Institut für Maschinelle Sprachverarbeitung

Seminar: Speech Signal Processing and Speech Enhancement

MSc. Computational Linguistics – SuSe 19

Modelling Echo Cancellation: Speech enhancement with an LMS adaptive filter

Author

W. Tessadri

Supervisor

Dr. W. Wokurek

April 9, 2020

1 Introduction

As Smith (1999: 3) [9] writes in his coherent guide, Digital Signal Processing (DSP) "has produced revolutionary changes" in very diverse scientific and industrial fields. Starting from the 1960s/1970s and with computer development as driving force, DSP led to a technological breakthrough in areas such as space research, medicine and audio processing. Smith (1999: 1–10) [9] cites various examples for its vast technological applicability: In space research and medical applications, for example, DSP is used to improve acquisition techniques. As images in space or during MRI often must be taken under unfavourable conditions DSP can greatly improve image quality in post-processing by removing noisy features, i.e. optimizing the Signal-to-Noise Ratio (SNR). When it comes to audio processing and, more specifically, applications relevant to computational linguists, DSP has played an essential role in the development of speech recognition and generation systems. An example the author names, is the rise of vocal tract simulators for speech generation. In this approach, speech is produced by computational systems by simulating the resonance characteristics of the vocal tract. In this way speech could be produced much more flexibly as when using human recorded segments of speech stored in the device. Besides the many technological advantages and the need for digital data for computer processing another reason for the tremendous success of DSP was the extraordinary performance of digital filters. As Smith (1999: 262) [9] puts it, "digital filters can achieve thousands of times better performance than analog filters". While the performance of analog filters is limited by their hardware, by the electronic restrictions of resistors, capacitors and inductors, digital filters do not face these problems. They can be implemented by computer programs and are, thus, cheaper, more flexible and easier to design. While a class on Speech Signal Processing (SSP) and Speech Enhancement (SE), thus, provides valuable insights into an essential concept of NLP, a project work on digital filters opens the possibility to take a closer look at one of the most influential technologies in this field. The present seminar paper aims at giving a very brief overview of the most important concepts of digital filters and explores, in a second step, the properties of a specific type of filter: the LMS adaptive filter. This exploration is performed under the assumption of a simplified echo-cancellation set-up, a frequent use-case of LMS adaptive filters. To achieve the mentioned goals the chapters of the present paper cover the following topics: Chapter 2 describes some key terms for digital filters, fundamental for a thorough understanding of up-coming chapters and gives a short overview of filter types in general. Chapter 3, again, covers the general characteristics of adaptive filters. Chapter 4 provides the necessary knowledge about the LMS algorithm, followed up by the central part of the present work: Chapter 5 describes the project carried out for this seminar, presenting the data used to model an echo cancellation scenario, the LMS implementation and a performance analysis by means of three different scenarios.

2 Digital filters

2.1 Key concepts

In the domain of SSP, relevant for the present paper, we are dealing with signals in time domain, i.e. a function of amplitude changing over time. Acoustic signals, though, are by nature analog, i.e. continuous. This is no problem for the human hearing system as the eardrum converts the continuously incoming sound pressure fluctuations into a corresponding continuous signal transmitted to the cochlea via the auditory ossicles. Microphones make use of a very similar technique relying on a vibrating membrane which converts sound waves into electrical impulses. What is no problem for humans and microphones, though, is a major challenge for computers. They require a discrete signal as they cannot store infinitely precise numerical data. For this reason, to be processable with a computer, the analog audio signal is converted to a digital signal by an Analog-to-digital converter (ADC). An ADC samples a signal in time domain with a certain interval. The sampling rate, given in Hertz (Hz), is then computed by dividing one second through the duration of the sampling interval. A recurrent sampling rate in SSP is 44100 Hz, which can capture spectral frequencies up to 22050 Hz (Nyquist frequency). For each point in time, i.e. for each sample, the amplitude value of the signal at that point is stored. How fine-grained the amplitude value can be represented depends on the number of bits used. With 8 bits we can store 256 different amplitude values, with 16 bits already 65536 values. This process is called quantization (see Johnson (2012: 55) [4] for more details on quantization). A signal sampled and quantized in this way represents the input for digital filters. Given an adequate discrete signal as input and a digital filter, the filter changes the signal in a certain manner, leading to an altered version of the signal as output. The way signals are changed by a filter is called the impulse response or filter kernel. As the name suggests, the easiest way to find out the impulse response of a filter is to give the filter a so-called unit impulse as input. A unit impulse is a normalized impulse with a value of one at sample zero and zeros for the remaining samples. Another possibility to characterize the response of a filter to a certain signal is by specifying the frequency response. The frequency response is found by applying a Discrete Fourier Transform to the impulse response and specifies how the frequencies of an input signal are altered. To this point, though, the question has not been answered, how precisely a digital filter changes an incoming signal: Digital filters are implemented by applying a mathematical method called convolution. To understand this concept, it is necessary to know that the response of a filter depends on its coefficients, simply put, a set of weights. As Smith (1999: 95) [4] defines it, convolution is the "mathematical operation, where each value in the output is expressed as the sum of values in the input multiplied by a set of weighting coefficients". To perform this operation a filter has a tapped delay line with at least one tap. By means of these taps a filter can store/delay the N most recent input samples. One output sample results, then, from multiplying those N input sample by the corresponding N filter weights and summing up the products. Consequently, each output sample depends on N input samples. Even though this can only provide a very superficial insight into the working and structure of digital filters, the concepts and terminology established so far provide a good starting point to elaborate more advanced concepts, such as types of filters, described in the next chapter.

2.2 Filter types

Depending on which criteria are applied there exist several classifications of filters. Relying on the classification in Smith (1999: 261 ff.) [4] two classifications have been chosen for this seminar paper that are relevant for a better understanding of the LMS filter and adaptive filters in general.

2.2.1 Classification following implementation

When it comes to implementation there are two main types of filters: Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) filters (disregarding Cascaded-Integrator-Comb filters). FIR filters are implemented by convolution. They produce the output by weighting and summing up several points from the input. Consequently, and as the name states, the response of a FIR filter will be zero after a finite number of samples. A canonical example for a FIR filter is the Moving Average filter, where an output sample is computed by taking the average over N input samples. Smith (1999: 277) [4] gives the example of a five point moving average filter, in which point 80 would be computed by the following formula, taking into account the five most recent samples:

$$y[80] = \frac{x[80] + x[81] + x[82] + x[83] + x[84]}{5} \tag{1}$$

The following figure outlines a second order moving average filter. $x[n]$ is the input, $y[n]$ the output. There are two delay taps z^{-1} as well as three filter coefficients, which, in case of the average filter are all equal (taken from https://en.wikipedia.org/wiki/Finite_impulse_response [12]).

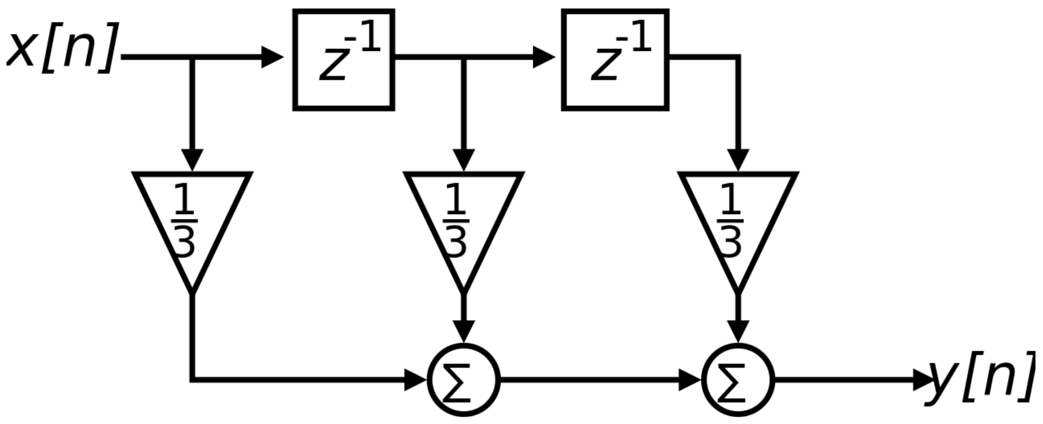


Figure 1: Second order Moving Average Filter

By contrast, IIR filters are implemented by recursion. Such recursive filters can be seen as an extension of filters using mere convolution: IIR filters like FIR filters compute the output signal by multiplying N input samples with the filter coefficients and taking the sum. The crucial difference, though, is that this is not the only basis for the output: Besides N input samples an IIR also uses the N-1 previously computed output samples to compute the next output. This is done by using a feedback loop and so-called recursive coefficients. Thus, to compute the output sample $y[n]$ the following formula would be applied:

$$\begin{aligned} y[n] = a_0x[n] &+ a_1x[n-1] + a_2x[n-2] + a_3x[n-3] + \cdots \\ &+ b_1y[n-1] + b_2y[n-2] + b_3y[n-3] + \cdots \end{aligned} \tag{2}$$

