time kernel and optimizing the software, each system daemon and software should be deactivated to reduce the amount of running tasks. Each running task can start processes during the experiments, so that the CPU may experience more load and the speech processing system gets less CPU time. The result might be, that the system sporadically crashes.

Another important aspect is the system transfer function of the audio hardware. It can be measured by connecting the input ports of the DA/AD-Converter with its output ports. Playing a known signal, for example white gaussian noise on these ports and recording it at the same time, allows the estimation of the system response. Figure 4-3 shows a system response measured with the described system.

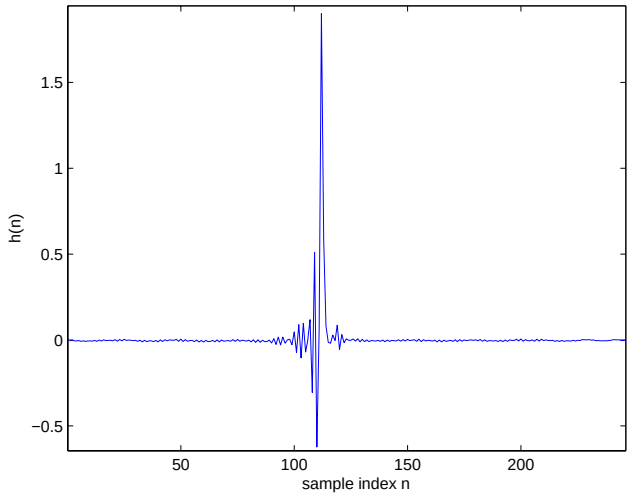

Figure 4-3: System impulse response

Keep in mind that every played and recorded signal is modified by the system impulse response and adds for example to the room impulse response.

5 Algorithmic Software and Modules

Specific aspects of the software used in the experiments will be the topic of the following section. Resampling methods are introduced first, to show some basic processing steps. Then the Filter-and-Sum-Beamformer (FSB) and the Adaptive-Interference-Canceller (AIC) are described.

5.1 Resampling Methods

Resampling methods are an urgent need for the system, as the RME Hammerfall soundcard can only handle signals at rates between 32 and 96 kHz. Theoretically it is possible to build up the speech processing part in 48 kHz, but the computational effort would be too high for available computers. Usually speech processing or speech recognition is done with sampling rates between 8kHz and 16kHz. So we decided to implement a downsampling method for the microphone signals and a upsampling method for the loudspeaker signals.

5.1.1 Downsampling Methods

Downsampling methods require lowpass filters to restrict the signal to a certain bandwidth. The design of a filter assumes some basic knowledge about digital signal processing. For detailed information about filter designed we refer to the literature. Here, we choose a 6^{th} order elliptic filter with a maximum passband attenuation of 0.01dB and a minimum stopband attenuation of 40 dB. The filter has been designed with Matlab and a time domain implementation in C++ with 130 filter taps is working in Spark.

Several filters were designed and tested with Matlab. The finally implemented filter has been chosen as the best compromise between computational effort and speech quality.

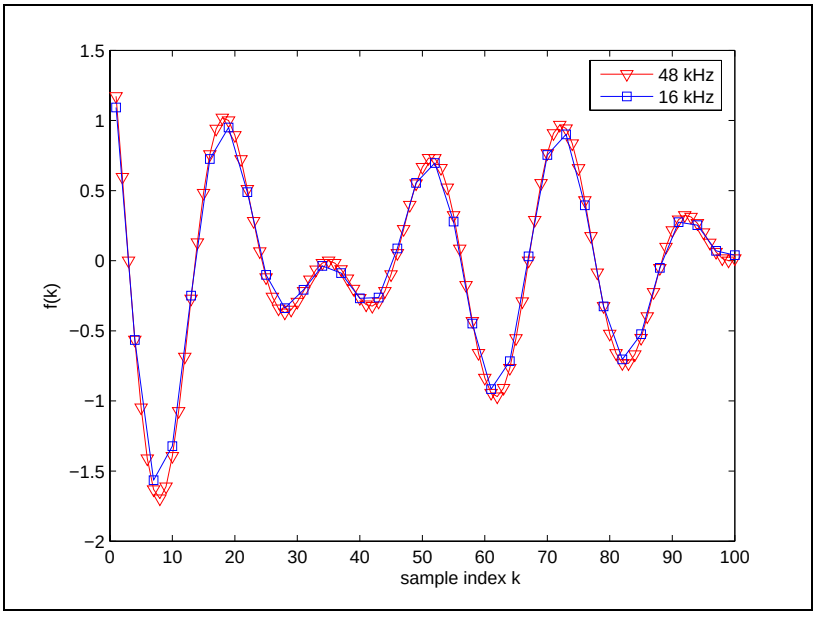

Figure 5-1: Downsampling example

5.1.2 Upsampling methods

Replaying a signal with the hardware requires an upsampling method and thus a lowpass filter. If a broadband signal should be played, for example music, the filter length must be high in order to attenuate the alias effects of the upsampling step. Alias effects can be heard as a high frequency noise and they disturb the performance of an echo cancellation system for example.

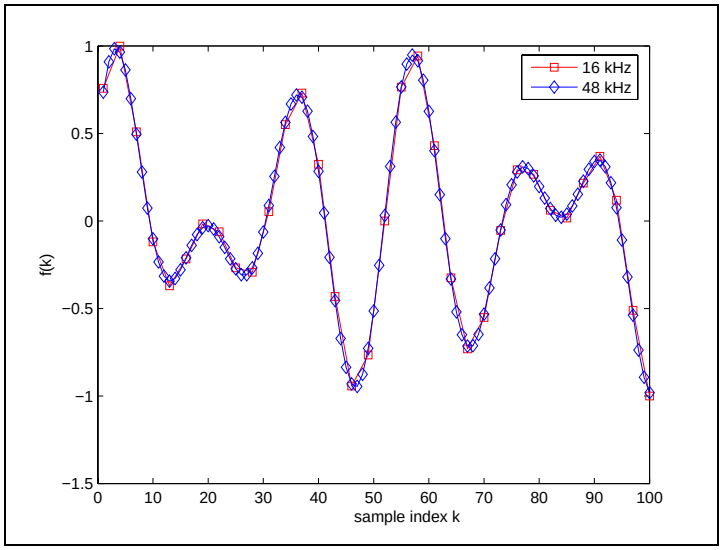

Figure 5-2: Upsampling example

Upsampling can be done by filling in zeros between two samples and filtering the result with a lowpass. It is necessary to remove the alias spectral components, otherwise the signal quality is deteriorated.

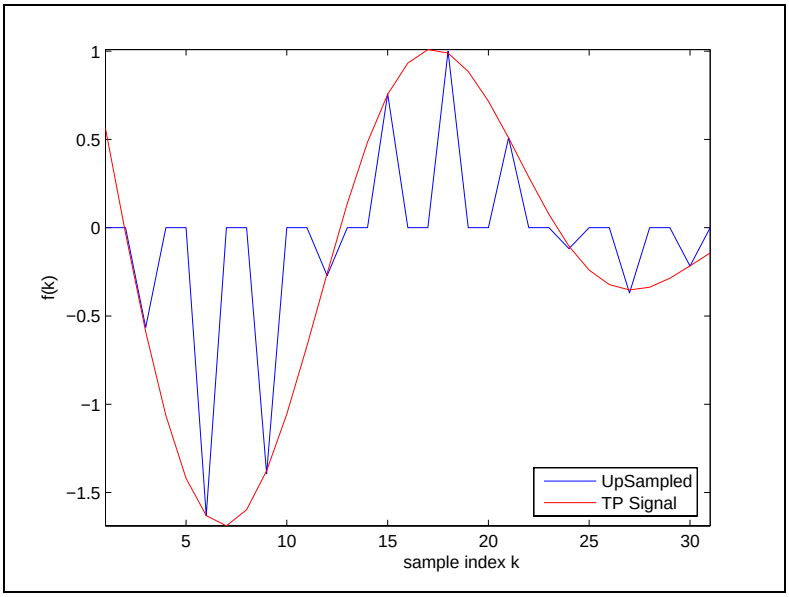

Figure 5-3: Signal before and after lowpass

In figure 5-3 a signal is displayed before and after a lowpass filter. Before using the lowpass the filled in zeros of the upsampled signal are clearly identifiable. After the lowpass the desired signal is displayed, with the higher sampling rate.

5.1.3 Lowpass Filter Design

The lowpass filter has been designed with Matlab and in the following section some aspects of the filter will be discussed. Figures 5-4, 5-5 and 5-6 show some results from the filter design with a cutoff frequency of 8kHz.

Voice quality should be the objective of each filter design, because a filter with low complexity and low voice quality is useless. Perfect voice quality with a short filter is impossible, so a compromise has to be found. Filtering in the frequency domain might be a solution to this problem, but new problems will come along, like stability for example. The FFT is a very effective way to implement filter problems, but good filter results demands a high resolution of the FFT. A high resolution needs long input blocks and this result in long system delays. A time domain implementation avoids all these problems and a low latency lowpass with good parameters can be designed. The filter used in the experiments is 130 samples long and achieves good performance.

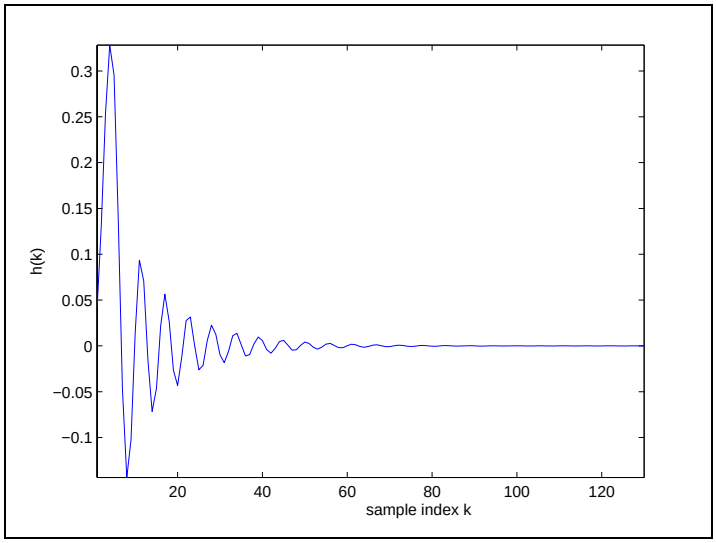

Figure 5-4: Impulse response of lowpass

If the band between the passband and the stopband must be smaller than in the chosen filter design, the amount of coefficients in the time domain must be increased or the stopband attenuation must be decreased.

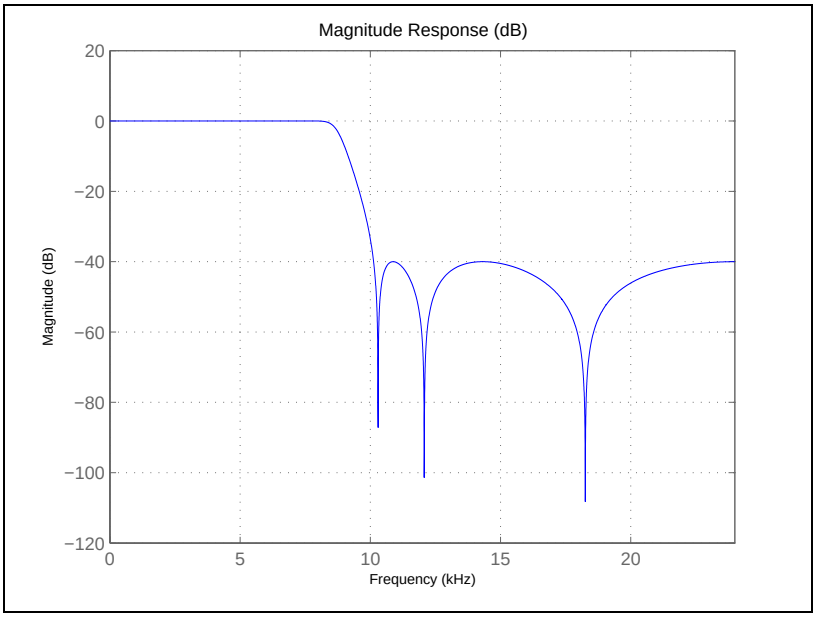

Figure 5-5: Magnitude response of lowpass

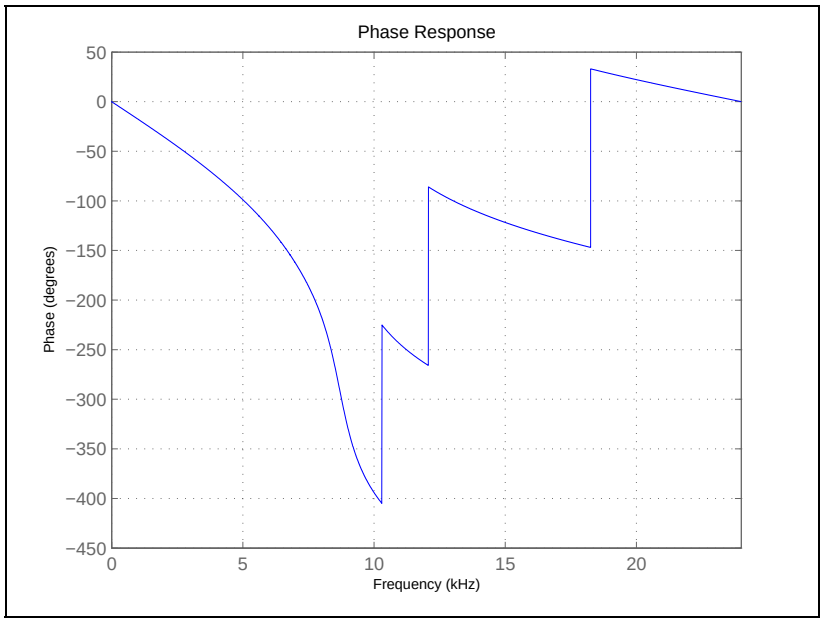

Figure 5-6: Phase response of lowpass

5.2 Filter-and-Sum-Beamformer

The Filter-and-Sum-Beamformer (FSB) is an adaptive beamformer, where the microphone signals are filtered by short FIR filters and then summed up as can be seen in figure 5-7. The signal to noise power ratio of the output can be maximised by principle component analysis: A stochastic gradient ascent algorithm estimates iteratively the eigenvector corresponding to the largest eigenvalue of the cross power spectral density of the microphone signals. The implemented algorithm shows fast adaptation and high robustness for tracking a moving speaker.

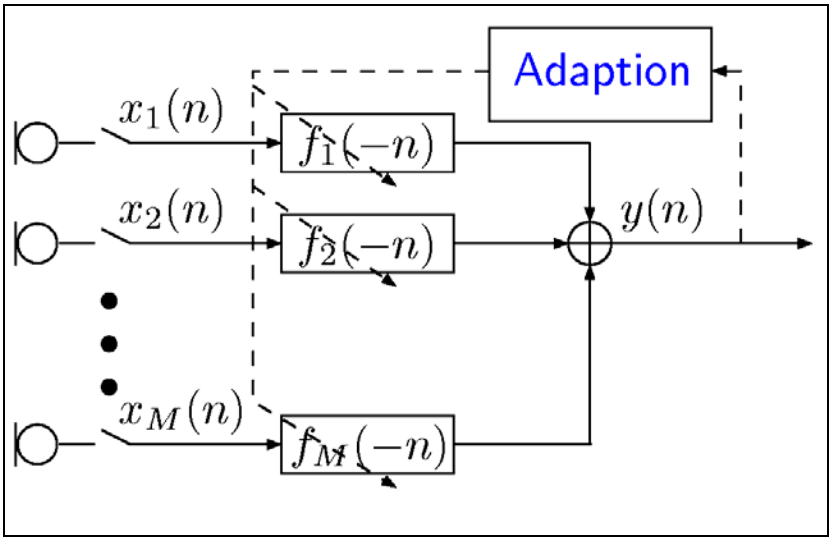

Figure 5-7: Filter-and-Sum-Beamformer

The experiment for estimating the speaker position by microphone arrays shows the advantage of the FSB. The algorithm can steer the main lobe of the beam pattern towards the acoustic source. Then it is possible to use the correlation between the adaptive filter impulse responses of two microphones, to calculate the time difference between the main peaks of the filter impulse responses. This time correspond to the direct path length differences between the acoustic source and two elements of the array and therefore to the speaker direction.

In combination with speaker identification software the speaker localisation might be used for context aggregation and service composition.