As each output is always re-used to compute the next output sample, the impulse response of IIR filters continues even if no input samples are left. However, over time the response decays to an insignificant level. The following figure is an illustration of a simple IIR filter with the input coefficients $a0 - a2$ and recursive coefficients $b1$ and $b2$ (taken from https://www.researchgate.net /figure/Block-Diagram-of-an-IIR-Filter_fig2_307577369 [7]):

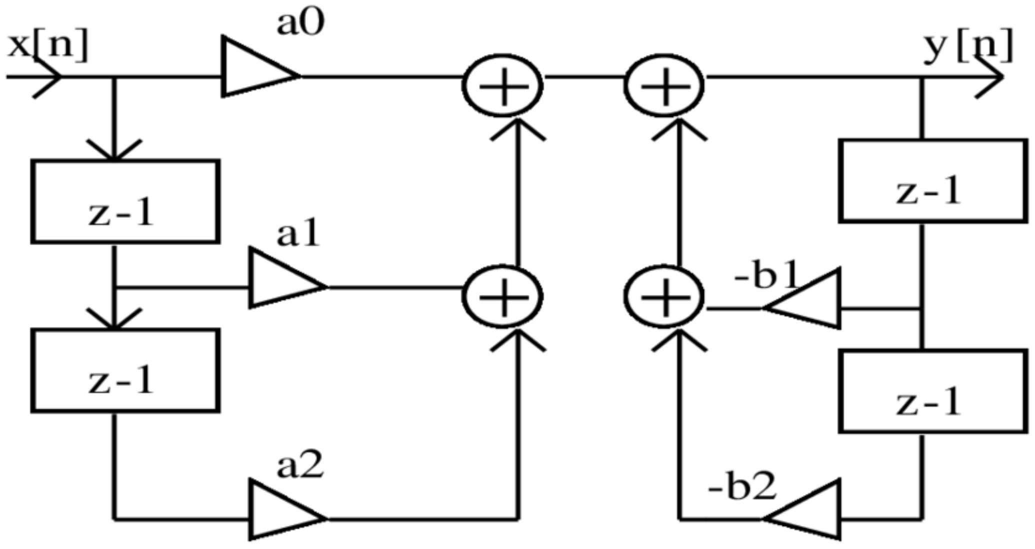


Figure 2: Second order Moving Average Filter

Besides implementation another way to classify filters is by their frequency response.

2.2.2 Classification following frequency response

To speak about the frequency response of filters, another piece of terminology has to be introduced (Johnson 2012: 19–22) [4]. Filters with different frequency responses block certain frequencies while allowing others. Frequencies lying in the so-called reject band are blocked, frequencies in the pass band are let through. The boundary between reject band and pass band is not infinitely steep, but the transition normally is gradual (slope characteristics are another method to classify filters). Regarding pass band and reject band location the four main types of filters are:

1. Band pass filter: Only a certain band of frequencies is allowed. Frequencies below or above this band are rejected.
2. Stop band filter: Complement of band pass filters. Only a certain band of frequencies is blocked, the remaining frequencies can pass.
3. Low-pass filter: Lower frequencies pass, higher frequencies are blocked.

4. High-pass filter: Complement of low pass filters. Higher frequencies pass, lower frequencies are blocked.

As an example, the following figure illustrates the working of a low-pass filter:

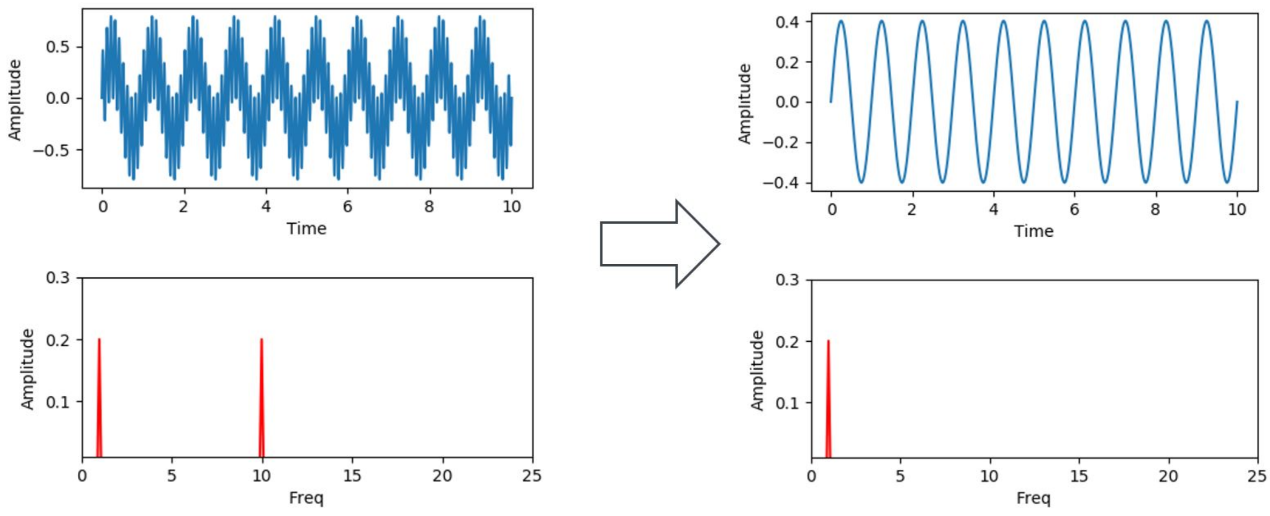


Figure 3: Effect of a low-pass filter

The signal represented on the left consists of two overlapping sine waves. As can be seen in the spectrum the frequencies of these sine waves are 1 Hz and 10 Hz. Assuming that the 10 Hz frequency is undesirable, we can use a properly designed low-pass filter to filter out the noisy component. The right side of the figure represents the signal after application of the low-pass filter. The 10 Hz component has been filtered out, resulting in a perfect 1 Hz sine wave in the oscillogram. Besides illustrating the aim of filters with a certain frequency response this simple example should also make clear the limitations of such filters: In this scenario the noisy and the desired components were clearly distinguishable. Very often this is not the case. It may for example happen that the noise-frequency varies over time, that it is not known which frequencies must be filtered. Moreover, the bands of signal and noise may also overlap such that it is difficult to tell which frequencies are constituting the noise and which the signal. The solution to this problem lies in the design of the filter coefficients. Up to this point the filters described in this sub-chapter were assumed to have fixed, time-invariant coefficients. To solve the problem of adaptability to the input, the filter coefficients must be updated to be able to react to the input. This is the idea behind adaptive filters, which are the topic of the next chapter.

3 Adaptive Filters

3.1 General description

The best way to describe the design of adaptive filters is by means of the following illustration, taken from Diniz (2008: 3) [2]:

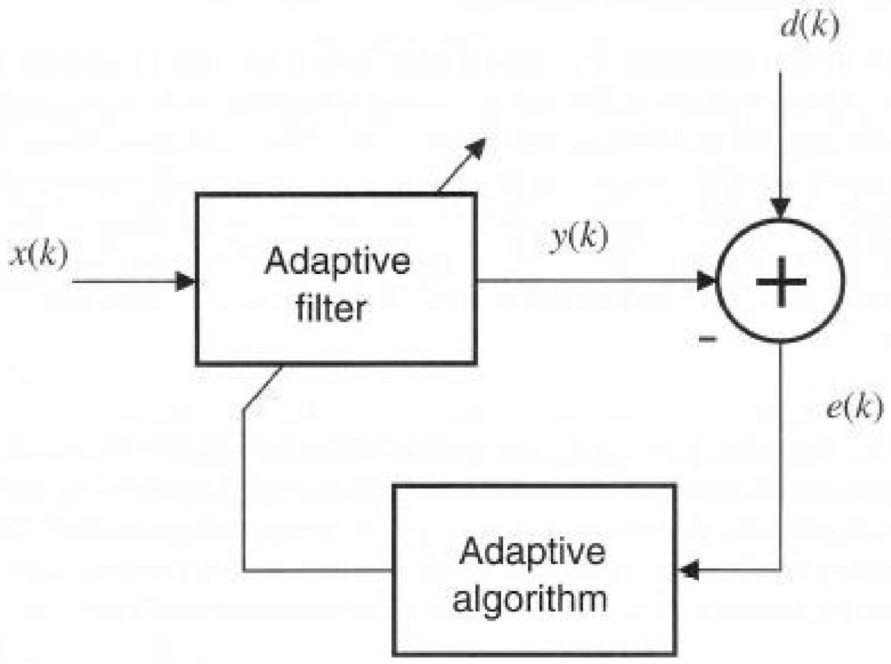


Figure 4: Structure of an adaptive filter

The first difference to filters with time-invariant coefficients, which should become evident when looking at figure 4 is the number of inputs. Instead of one input, adaptive filters receive two inputs. In this case $x(k)$ represents the input signal and $d(k)$ the desired signal, while k denotes the number of iterations. $y(k)$ is the signal that results if we apply our filter to the input signal. The aim is to approximate $d(k)$ as good as possible. How good our estimation of the target signal is, is quantized by the error $e(k)$. This error is computed by subtracting $y(k)$ from $d(k)$. The result is used in each iteration to update the filter coefficients via the adaptive algorithm.

3.2 Adaptive filter specifications

Based on this short description and referring to the discussion in Diniz (2008: 3–4) [2] there are three items, which have to be specified when designing an adaptive filter: application, adaptive-filter structure and algorithm. In the following those three specifications are described for the present project:

1. Application: The first thing to consider is the aim of the filter – what he is designed for. As indicated in the introduction, the project described in this seminar paper aimed to model adaptive filtering echo cancellation. When taking this set-up as a basis the preceding figure would change as follows (taken from https://www.researchgate.net/figure/Block-diagram-of-an-adaptive-filter-as-a-noise-canceller_fig4_267150925) [6]:

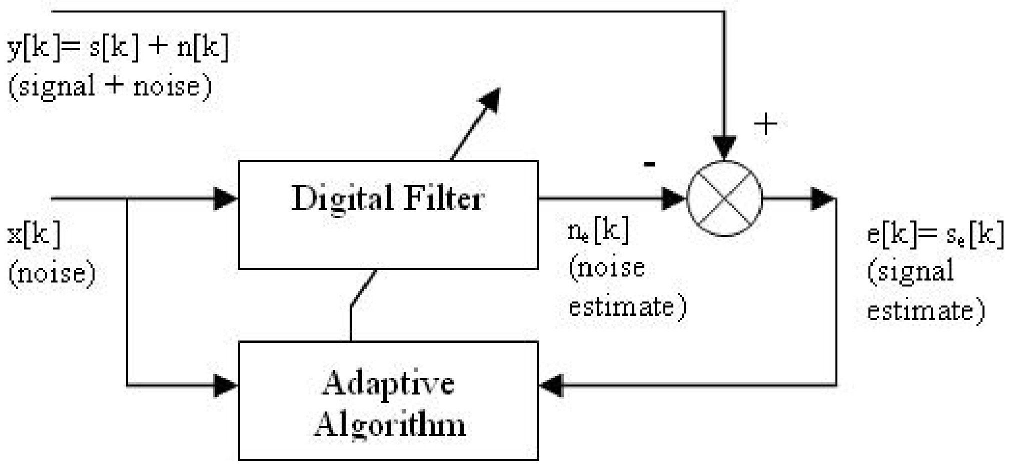


Figure 5: Structure of an adaptive filter as noise canceller

As can be seen, the main difference between figure 4 and figure 5 lies in the input specifications. In an echo or noise cancellation application we have $y(k)$, a noisy signal, on the one hand, and the reference signal $x(k)$, a signal correlated with the noise, but not with the desired signal component, on the other hand. The error is computed by calculating a noise estimate $n_e(k)$ using the digital filter and subtracting this estimate from the noisy signal. The resulting error $e(k)$ is equal to the signal estimate $s_e(k)$, computed by:

$$s_e(\mathrm{k}) = y(\mathrm{k}) - n_e(\mathrm{k}) = s(\mathrm{k}) + n(\mathrm{k}) - n_e(\mathrm{k}) \tag{3}$$

Consequently, to optimize the SNR and get the best estimate of the desired signal, the noise estimate must be optimized. This corresponds to a minimization of the error function.

2. Besides the aim of the filter also its adaptive-filter structure, the design of the actual filter structure is relevant. The decision at this point is binary in nature: the filter can be designed as FIR or as IIR filter. This depends on the application, which the filter is implemented for. For the present project a FIR filter design was chosen. Hence, output samples are computed solely by using input samples and not by recursively using output samples as well.

3. The last item to consider when designing an adaptive filter is also the central one: the algorithm chosen to update the filter coefficients. As Diniz (2008: 4) [2] writes, the algorithm is determined by defining the search method (or minimization algorithm), the objective function, and the error signal nature.

The next chapter will characterize the described properties for the Least Mean Square (LMS) algorithm, which stands in the centre of the present project.

4 The Least Mean Square algorithm

4.1 Specifications

The Least Mean Square (short for Least Mean Square Error) algorithm was first introduced by Widrow & Hoff (1960) [10]. As already described in chapter 3.1 and 3.2 it requires two inputs. In a noise cancellation task the two inputs would be the corrupted signal and a signal correlated to the noise in the corrupted signal: The aim of the algorithm, when used in such a noise cancellation task, is to find the optimal filter coefficients to filter out the noisy component as good as possible. To put it in more technical terms: The LMS seeks to reduce the sum of the squared errors, which result from subtracting the noise estimate from the corrupted signal:

$$J(\boldsymbol{w}) = \sum_{n=0}^{N-1} e_n^2 = \sum_{n=0}^{N-1} \left(t_n - \boldsymbol{x}_n^T \boldsymbol{w} \right)^2 \tag{4}$$

The sum of the squared errors here is conceptualized as the objective function $J(w)$. Hence, to optimize the SNR this objective function must to be minimized. The LMS now uses gradient descent as a searching method to approximate the optimal solution. This results in the following formula for updating the filter coefficients:

$$w_{l+1} = w_l + \mu \left(t_l - x_l^T w_l \right) x_l \tag{5}$$

The formula states that the filter coefficients for the next iteration are found by first multiplying the error of the present iteration with the input samples coming from the noise and adding the result, which in fact represents the gradient, to the present set of filter coefficients. Before this addition the gradient moreover is multiplied by a fixed step-size or learning rate μ. The design of an LMS filter is outlined in figure 6 taken from Diniz (2008: 79) [2]