5.2.1 Direction-of-Arrival Estimation

A linear microphone array can only be used to estimate the speaker direction or more precisely the direction of arrival (DOA) of the audio signal. Position estimation can only be done by two arrays, but in the following text position estimation is used synonymously to the angle estimation, when a single microphone array is discussed.

The position estimation of a speaker for example is done by correlating the filter impulse responses of the Filter-and-Sum Beamformer. The maximum of the correlation function is a good estimate of the possible position of the speaker. An efficient implementation of the correlation can be achieved by multiplying the filter transfer functions in the frequency domain and transforming the result to the time domain. Some attention should be paid to cyclic correlation effects.

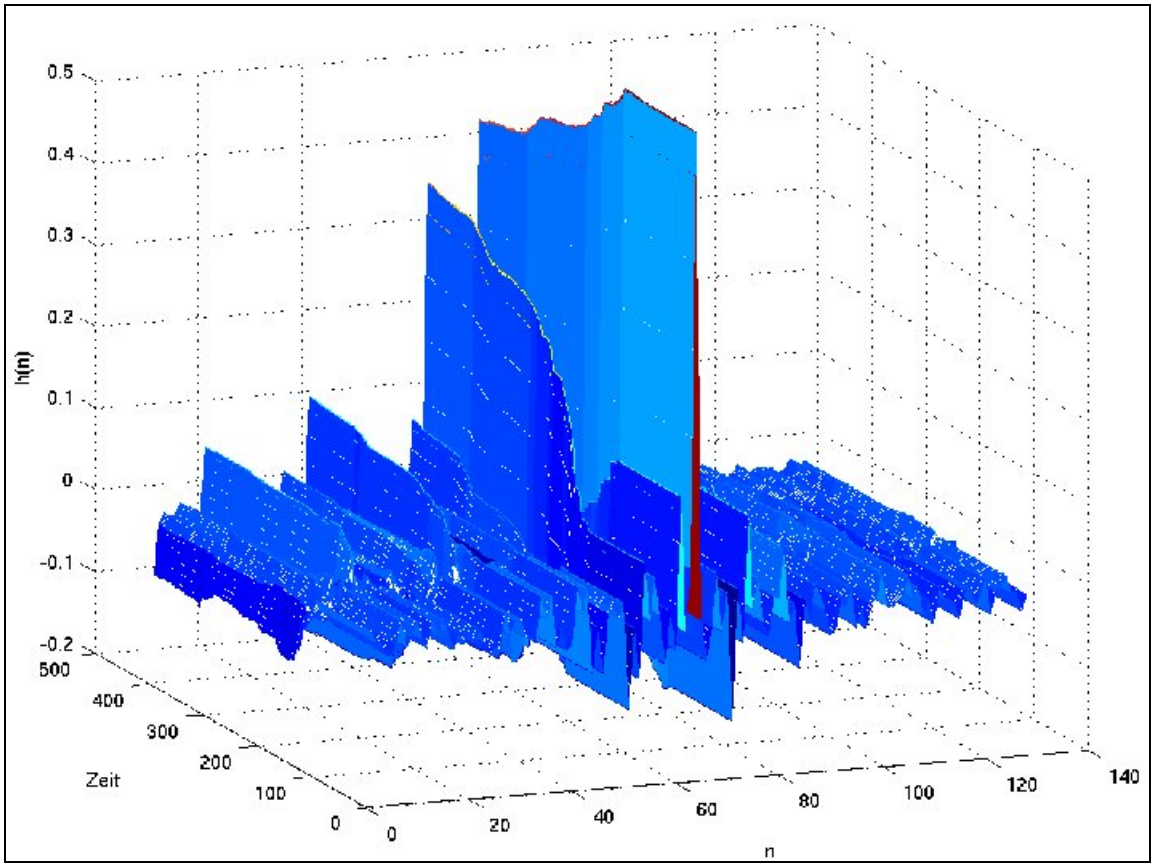

Figure 5-8: Temporal change of filter impulse response

First we will have a look at the temporal change of a filter impulse response, when the speaker position changes. Figure 5-8 shows the filter coefficients over time. The FSB has a filter length of N=128 samples and the stereo sound file has been generated from a mono sound file as reference for the first channel and a delayed version for the second channel (this means no reverberation). Every 10 seconds the delay altered between 0 samples (first speaker position) and 10 samples (second speaker position). This corresponds to a source direction at 0 degrees and 90 degrees broadside to the array, when the microphone distance is 0.22m and the sampling frequency is 16 kHz. Both speakers talk alternately for 10 seconds. During this time the FSB adapts to the speaker and with this the filter impulse response changes.

Examining figure 5-8 it can be observed, that the filter impulse response h(n) does not move from one position to another. Instead, the amplitude at one point decrases and at another point increases. In this figure the main amplitude at filter index n=63 decreases and the amplitude at filter index n=53 spontaneously increases when the speaker changes. This also means, that the correlation between two filter impulse responses will change erratically, if no interpolation is used.

Using a FSB offers also the possibility to estimate delays smaller than 1/f with f as the sampling frequency. To do that the correlation result has to be interpolated with the required resolution.

5.2.2 Microphone distance

The distance between microphones is an important parameter for a microphone array. It limits the maximum amount of samples, which can occur between the arrival at different microphones (delay between two recorded signals). Furthermore, it affects the beamformer through restricting the maximum resolvable frequency.

5.2.3 Maximum delay

Suppose the distance between two microphones is d=0.2m. In case, the speaker stands 90 degrees broadside to the array (endfire position), the largest delay between the two microphone filter impulse responses can be expected. The maximum amount of samples is calculated by

$$a_{\max} = \frac{d}{c} \cdot f_A = \frac{0.2m}{340\frac{m}{s}} \cdot 16000\frac{1}{s} = 9.41$$

if the sampling frequency is 16kHz and the speed of sound is c=340m/s. This means that the maximum delay between the arrival of a signal at one microphone and the arrival at the other microphone is 9.41 samples.

For estimating the delay between the channels, the correlation will be examined. As the theoretical delay is limited, only the values $[0, a_{\max}]$ and $[N - a_{\max}, N]$ must be examined, with N as the filter length.

An interpolation step is applied to improve the resolution of the position estimation. Experiments have shown that using 6 samples around the determined maximum of the correlation is sufficient for an interpolation. The maximum resolution should be around 2-3 degrees. Higher resolutions do not make sense because normally the estimation error is much higher than 1 degree.

5.2.4 Beamfomer properties

Spatial alias failures occur if the distance d between two microphones is too large. The maximum allowable distance according to the sampling theorem is

$$d < \lambda_{\min}$$

where $\lambda_{\min}$ is the wavelength of the signal. The maximum frequency belonging to the minimum wavelength can be calculated as follows:

$$f_{\max} = \frac{c}{d} = \frac{340\frac{m}{s}}{0.2m} = 1700Hz$$

Accordingly a microphone array with d=0.2m resolves frequencies up to 1700Hz. All other frequencies are recorded correctly, but the beamformer can not decide from which direction the sound came.

This can be observed from the beampatterns of a beamformer. In figure 5-9 beampatterns for different frequencies are displayed. With increasing frequencies the number of lobes increases also.

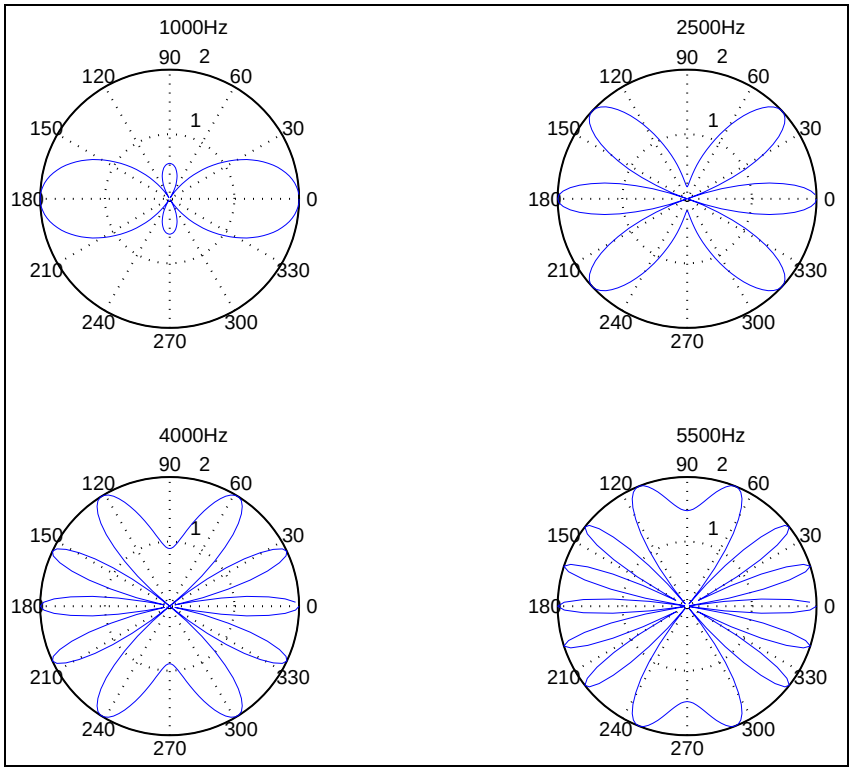

Figure 5-9: Characteristics for different frequencies (d=0.2m ; f=16kHz)

This means, that a beamformer at a frequency above the maximum resolvable, collects signals from the main lobe and combines them with signals from the other lobes. The beamformer does not adapt correctly only to the desired speaker, but also to other sources.

One solution to this problem might be the usage of several microphones with smaller distances. The microphones on the edges can be used to estimate the position of a speaker and all the microphones together are used to get a small beamformer main lobe. It is also possible to connect the microphones in groups, in order to use them in the postfilter to process certain frequency intervals.

5.2.5 System for position estimation

As discussed in the previous sections, the speaker position relative to the microphone array is estimated from the delay between the incoming signals.

For this the crosscorrelation function between 2 filter impulse responses is examined and with the maximum of the correlation it is possible to estimate the speaker direction.

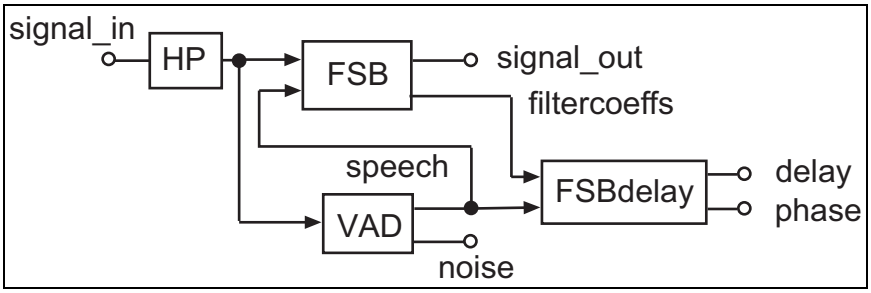

Figure 5-10: Position estimating system

In figure 5-10 you can see the position estimation system implemented in Spark. The module FSBDelay calculates from the filter transfer functions of the FSB the crosscorrelation and from this the angle between the array and the speaker. All displayed modules are scaleable, so that the number of microphones can be easily increased.

5.2.6 Simulations

The sensor signals have been simulated by a mono recording as the first signal and a delayed version of it as the second. For the following simulation the delay is between -10 and 10 samples. This is equal to the maximum delay between two microphones at a distance d=0.22m at a sampling frequency of 16kHz.

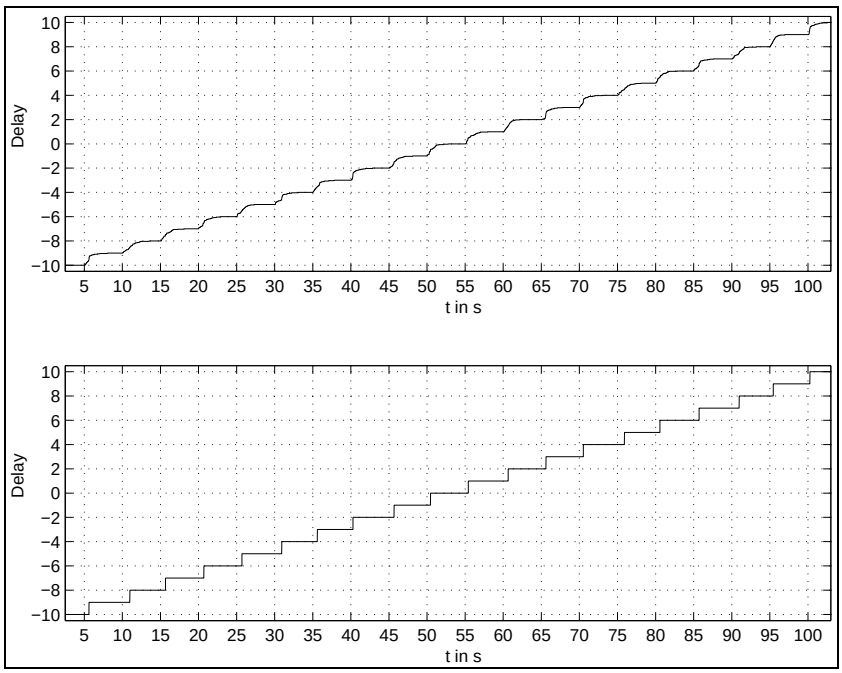

Figure 5-11: Signal delay estimation (a)with interpolation; (b) without interpolation

In order to get a higher resolution, an interpolation is applied on the result of the crosscorrelation. The interpolation method uses a upsampling method to calculate the interpolated values. For this the number of input values and the number of desired samples between two input samples must be set. The maximum of the interpolated values is used for position estimation. The amount of interpolated samples should not be too high, as the result might show more details than needed.

The impact of the interpolation is observable in figure 5-11. Without interpolation the estimated delay between the channels jumps from one integer value to another. An interpolation smoothes the estimated delay, if the extremes of the correlation function are in the interpolation interval during the change. If the difference from one delay to the next one is too large, the correlation function will jump between them, as the extremes are not in the same interpolation interval.

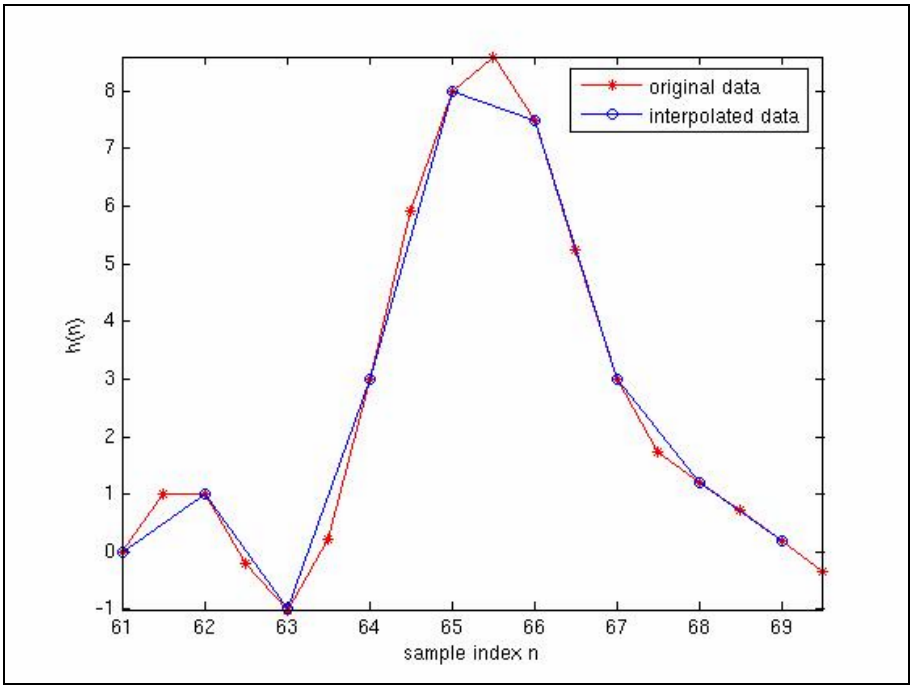

Figure 5-12: Interpolation example

The effect of the interpolation is more obvious, if the delay between two channels is not an integer value. In figure 5-13 such an experiment is shown. Without interpolation the estimated delay is rounded and jumps between the possible values around the real value. An interpolation improves the estimate and thus reduces the error.