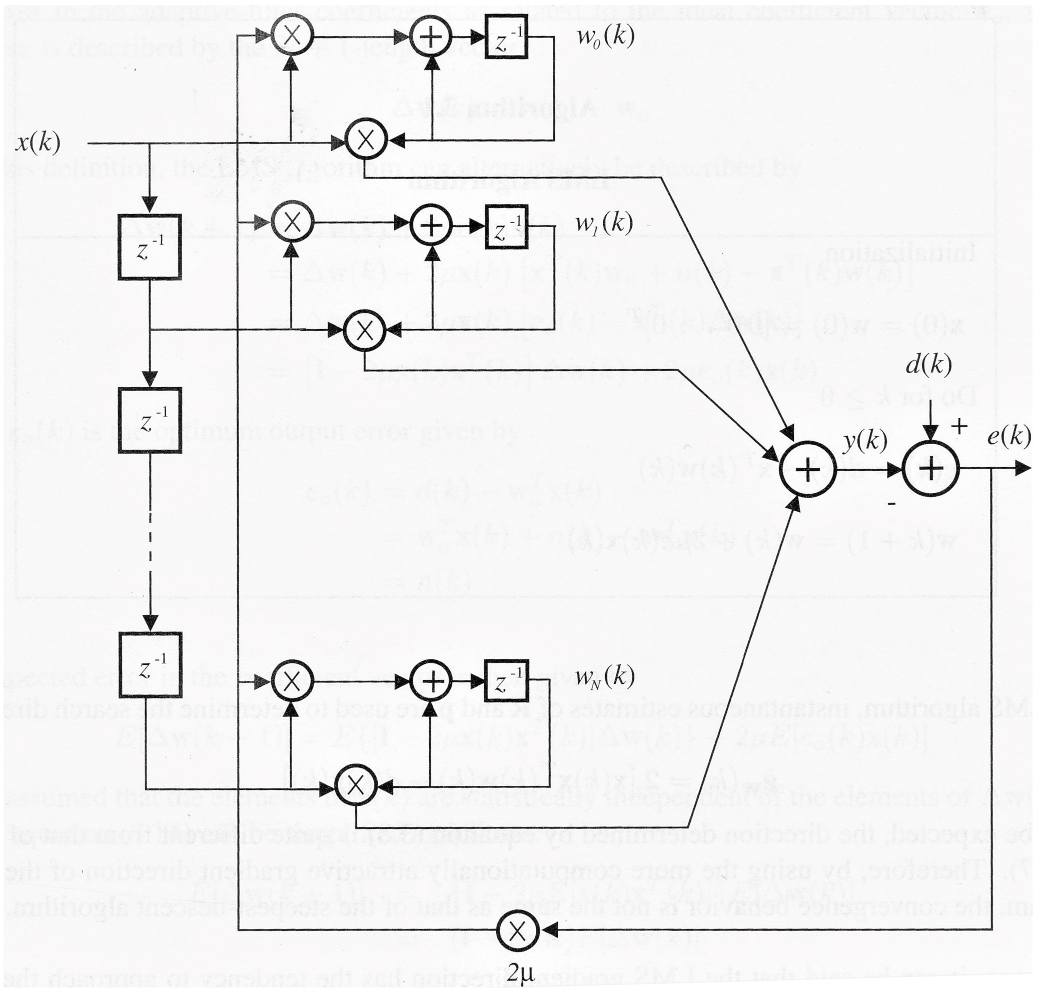


Figure 6: Structure of an LMS adaptive filter

As in previous examples $x(k)$ is the signal correlated with the noise in our corrupted signal $d(k)$. The elements labelled with z^{-1} are the filter taps, delaying elements, while $w0$ to wN are the coefficients at iteration k. When reading the graph from left to right the following happens: The incoming input samples first reach position 1 and 3. At position 1 they are multiplied with the product of error and learning rate, formed at position 6. At position 3 the input sample is multiplied with its corresponding coefficient from the previous iteration. At position 2 the central step of the algorithm happens: By taking the product computed at position 1 and adding it to the filter coefficients from the previous iteration, the filter coefficients are updated for the next iteration. By using a delay tap the computed coefficients are stored for the filter update at position 2 and the multiplication of coefficient and input sample at position 3 happening in the next iteration. The values computed at position 3 are, then, summed up at position 4. Thus, at position 4 the output of the filter is computed, which in a noise cancellation scenario corresponds to the noise estimate ($\rightarrow y(k) = n_e(k)$). Finally, at position 5 the error is calculated by subtracting the noise estimate from the incoming corrupted signal and with this new error we enter the next iteration.

While this chapter gave a short outline of the specifications and design of an LMS filter, it did not answer the question, how the presented updating algorithm is derived mathematically. The next chapter will bridge this gap and relate the LMS algorithm to a "prominent solution" (Kammeyer & Kroschel 2018: 242) [5] for noise cancelling – the Wiener filter, published by Norbert Wiener in 1949 (Wiener 1949) [11].

4.2 Mathematical derivation

This chapter treating the mathematical derivation of the LMS algorithm is based on the discussion in Do (2017) [3]. As already mentioned in the previous chapters, to come to the optimal filtering result, the task of an LMS filter is to minimize the error, calculated by subtracting the noise estimate from the corrupted signal. Both the noise estimate, resulting by convolving the reference signal with the filter coefficients, as well as the corrupted signal can be conceptualised as vectors, here represented in matrix form:

$$\begin{bmatrix} e_0 \\ e_1 \\ e_2 \\ \vdots \\ e_{N-1} \end{bmatrix} = \underbrace{\begin{bmatrix} t_0 \\ t_1 \\ t_2 \\ \vdots \\ t_{N-1} \end{bmatrix}}_{t} - \underbrace{\begin{bmatrix} x_0^T \\ x_1^T \\ x_2^T \\ \vdots \\ x_{N-1}^T \end{bmatrix}}_{X^T} \underbrace{\begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_K \end{bmatrix}}_{w} \tag{6}$$

To accomplish the minimization of the objective function in (4) (sum of the squared errors) we first take the gradient. With w as variable the partial derivation of (4) is simply x. Using the chain rule this gives us the following formula for the gradient of the objective function:

$$\begin{aligned} \nabla J(\boldsymbol{w}) &= \sum_{n=0}^{N-1} 2\left(t_n - \boldsymbol{x}_n^T \boldsymbol{w}\right) \boldsymbol{x}_n \\ &= 2\sum_{n=0}^{N-1} e_n \boldsymbol{x}_n \\ &= 2\boldsymbol{X}\boldsymbol{e} \\ &= 2\boldsymbol{X}\left(\boldsymbol{t} - \boldsymbol{X}^T \boldsymbol{w}\right) \end{aligned} \tag{7}$$

Looking at the last term, the gradient of the objective function, thus, is computed by multiplying the input vector with two and multiplying the resulting vector with the error vector. The error vector, again, is calculated by subtracting the corrupted signal vector from the noise estimate vector. By setting the gradient to zero and dividing by two, this last term can be transformed to:

$$\boldsymbol{X}\boldsymbol{X}^T \boldsymbol{w} = \boldsymbol{X}\boldsymbol{t} \tag{8}$$

This expression can be rearranged, giving the formula:

$$\boldsymbol{w} = \left(\boldsymbol{X}\boldsymbol{X}^T\right)^{-1} \boldsymbol{X}\boldsymbol{t} \tag{9}$$

The formula in (9) is the Wiener solution to find the optimal coefficients for noise cancellation and states the following: To find the optimal solution multiply the inverse autocorrelation matrix of the input signal with the cross-correlation matrix of input and target signal. But why does setting the gradient to zero lead to the optimal solution? To understand this, we can model the present optimization problem geometrically as a vector space. The basis vectors spanning the vector space, i.e. it's dimensions, are the single filter coefficients as well as the output of the objective function. Taking the objective function and the first two filter coefficients as reference would give a three dimensional vector space (taken from Kammeyer & Kroschel 2018: 245) [5]:

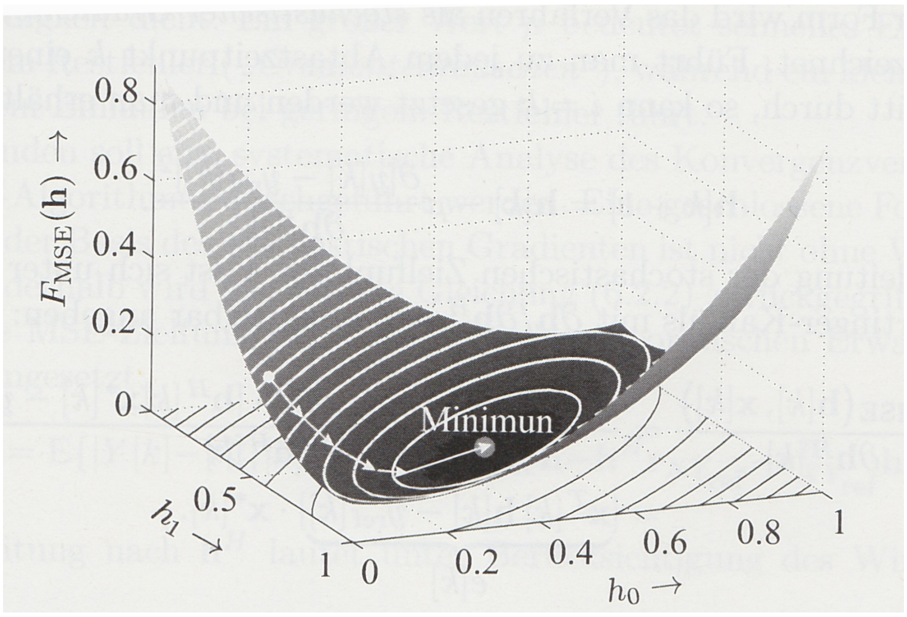


Figure 7: Principle of the gradient algorithm

The plane painted in black is the co-domain of our objective function, i.e. all values the function can take on if we feed in different combinations of the filter coefficient values. As we want to minimize the objective function, our task is to find the minimum value for $J(w)$ in this vector space. And here is where the gradient comes into play: The gradient quantizes the slope of a function at a certain point. A useful feature of the slope is, that the slope is zero at the minimum of a curve or plane. Consequently, by setting the gradient to zero and applying the above transformations, we come to the filter coefficient values, i.e. dimensions, which will minimize $J(w)$. The Wiener solution, thus, enables us to find the optimal filter coefficients using the corrupted signal and a signal correlated with the noise. The problem with this approach, though, is that in a noise cancellation task, we hardly ever have both signals fully available. Instead, an adaptive filter must be readjusted iteratively as new input samples come in. The task is a predictive one: Filter out the noise in still coming samples based on the noise estimation in previous samples. To accomplish this, a method called gradient descent is used, which tries to approximate the optimal Wiener solution. With this method we "iteratively refine an estimate of w in the steepest (i.e. negative gradient) direction" (Do 2017: 3) [3], i.e. in each iteration we recalculate the gradient and use it to update the filter coefficients. Referring to the formula for the gradient derived in (7) and introducing the learning rate μ, which controls the convergence behaviour of our algorithm, the following formula results:

$$\begin{aligned} \boldsymbol{w}_{l+1} &= \boldsymbol{w}_l + \mu \boldsymbol{X} \left(\boldsymbol{t} - \boldsymbol{X}^T \boldsymbol{w}_l \right) \\ &= \boldsymbol{w}_l + \boldsymbol{\mu} \sum_{n=0}^{N-1} \left(t_n - \boldsymbol{x}_n^T \boldsymbol{w}_l \right) \boldsymbol{x}_n \end{aligned} \tag{10}$$

This formula can be further simplified by replacing the gradient sum over all samples by only the last sample, which leads us to the final formula for the LMS, given in (5).