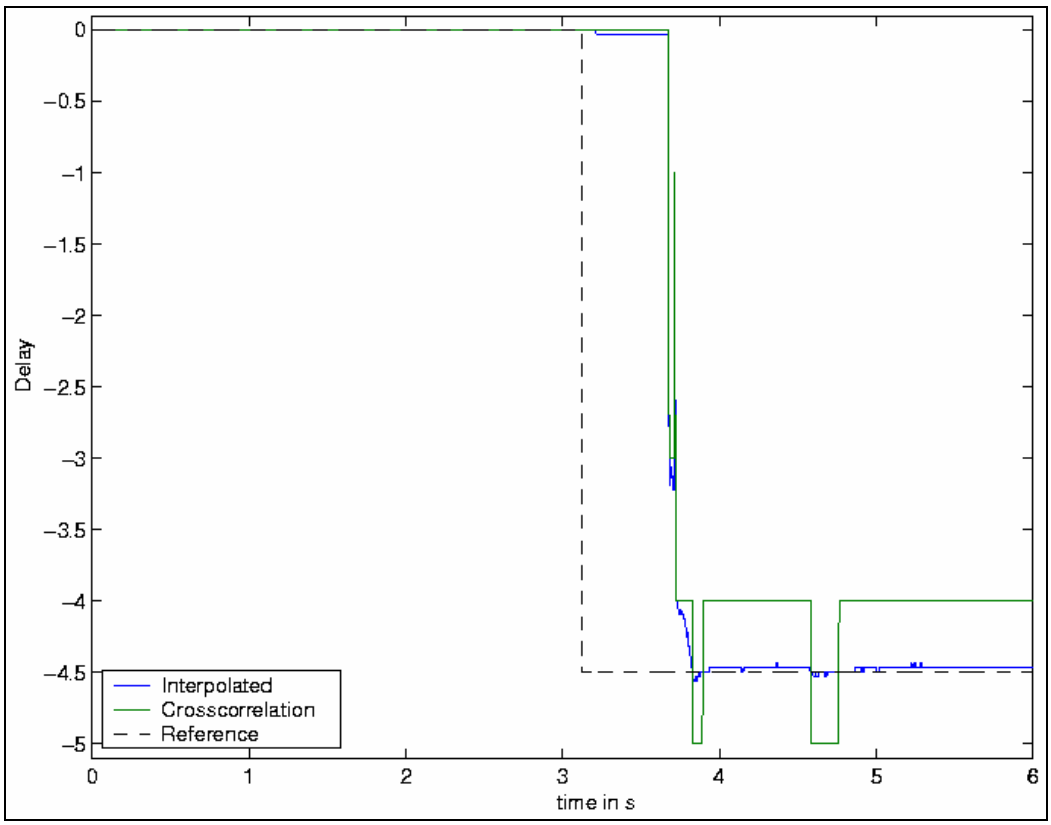

Figure 5-13: Interpolation of estimated delay

For figure 5-13 a mono recording with a sampling frequency of 32 kHz has been used. This recording has been transformed to a stereo recording, with a delay of 9 samples between the channels. After downsampling the file from 32kHz to 16 kHz the delay was 4.5 samples.

5.3 Adaptive-Interference-Canceller

The filter coefficients in the Adaptive-Interference-Canceller (AIC) are adjusted in such a way, that the part of some input signal, which is correlated with the interfering signal, is removed. The filter is updated iterative to minimize the result after the subtraction. The algorithm is implemented in the frequency domain using an overlap-save method.

Figure 5-14 shows a typical example for a setup, where a AIC is used. The far-end speaker on the right hand side speaks into a microphone and the recorded signal is transmitted to the near end. Here the signal *x* is played by a loudspeaker and recorded at the same time by a microphone. The adaptive filter tries to adjust the signal y in such a way that after the subtraction the energy of the resulting signal *e* is minimized. This can be interpreted as the fact, that the played signal is removed from the recorded microphone signal. This signal is transmitted to the far end and hopefully the far-end speaker does not hear himself. In this setup the adaptive filter works optimally, if the filter impulse response is equal to the room impulse response of the near end.

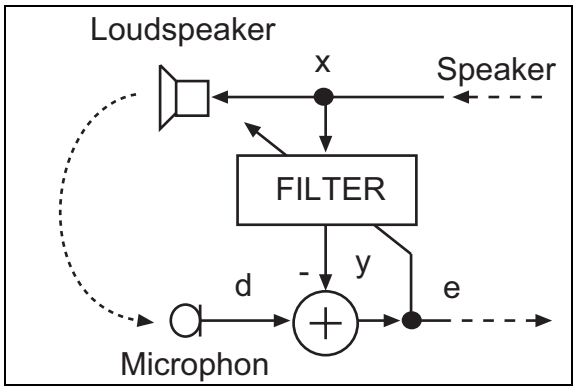

Figure 5-14: Example for echo cancellation

This kind of echo cancellation system is used in many applications, for example mobile phones or instant messengers.

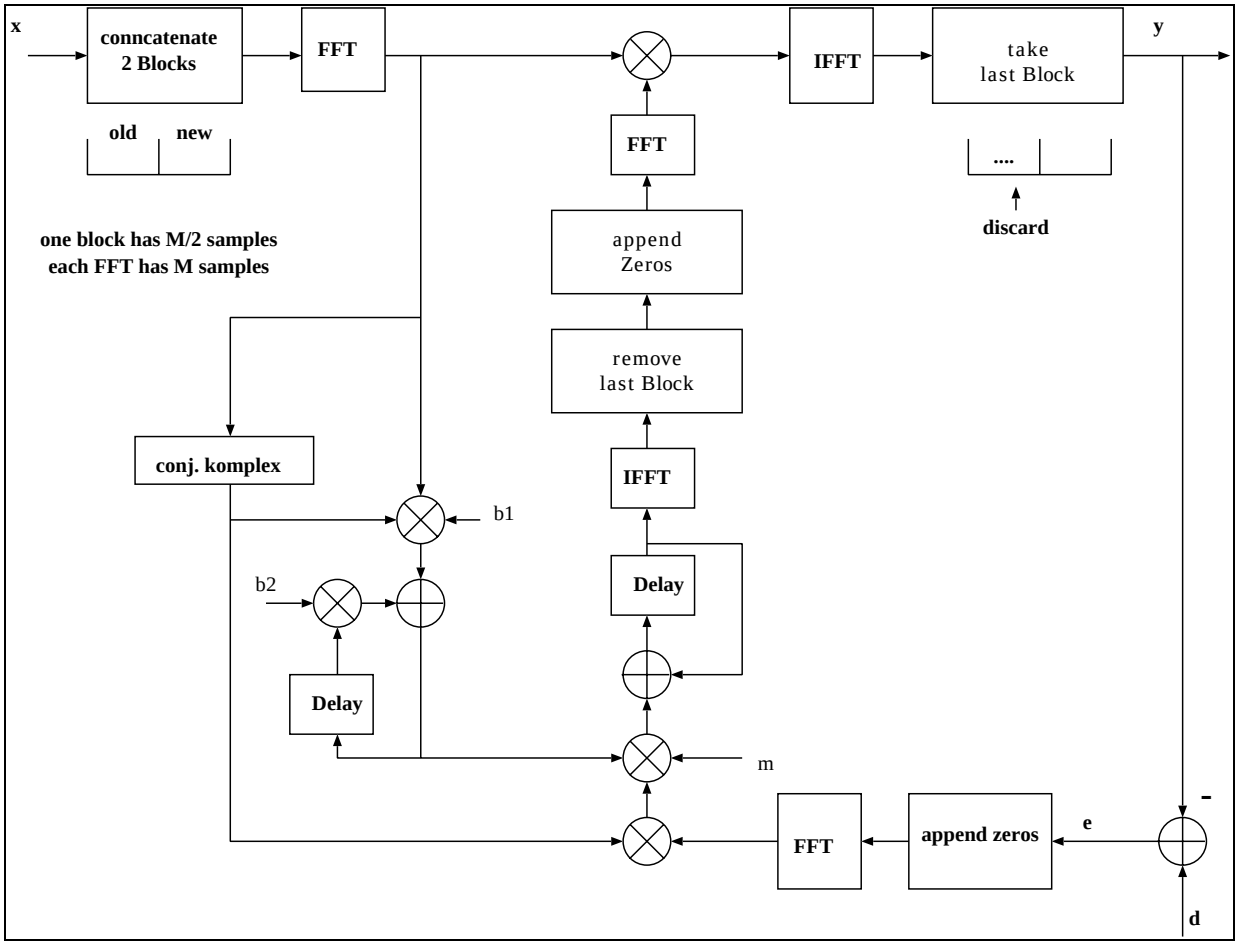

Figure 5-15: Adaptive-Interference-Canceller

Figure 5-15 depicts the frequency domain implementation of the AIC. The system works block wise, with an input frame size equal to the filter length. If the filter length should be greater than the input frame size, a different implementation must be chosen (for example Partitioned-Block-AIC).