4.3 The learning rate μ and the NLMS

An important part of the LMS, which up to this point did not get a lot of attention, is the learning rate μ. As already mentioned, it controls the convergence behaviour of the objective function: The higher μ is, the faster the algorithm will converge to the optimal solution. However, if μ is too high, the objective function will diverge. The lower μ is, the closer it will come to the optimal solution, but the slower the convergence speed will be. Choosing the optimal μ, thus, is an important part in implementing an LMS. The problem with the standard LMS is that the choice of the learning rate depends on the power of the input. This drawback is solved by the NLMS, the Normalized Least Mean Square algorithm. The advantage of the NLMS is that it is normalized by power: This is accomplished by dividing trough the autocorrelated reference signal plus a regularization parameter to avoid zero division (Wokurek 2019: 98) [13]. This leads to the following formula:

$$\mathbf{h}+ = \mu \frac{1}{\mathbf{x}^* \mathbf{x} + \Delta} \mathbf{x} e^* \tag{11}$$

Due to this normalization the convergence behaviour of our objective function is stable as long as $0 < \mu < 2$. For this reason, for the present project instead of the LMS the NLMS was used.

The preceding chapters tried to prepare the ground for a better understanding of the project work carried out. The following chapter will describe the idea, implementation and results of the project itself.

5 Project description

5.1 Idea and data collection

As already mentioned in the introduction one of the most frequent use cases of LMS adaptive filters is noise cancelling. As Deller, JR. et al. (2000: 528–529) [1] write, "adaptive noise cancelling (ANC) has been applied successfully to a number of problems that include speech, aspects of electrocardiography, elimination of periodic interference, elimination of echoes on long-distance telephone transmission lines, and adaptive antenna theory". As in SSP we deal with audio signals, the idea which stood at the beginning of this project was to implement an LMS adaptive filter in a simplified echo cancellation scenario. As already mentioned in the above quotation, this could be useful in telecommunication situations. However, while echoes in long-distance transmission lines can be caused either by circuit imperfections and impedance mismatches between telephone circuits or by acoustic echoes, this project assumes an acoustic echo cancellation scenario. Such a scenario is best illustrated through the following figure (taken from http://zone.ni.com/reference/en-XX/help/372357B-01/lvaftconcepts/aft_aec/ [8]):

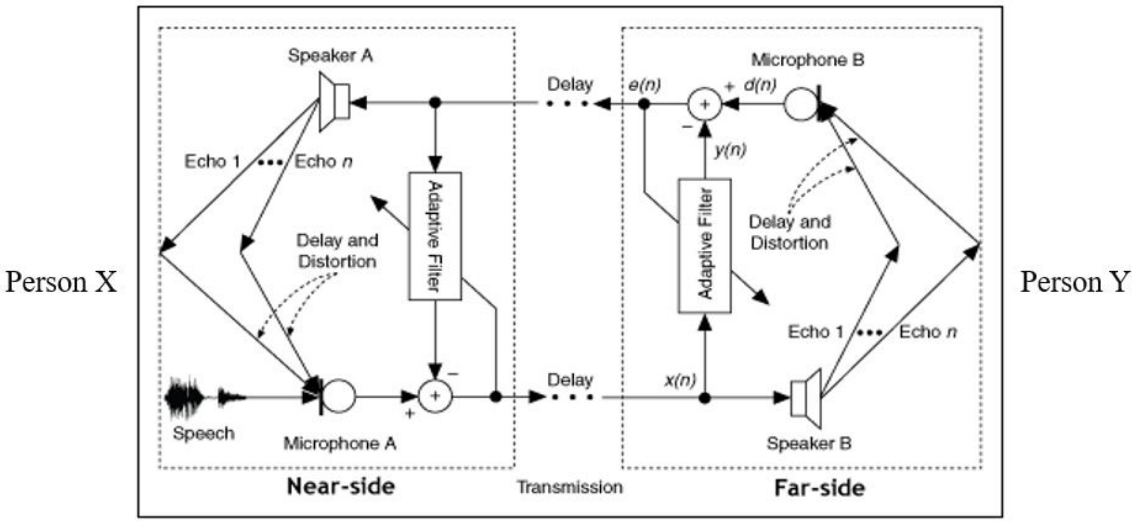


Figure 8: Adaptive noise cancellation scenario

In this figure two persons, person X and person Y, are involved in a telephone conversation. Person Y speaks into microphone B, while X speaks into microphone A. Both participants listen to each other's speech via loudspeaker. In an ideal set-up, e.g. if both participants would be in a totally sound-absorbent environment, they would only hear the other participant's speech through their loudspeaker. The problem, though, is that in a natural environment, everything, which comes out of the loudspeaker, can be reflected, leading to a delayed and distorted version of the loudspeaker's output, that travels back to the sender. If, for example, person Y is talking, the output of speaker A leads to an echo in the surroundings of person X. This echo is registered by microphone A and travels back to person Y, which is why person Y, can hear his echo. As illustrated in the above figure as well, to cancel out this echo an adaptive filter may be used. The input for the adaptive filter is again twofold: a signal used for noise estimation and the signal corrupted by the noise. In the present scenario the signal used for estimating the noise is the signal issuing from the loudspeaker's, and the noise itself is the echo of this signal. To give a concrete example: If person Y is talking, his speech in the transmission line is used for noise estimation. The noise is his voice reflected by the surroundings of X. If person X is speaking as well, this echo is intermingled with X's speech, which leads to a corrupted signal with speech and echo components, that can be improved by the interposed adaptive filter before travelling back to Y.

The original project idea was to model this set-up by assuming the following: The signal used to estimate the noise, transmitted by person Y should be represented by the melody of a song. The noise or echo should be modelled by this instrumental signal delayed by one sample and reduced by half the amplitude. This noise signal should be combined with the vocals of a song giving the corrupted signal, ready to be filtered. Thus, two files were needed: One file with only the musical/instrumental component to create the noise estimate as well as the actual noise and one file with the vocals the noise could be mixed with. Moreover, I searched for uncompressed files to exclude distortions of the results by the compression process. After an unsuccessful one-week online searching period, I contacted a sound technician I knew from a previous project, asking him for help with the collection of appropriate data. He made the proposal to use the soundtrack of a documentary he had recently worked on and so I got two files from him: a file containing the narrator track and a file containing the film music, both in high, uncompressed quality. Both files were 21.243 seconds long and were sampled with a 48000 Hz sampling rate in 32-bit quality. The files had a size of 5.83 MB containing 1019661 samples. Moreover, the audio signals were represented in a two-channel stereo format. As for the purposes of the present project a mono format was sufficient, as first step the two channels were merged into one channel. For this conversion into mono format the program Audacity was used. The converted music/instrumental file was, then, used to create the supposed noise and compute the noise estimate, while the converted narrator signal, combined with a version of the music delayed by one sample and with halved amplitude, modelled the desired, but echo-corrupted signal.

After the successful completion of the data collection and pre-processing stage, I could start to implement the LMS adaptive filter, which the data could be applied to.

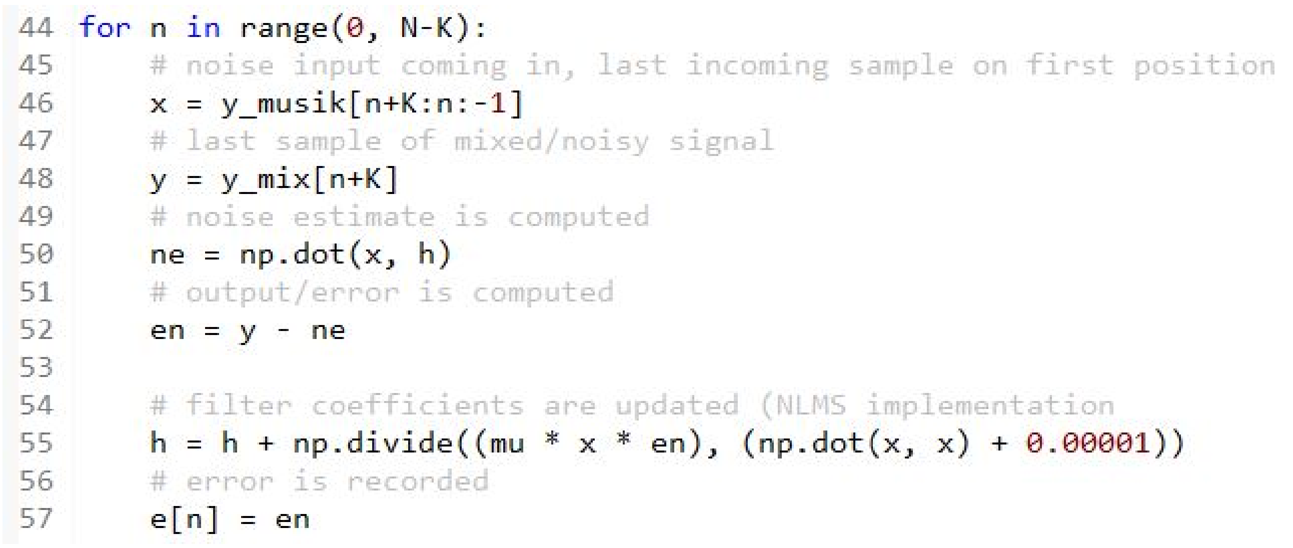
5.2 Implementation

The code to implement the LMS filter computationally was based on the code presented in Do (2017: 4) [3] and the MATLAB implementation treated in the seminar. The program was implemented in Python 3.7 for 32-bit systems. For processing the data various libraries were used. The most important ones, though, were *librosa*, *numpy* and *matplotlib*:

- *librosa* reads in audio files and converts them to numpy arrays containing amplitude values.
- *numpy* was used to process the input vectors/arrays.

- *matplotlib* was used to represent the results graphically.

As a complete description of the code written to implement and display the LMS filter would go beyond the scope of this paper, in the following only the essential part of the LMS algorithm is depicted:



```python
for n in range(0, N-K):
    # noise input coming in, last incoming sample on first position
    x = y_musik[n+K:n:-1]
    # last sample of mixed/noisy signal
    y = y_mix[n+K]
    # noise estimate is computed
    ne = np.dot(x, h)
    # output/error is computed
    en = y - ne

    # filter coefficients are updated (NLMS implementation
    h = h + np.divide((mu * x * en), (np.dot(x, x) + 0.00001))
    # error is recorded
    e[n] = en
```

Figure 9: Python code for the LMS filter

The code follows the logic of the (N)LMS adaptive filter described in the previous chapters:

- n represents the iteration counter.
- N is the length of the input vectors, which in this case are derived from the music and narrator files.
- K specifies the number of filter coefficients used for implementing the FIR filter, while h is a numpy array with the actual coefficient values.
- x and y are the incoming samples from the input. x are samples from the signal for noise estimation, while y is the most recent sample of the corrupted signal. In case of x the most recent sample is always attached to the beginning of the array, while the oldest sample is excluded.
- n_e is the noise estimate as computed by convolving x with the filter coefficients in h. Here, convolution corresponds to taking the inner or dot product.

- en is the error of the current iteration, calculated by subtracting the noise estimate n_e from the corrupted signal y.
- After error computation at line 55 the filter coefficients are updated using the NLMS.
- To be able to plot the development of the error over time, the error at the current iteration is stored in the numpy array e collecting all errors.

The other sections of the code mainly concern the loading and pre-processing of the data as well as the plotting of the results and are not reported here. The LMS filter in the case of the present project was not implemented as a function. Instead the same code was re-used in various Python modules. These modules should test the behaviour of the filter in various scenarios. The next chapter describes the results of the project by presenting three of these scenarios.

5.3 Results and Analysis

The three scenarios presented in this chapter all focus on different aspects of the LMS. Given the described data the first scenario sheds light on the convergence behaviour of the implemented adaptive algorithm. The second scenario will analyse the effect of the LMS on noise frequencies. Finally, the third scenario will analyse the filtering results of the original set-up, described in chapter 5.1.

5.3.1 Scenario 1: Convergence

The first scenario should test and visualize the convergence behaviour of the LMS implementation. For this reason, a set-up was assumed were only the music and a delayed version of the music were used. Referring to figure 8 the music represents the output at loudspeaker A, while a modified version of this same file represented the noise. As described above, the modification involved a signal delay by one sample and a reduction of the amplitude by half. As no corrupted signal of speech (narrator) and echo, but only the echo was given in this scenario, ideally, the error would converge to zero, i.e. the artificial echo would be cancelled out totally by the implemented filter, resulting in silence as desired signal. The following figure shows the obtained results:

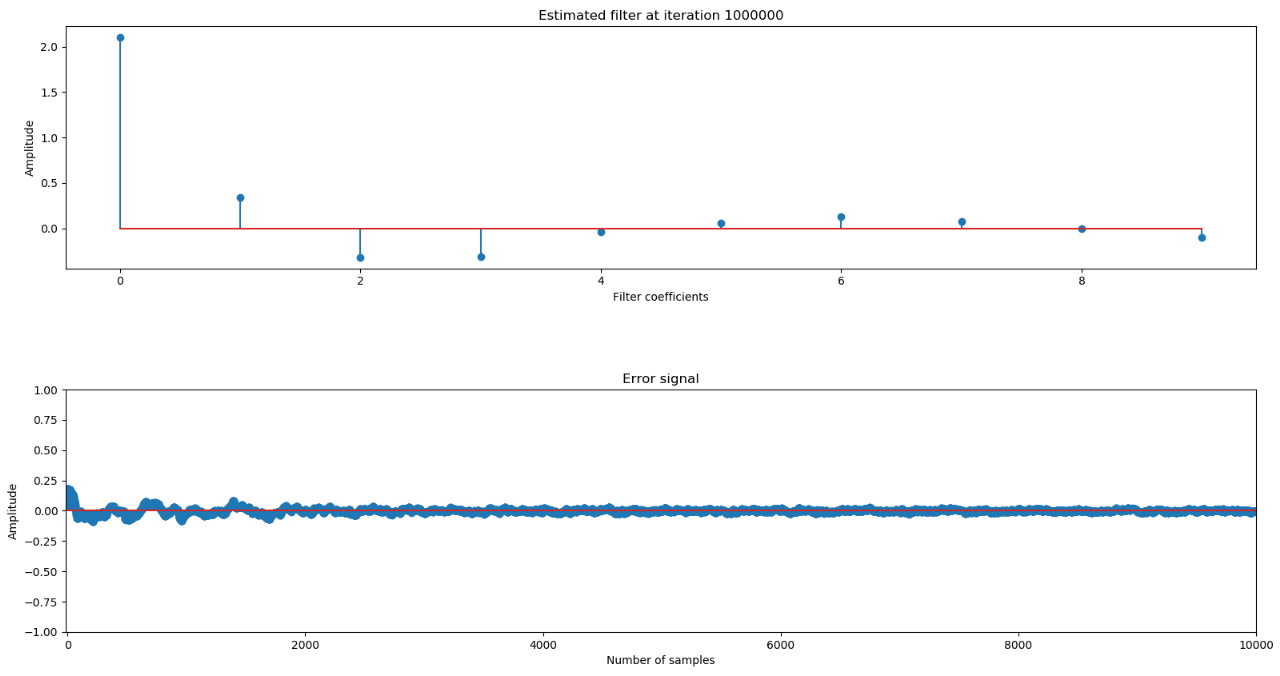


Figure 10: Estimated filter and error development

At the upper half of figure 10 the shape of the filter coefficients is displayed after one million samples were read in. For a better overview in this case 10 filter coefficients were chosen, but the program was also tested with higher and smaller numbers of filter coefficients. The lower half of the above figure depicts the development of the error signal during the first 10000 samples. As can be seen, the algorithm displays the expected behaviour: During the first 2000 samples the error oscillates around the zero line in the amplitude range between 0.25 and -0.25 increasingly converging to zero. After approximately sample 2000 the error stays relatively stable within a range very close to zero. It is evident, though, that the error is not totally zero. Thus, the filter does not cancel out the echo totally. This results in an output where the echo is greatly reduced in amplitude, but still audible. The following figure represents the factor of amplitude reduction in relation to the number of filter coefficients up to a hundred coefficients:

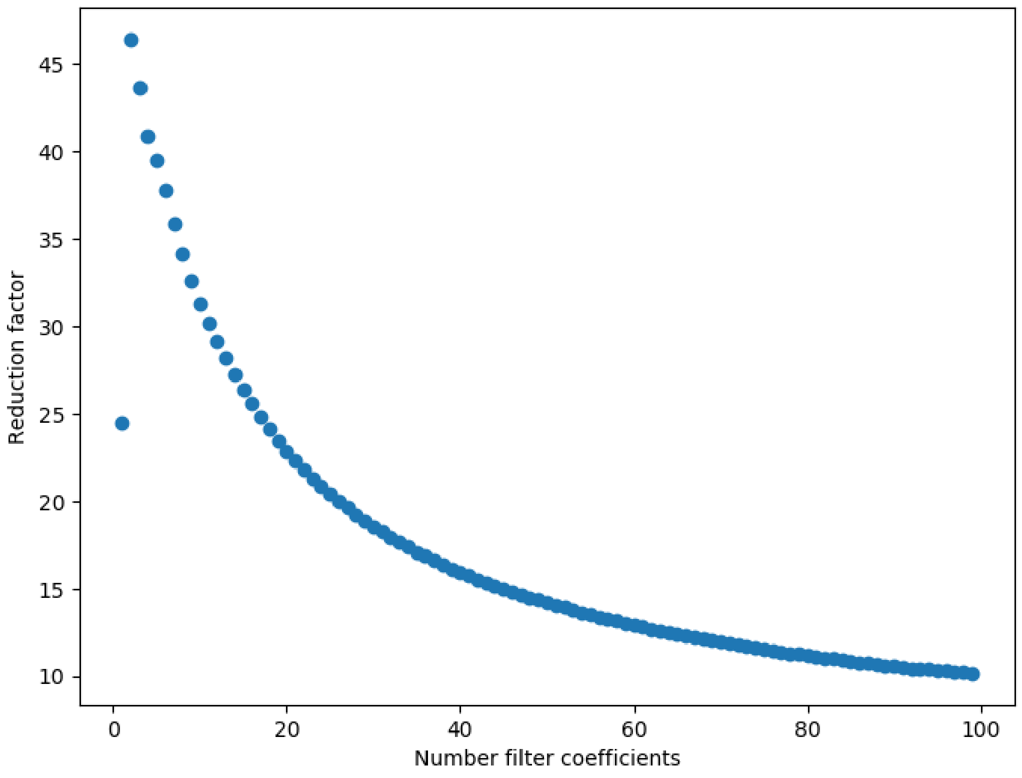


Figure 11: Factor of amplitude reduction in relation to number of filter coefficients