6 Experiments

This chapter describes the setup of some experiments. First the focus lies on multi-microphone signal processing with beamforming and position estimation. The second objective is an experimental setup for echo cancellation.

6.1 Position Estimation

In this experiment two microphone arrays mounted on different walls will be used to estimate the position of a possibly moving speaker. The room has a size of 3.4m x 6.8m x 3.5m and the two pairs of microphones should be placed as shown in figure 6-1.

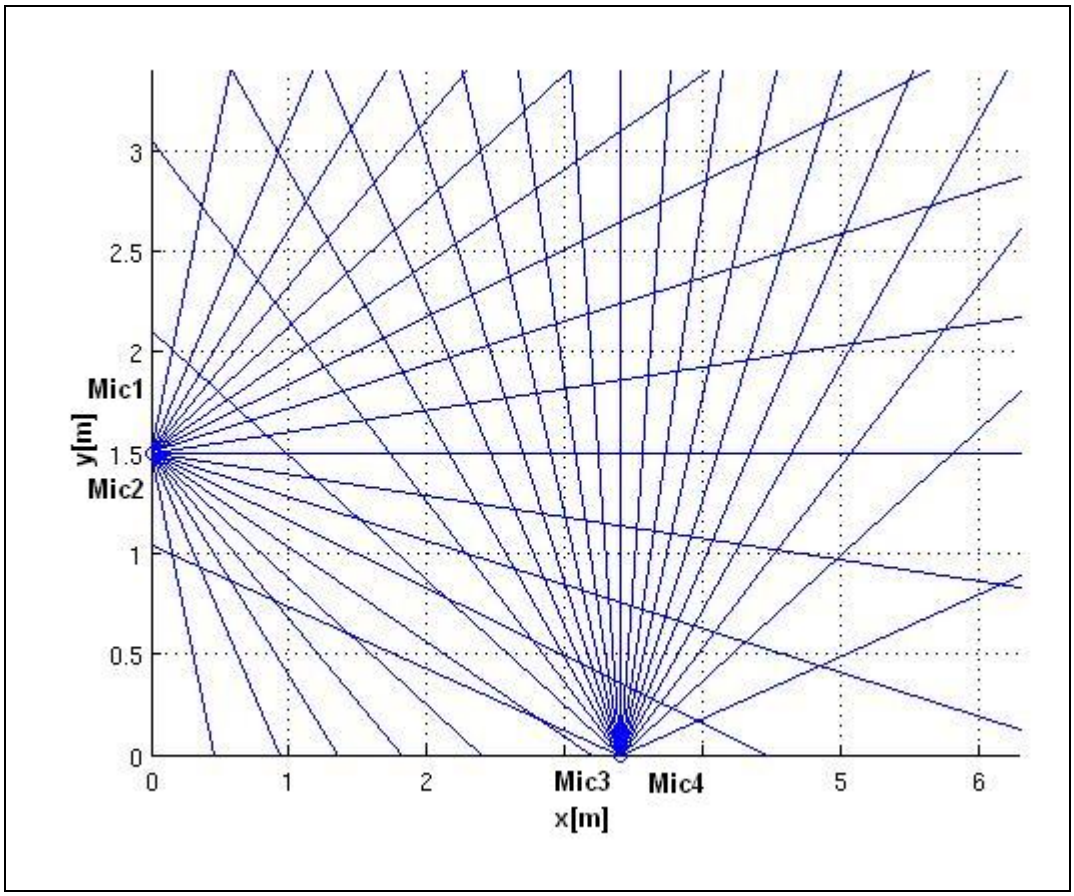

Figure 6-1: Setup for position estimation

Exercises

1. Build up the necessary hardware connections (see fig. 6-2) and use the configuration file 'position.conf' for the Spark software to estimate the position of a speaker. Can you estimate each position?
2. Try to track a slowly moving and speaking person in the room. What can be observed?
3. Use a sound absorbing board and a highly reflective board to simulate a single reflection sound path without a direct path from the speaker to one microphone array. How does the beamformer react to this?

4. Place to speakers in the room. What happens if both speakers are talking?

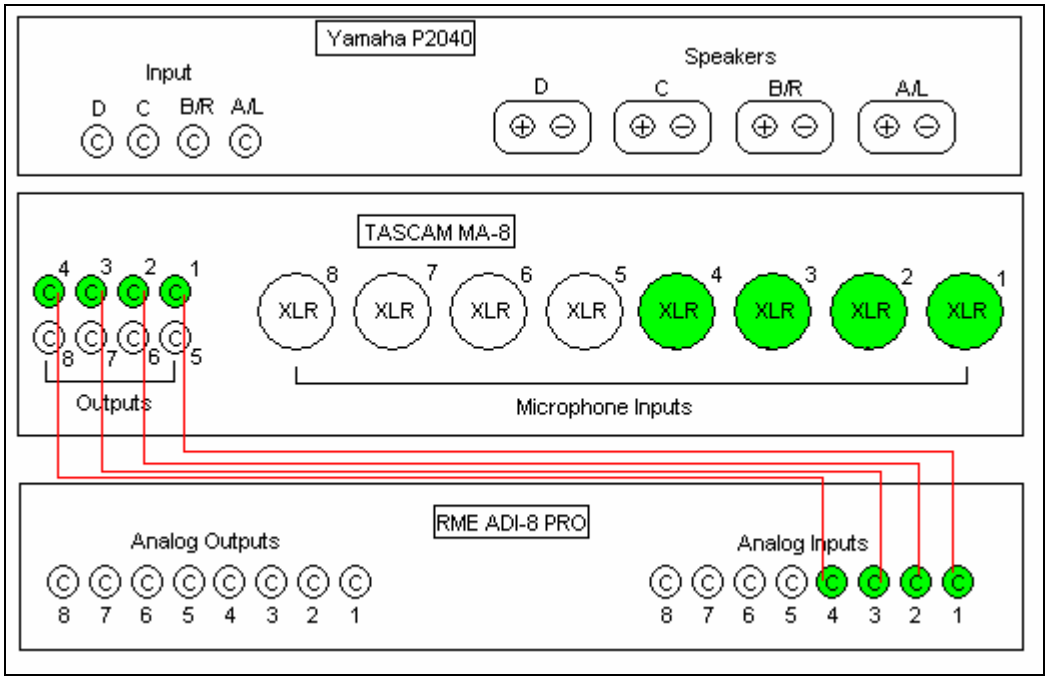

Figure 6-2: Hardware connections for the localization experiment

Figure 6-2 shows the hardware connections for the experiment. The green color shows that the connection is used.

6.2 Echo cancellation

In this experiment some aspects of echo cancellation are examined. First place a loudspeaker and a microphone in the room and connect them to the hardware.

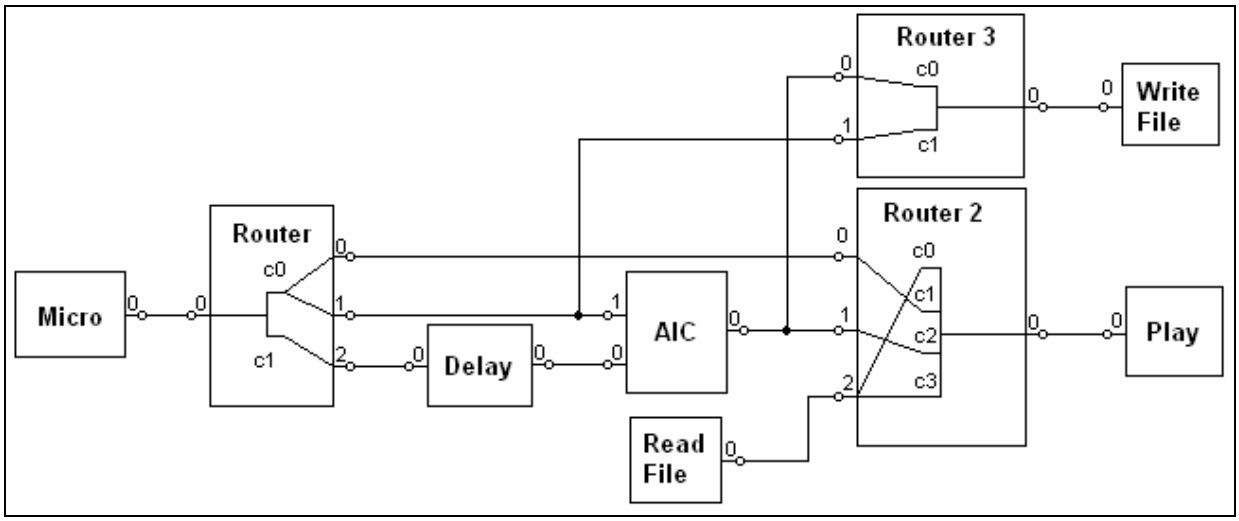

Figure 6-3: Software setup for the AIC experiment

Figure 6-3 shows the modules used in the experiment and the connections between them. The configuration file can be found in the appendix.