The amplitude reduction was measured by taking the average amplitude of the original signal and the filtered signal and taking the quotient. As can be observed, with the exception of one filter coefficient, there is a negative correlation between reduction factor and number of filter coefficients: The more filter coefficients, the smaller the reduction factor. A filter with two filter coefficients reduces the amplitude the most, i.e. by a factor of 46.4. An auditive analysis of the output showed that especially the violin sounds in the film music occurring in a regular interval of approximately one second were still very well audible. This result is not surprising as these sounds had an impulse-like character, which for adaptive algorithms is not as easy filterable as more constant noise. An issue that has not been raised so far is the choice of μ, which obviously is fundamental for all results presented so far. These results have always to be seen in relation to a given learning rate. As the NLMS was chosen as the underlying algorithm, the error should not diverge as long as $0 > \mu > 2$. An analysis showed that this was the case: The error was stable as long as μ was in the given range, but already started to diverge at $\mu = 2.06$. For the present project μ was chosen to be 0.001. As this is a relatively small learning rate, the convergence speed was relatively slow. The reason why this μ value was retained nevertheless, laid in the quality of the output. If μ was given a higher value, for example a value around one, a high-frequent hissing noise was audible in the output, distorting the whole signal. This was especially relevant for the result described in scenario 3. But before coming to this last scenario the next chapter addresses the effect on the noise frequencies of the implemented filter.

5.3.2 Scenario 2: Frequency effect

The set-up to test the filter's effect on noise frequencies was comparable to the previously described scenario: The tested algorithm consisted of a FIR filter with 10 coefficients and had a learning rate of 0.001. However, the used files were different. To test the frequency response a new noise file was created using Audacity: This new signal consisted of a signal of equally spaced sine waves with frequencies from 20 to 22 kHz. The signal had the same duration as the previously described files. A delayed version of this sine wave signal was, then, combined with the narrator file. The following figure illustrates the resulting frequency spectrum:

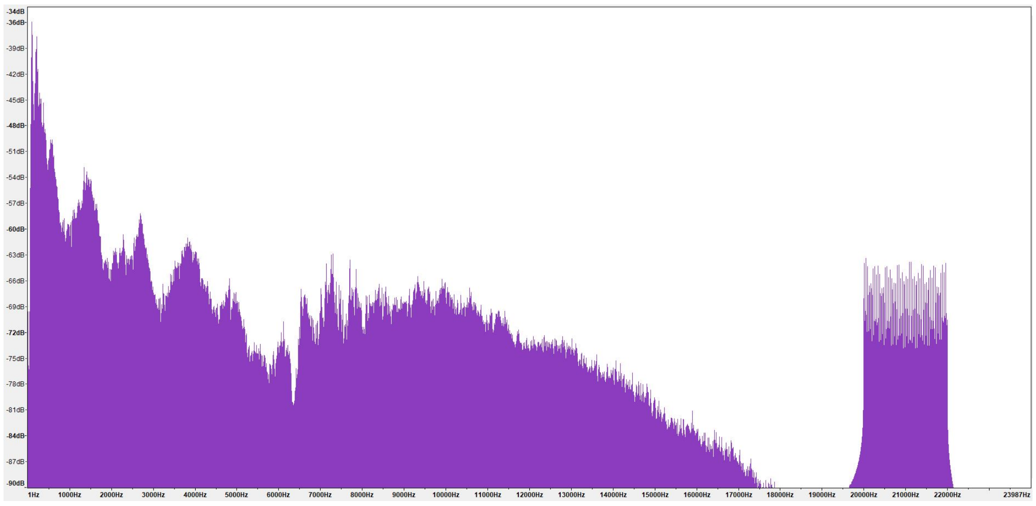


Figure 12: Frequency spectrum of sine wave corrupted narrator signal

The spectrum was created using a Hamming window as window function with a size of 32768 samples. At the right side of the spectrum the added sine wave signal is very well visible as a bar of very high frequencies ranging from 20 to 22 kHz. On the other hand, the rest of the frequency range, i.e. the frequencies from 1 Hz to approximately 18 kHz belong to the speech signal in the narrator file. To this corrupted signal the implemented filter was applied, resulting in the following frequency spectrum:

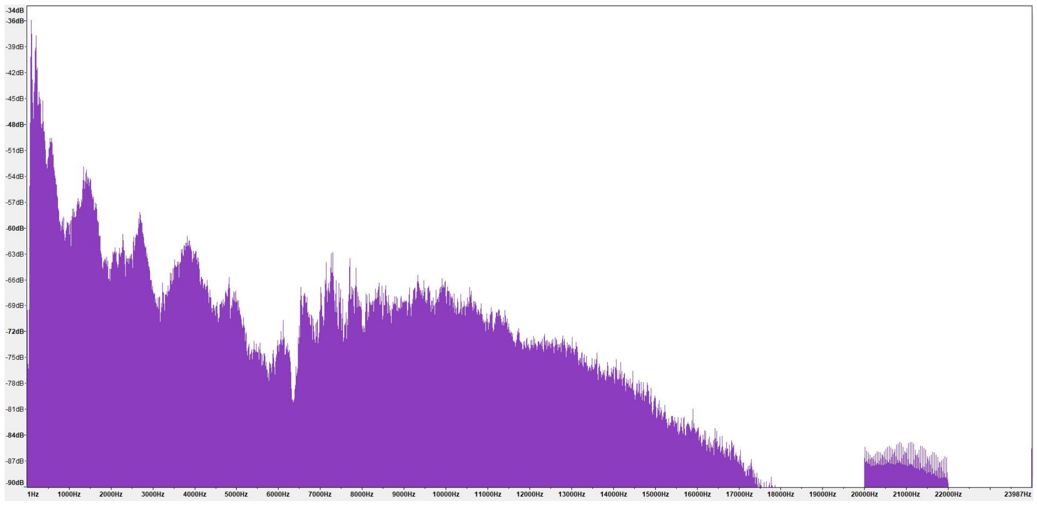


Figure 13: Spectrum of corrupted narrator signal after filtering

As expected, the frequencies belonging to the assumed noise are greatly reduced in amplitude. The frequencies in the lower frequency range, however, remained nearly unaltered. This confirms the observations already made in the previous chapter: Even though the noise is not fully removed, the relevant components of the signal are suppressed. After this short overview of the convergence behaviour and frequency effect of the implemented adaptive filter the last chapter of the analysis takes a closer look on the filter results in the original set-up. As both already analysed aspects are considered, this last chapter represents the synthesis of the previous analysis.

5.3.3 Sceanrio 3: Convergence and frequency effect in the original set-up

As in scenario 1 and 2 also in scenario 3 a filter with 10 coefficients and a learning rate of 0.001 was chosen. As already specified in chapter 5.1 the files used in the original set-up were the file with the documentary music to generate and estimate the noise, as well as the narrator speech for the desired, but noise corrupted signal. The error rate in this scenario follows a slightly different logic. The aim is still to minimize the objective function, the sum of squared errors. However, in contrast to scenario 1 we have the narrator signal as desired signal, which should not be cancelled out by our filter. Instead, the filter should approximate this desired signal as precisely as possible. This is still done by a minimization procedure, but instead of converging to zero at all points in time, the filter converges to the optimal estimate of the desired signal. The label "error" for the filter output, thus, in this case could be misleading and should therefore rather be labelled as "estimate of the uncorrupted signal".