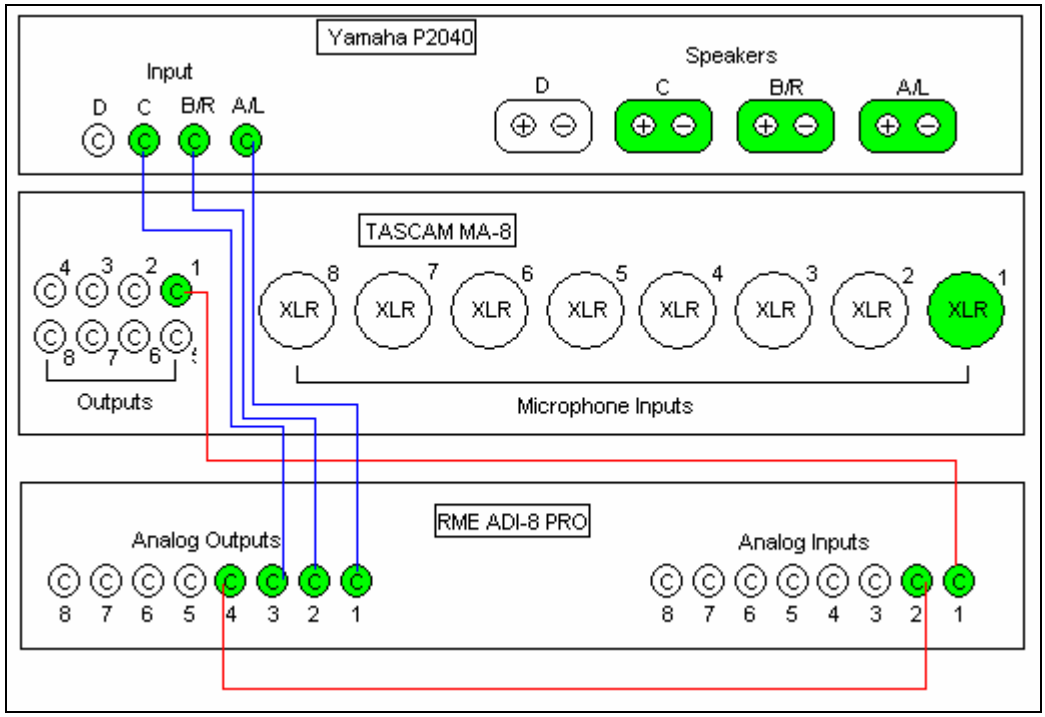

Figure 6-4: Hardware connections for the AIC experiment

Figure 6-4 shows the connections between the hardware components. Please verify all connections before the experiments are conducted.

Exercises

1. Place a microphone and a loudspeaker in the acoustic enclosure. The distance between the microphone and the loudspeaker should be about one step size. Plot the filter impulse response after convergence with Matlab. Calculate the exact distance between the loudspeaker and the microphone from the resulting impulse response. What else can you see in the impulse response?
2. Create a distinct reflection path and try to find it in the impulse response of the AIC.
3. Calculate the attenuation of the interfering signal achieved by the AIC using Matlab. Use a filterlength of 256 and 1024 samples. What do you observe?
4. Vary the stepsize of the adaptive filter and analyse the effect on the attenuation.
5. Choose a filterlength of 1024 samples and vary the distance between the microphone and the loudspeaker in the range of 0.5m to 3.5m. Compensate the minimum delay of the direct path by a delay module.
6. Now a speaker should talk while the AIC system is running. Wait for 10 seconds after the music started and then start to talk. Plot the signals and discuss them.
7. The speaker starts talking right from the beginning on (also during the acquisition phase), with short pauses. Plot the signals. What can you observe and how can you improve the system?

7 Lessons learned for SME Training

In this section we summarize some experiences we made during the experiments, which will be incorporated into the SME Training courses. We will focus on the most important issues.

Familiarity with the basics of digital signal processing, as well as some knowledge of the Linux operating system were needed to follow the experiments. Participants without this expertise will not be able to conduct the experiments. They will, however, still gain a basic understanding of the potentials and limitations of current speech signal processing. The later SME training will therefore not include the technical details of the experiments. The training module on speech processing and speech interfaces will be presented in such a way that it is accessible to the following target groups:

- Project managers and other decision makers with a technical background
- Application developers and engineers from research and development departments

Coming back to the student project, the participants acquired the following expertise:

- Basic familiarity with software and hardware
- Potentials and limitations of state-of-the-art speech processing
- Computational and memory demands of speech signal processing algorithms

The different background of the participating students had only little influence on the successful execution of the experiments. Economics Engineering students were less familiar with the algorithmic and mathematical aspects of the student project than Electrical Engineering and Computer Science students. This deficiency was compensated by somewhat extra effort and time spent by the supervisors on explaining the tasks.

This “dry run” with students revealed, however, some interesting specific issues which should be addressed in the later SME training.

First of all, the computational demands of speech signal processing algorithms are easily underestimated. Many students were impressed when they realized that 4 microphones connected to a beamformer and some adaptive filters ask for a state-of-the-art personal computer to work in real-time.

The second point was that real-time processing under the general purpose open source operating system Linux is possible, but it requires special care and experience. For example the implementation of a real-time module into the kernel and the design of the hardware-software interface need special knowledge about the operating system.

It turned out that the opportunities and limitations of acoustic scene analysis have to be marked out clearly during the upcoming SME Trainings, since there might be some misconceptions about these.

- Indeed, speech contains more than the linguistic content. An acoustic scene analysis can support the context aggregation and prediction of Amigo and through this improves the whole system.
- State-of-the-art acoustic echo compensation can achieve an impressive amount of echo suppression. There exist adaptation algorithms which track fast changes of the echo path.
- On the other hand, some tasks, which are easy for humans, can be difficult for machines. It is for example not so obvious, that robust voice activity detection is a challenging task in speech signal processing.
- Automatic speech recognition with distant microphones is difficult as the performance of the recognizer depends on the signal quality. Reverberations and other distortions

lead to a deterioration of the recognizer results. Humans have fewer problems to understand speech in reverberated situations than machines.

An important issue to point out to SMEs is that the quality of audio processing can be a means to differentiate. As the Amigo interfaces are open source, it will be easy to invent new products for the system. New audio processing algorithms, which can be developed under proprietary licences, can result in a competitive edge over competitors and by this create new business opportunities.

The last point we want to mention here is the problem of error recovery. A system can produce inaccurate estimates or errors. Some of these frequently encountered problems might be solvable by other sensors or with the help of speech independent modalities. Error handling and the solution of contradictions will be a task on a higher semantic level, as it means to collect, analyze and interpret the information from different sensors and modalities.

8 Conclusions

This deliverable documents a student project on speech signal processing conducted at the Department of Communications Engineering of the University of Paderborn. It describes typical hardware equipment and software infrastructure suitable for real-time processing based on the open source operating system Linux. Experiments on speaker position tracking and acoustic echo cancellation have been set up and carried out. The students were very enthusiastic about this hands-on experience which served as a complement to a lecture on digital speech signal processing (for more information see http://www-nt.uni-paderborn.de).

It turned out, that in particular the higher computational demand of speech processing algorithms and the complications resulting from the need for real-time processing are easily underestimated. Special emphasis should be placed on these issues in later training of SMEs or other industrial parties.

Due to time limitations, this student project focused on generic audio signal processing tasks and did not include automatic speech recognition or speech enhancement. The SME training module to be prepared within the Amigo training workpackage will, however, include material about automatic speech recognition and speech enhancement.

In summary we can state that the student project was successful, since the participants indeed acquired the skills mentioned in the introduction of this report, i.e. a solid understanding of speech signal processing, its potentials and today's limitations. Successful participants of later SME training should therefore be able to identify potential applications of speech signal processing in their application domains and assess the achievable performance and required computational demands.

9 Appendix

In the appendix the necessary commands of the software is listed. Furthermore the configuration files for the simulation software Spark can be found. All config-files are written in 3 rows in order to reduce the size of the document.

9.1 Commands

Start the Jack Daemon process

> jackd - -realtime -d alsa -d hw:0 -p 128 -r 48000

Run Spark

Speaker position estimation:

> asreng position.conf

Echo cancellation:

> asreng echo.conf

Matlab scripts

Plot AIC filter impulse response:

> PlotFilter.m

Calculate attenuation:

> CalcAttenuation.m

9.2 Configuration file Position.conf

```
[TASK]
TaskType: one_file
Input:  RMic(hw:0)
Output: OutputFile(capture.raw)
   [MODULES]
   [M]
   Name: RMic
   Type: JackMic
   Channels: 4
   FrameSize: 128
   SampleRate: 16000
   MicPCM1: alsa_pcm:capture_1
   MicPCM2: alsa_pcm:capture_2
   MicPCM3: alsa_pcm:capture_3
   MicPCM4: alsa_pcm:capture_4
   Highpass: true
   Gain: 655000
   [/M]
   [M]
   Name: FSB1
   Type: FSB
   FrameSize: 128
   Channels: 2
   FilterLength: 128
   SampleRate: 16000
   StepSize: 0.2
   UseVad: false
   [/M]
   [M]
   Name: FSB2
   Type: FSB
   FrameSize: 128
   Channels: 2
   FilterLength: 128
   SampleRate: 16000
   StepSize: 0.2
   UseVad: false
   [/M]
   [M]
   Name: Router1
   Type: Router
   FrameSize: 128
   ChannelsIn: 1
   ChannelsOut: 2
   PortsIn: { 4 }
   PortsOut: { 2 2 }
   MapsIn: { 0 1 2 3 }
   MapsOut: { 3 2 0 1 }
   [/M]
   [M]
   Name: FSBDelay1
   Type: FSBDelay
   Channels: 2
   SampleRate: 16000
   FilterLength: 256
   MicDistance: 0.1
   Average: false
   Limit_Search: false
   Interpolate: true
   Upfac: 32
   Interp_dev: 3
   [/M]
   [M]
   Name: FSBDelay2
   Type: FSBDelay
   Channels: 2
   SampleRate: 16000
   FilterLength: 256
   MicDistance: 0.1
   Average: false
   Limit_Search: false
   Interpolate: true
   Upfac: 32
   Interp_dev: 3
   [/M]
   [M]
   Name: OutputFile
   Type: WriteFile
   FrameSize: 128
   RawData: true
   DataType: sample_float
   [/M]
   [M]
   Name: PositionEstimator
   Type: PositionEstimator
   MicArrays: 2
   MicPositionX: { 3.4 0 }
   MicPositionY: { 0 1.5 }
   MicLookDirectionX: { 0 1 }
   MicLookDirectionY: { 1 0 }
   RoomSizeX: 6.8
   RoomSizeY: 3.4
   StepSizeX: 0.1
   StepSizeY: 0.1
   [/M]
   [M]
   Name: Router2
   Type: Router
   FrameSize: 1
   ChannelsIn: 2
   ChannelsOut: 1
   PortsIn: { 1 1 }
   PortsOut: { 2 }
   MapsIn: { 0 1 }
   MapsOut: { 0 1 }
   [/M]
   [/MODULES]
   [CONNECTIONS]
   RMic       0   Router1   0
   Router1  0    FSB1      0
   Router1  1    FSB2      0
   FSB1      1    FSBDelay1     0
   FSB2   1       FSBDelay2     0
   FSBDelay1  0    Router2     0
   FSBDelay2  0    Router2     1
   Router2   0   PositionEstimator 0
   FSB1   0       OutputFile       0
   [/CONNECTIONS]
[/TASK]
```

9.3 Configuration file Echo.conf

```
[TASK]
TaskType: one_file
Input:    RMic(hw:0)
Read(/local_home/schmalen/Hammerfall/RAW/Musi.raw)
Output:                    Play()
OutputFile(/local_home/schmalen/Hammerfall/RAW/capture.raw)
   [MODULES]
   [M]
   Name: Play
   Type: JackPlay
   Channels: 4
   FrameSize: 1024
   SampleRate: 16000
   PlayPCM1: alsa_pcm:playback_1
   PlayPCM2: alsa_pcm:playback_2
   PlayPCM3: alsa_pcm:playback_3
   PlayPCM4: alsa_pcm:playback_4
   [/M]
   [M]
   Name: RMic
   Type: JackMic
   Channels: 2
   FrameSize: 1024
   SampleRate: 16000
   MicPCM1: alsa_pcm:capture_1
   MicPCM2: alsa_pcm:capture_2
   Highpass: true
   Gain: 450000
   [/M]
   [M]
   Name: Read
   Type: ReadFile
   FrameSize: 1024
   FrameShift: 1024
   FileType: RAW
   Swap: false
   DataType: sample_float
   [/M]
   [M]
   Name: OutputFile
   Type: WriteFile
   FrameSize: 2048
   RawData: true
   DataType: sample_float
   [/M]
    [M]
   Name: Router
   Type: Router
   FrameSize: 1024
   ChannelsIn: 1
   ChannelsOut: 3
   PortsIn: { 2 }
   PortsOut: { 1 1 1 }
   MapsIn: { 0 1 }
   MapsOut: { 0  0   1 }
   [/M]
    [M]
   Name: framing
   Type: Framing
   FrameSize: 1024
   FrameShift: 1024
   Offset: 0
   DataType:  sample_float
   [/M]
    [M]
   Name: AIC
   Type: AIC
   SampleRate: 16000
   Channels: 1
   FilterLength: 1024
   FrameSize: 1024
   StepSize: 0.1
   [/M]
    [M]
   Name: Router2
   Type: Router
   FrameSize: 1024
   ChannelsIn: 3
   ChannelsOut: 1
   PortsIn: { 1 1 1 }
   PortsOut: { 4 }
   MapsIn: { 0 1 2 }
   MapsOut: { 2  0  1   2 }
   [/M]

   [M]
   Name: Router3
   Type: Router
   FrameSize: 1024
   ChannelsIn: 2
   ChannelsOut: 1
   PortsIn: { 1 1 }
   PortsOut: { 2 }
   MapsIn: { 0  1 }
   MapsOut: { 0 1 }
   [/M]
    [/MODULES]

   [CONNECTIONS]
   RMic       0      Router       0
   Router 1          AIC          1
   Router 2          framing      0
   framing  0        AIC          0
    Router 0         Router2      0
    AIC     0        Router2      1
    Router2 0        Play         0
    Read   0         Router2      2
    Router      1    Router3      1
    AIC     0        Router3      0
    Router3     0    OutputFile 0
   [/CONNECTIONS]

[/TASK]
```

10 References

[AMI05]	http://www.amigo-project.org	Homepage Amigo Project
[RME05]	http://rme-audio.de	RME Homepage
[YAH05]	http://www.yamahaproaudio.com	Yamahe Homepage
[JAC05]	http://jackit.sourceforge.net	Jack Audio Connection Kit Homepage
[SUS05]	http://www.suse.de	SuSe Linux Homepage
[RLT05]	http://sourceforge.net/projects/realtime-lsm	Realtime Module for Kernel
[AKG05]	http://www.akg.com	AKG Homepage
[NT05]	http://nt.uni-paderborn.de	Internetseite Fachgebiet Nachrichtentechnik

[Hay99] S.Haykin. Adaptive Filter Theory; Prentice-Hall Inc., 1991

[Hof00] G.Moschytz; M. Hofbauer. Adaptive Filter; Springer Verlag,2000

[Shy92] J. Shynk. Frequency Domain and Multirate Adaptive Filtering; IEEE SP 1992