The following figure shows the first 13.5 seconds of the Audacity oscillograms of the corrupted input signal and the estimate of the uncorrupted signal:

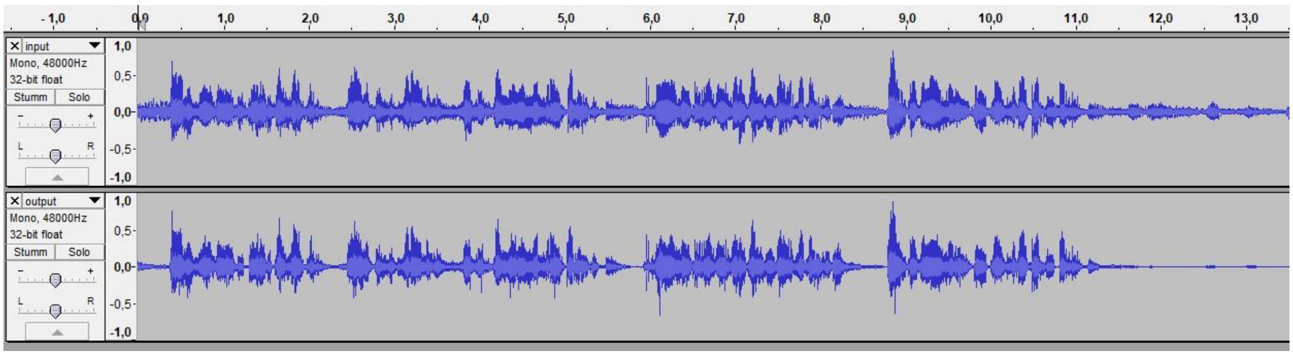


Figure 14: Oscillograms of corrupted narrator signal before (input) and after (output) filtering

The first interesting part starts right from second 0 to 0.3: In this section the first convergence cycle can be observed. In this case the signal estimate converges to zero, because only music, i.e. noise is present in this period. The same pattern is observable from second 11 onwards, where a longer pure noise period characterizes the corrupted signal. As already described in the previous chapters, the algorithm takes a while (approx. 0.5 seconds) to converge to a certain error level and then stays on this level from 11.5 seconds onwards. The still visible swings in the signal at 12.5 and 13.0 are the impulse-like violin sounds. Another interesting phenomenon, which is illustrated by these oscillograms, is the fact that the algorithm always must re-adapt if the input signal changes abruptly. Therefore, after each speaking pause of the narrator, where only music is present, the filter must re-adapt. This leads to a higher noise amplitude until the filter has re-adjusted. Another consequence of this adjustment process is that not only the noise amplitude but also the amplitude of the desired amplitude is partially increased. This is for example the case at second 6, where a large negative peak is recognizable, which is not present in the input signal. This amplitude overdrive after speaking pauses could also be the explanation for another observation: The auditive quality in general was again strongly dependent from the choice for μ. The algorithm did converge also in this case as long as $0 < \mu < 2$. A higher μ as expected led to a higher convergence rate. The reason why a relatively small μ was retained, was the quality of the outputted signal estimate: A higher μ led to a signal estimate where nearly all music had been cancelled out, but that was strongly distorted by random noise components. A possible explanation could be the amplitude overdrive just mentioned. As a higher μ leads to higher fluctuations of the error at the beginning of a convergence cycle, samples right after the end of a speaking pause are increased in amplitude in a way such that noise frequencies are introduced. The following figure 15 illustrates this issue:

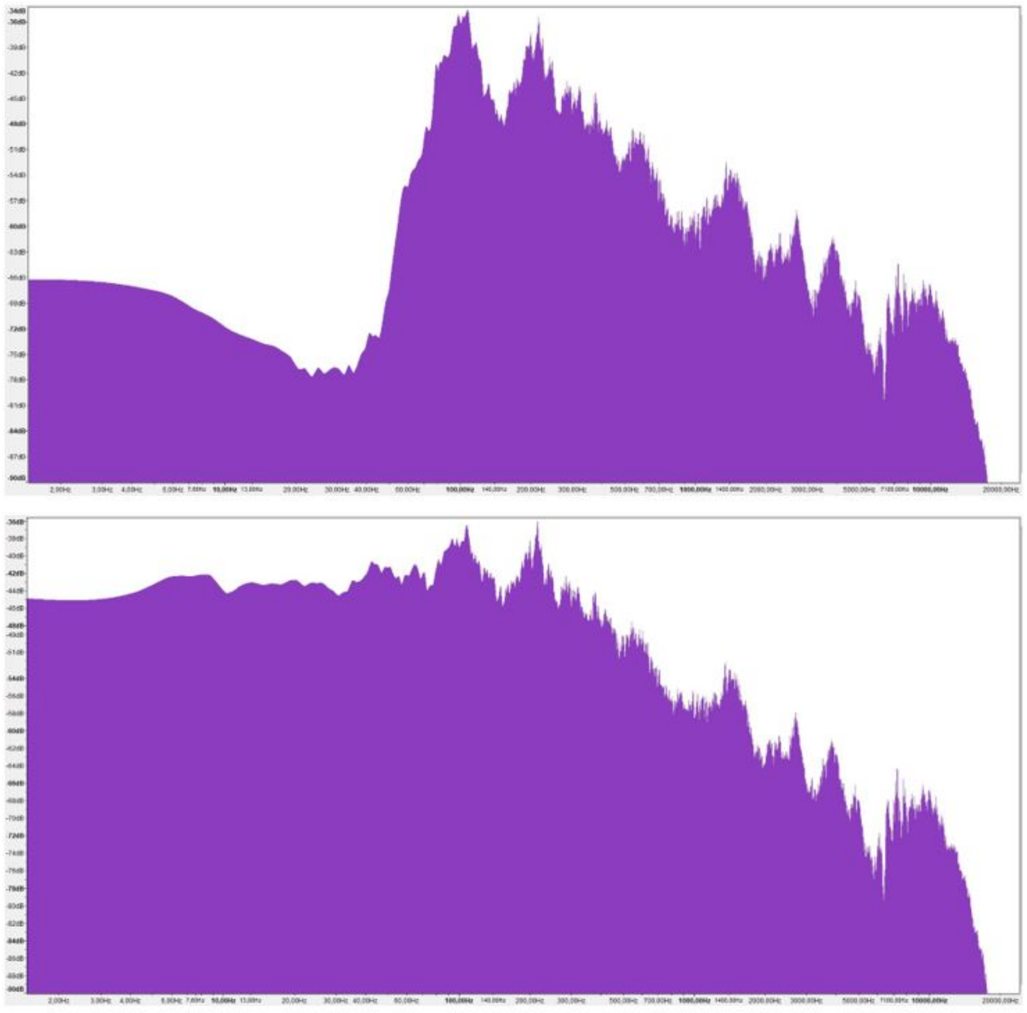


Figure 15: Target spectrum and spectrum of filtered narrator signal with learning rate 1

The upper spectrum is the spectrum of the target signal – the uncorrupted narrator speech. The lower spectrum is the spectrum resulting from applying the filter to the noise corrupted signal with a learning rate of 1. The right halves of both spectra, i.e. frequencies in the range from 70 to 20000 Hz, show a high similarity – in this case the adaptive filter worked properly. It is, however, also very well visible that frequencies below 100 Hz are highly increased in amplitude. The noise in the filtered signal when using higher learning rates, thus, has its origin in a low-frequency distortion. This distortion is not observed to this extent when using the aforementioned learning rate of 0.001:

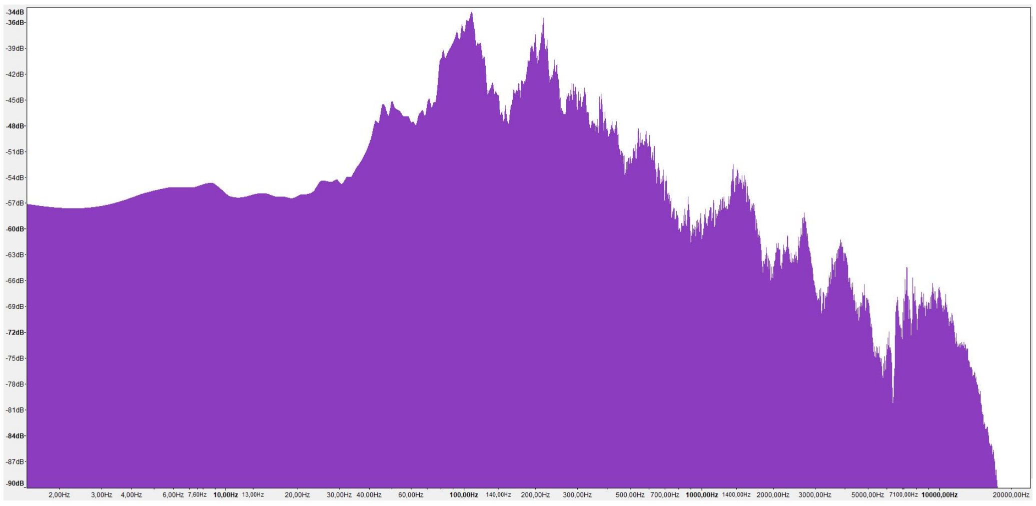


Figure 16: Spectrum of filtered narrator signal with learning rate 0.001

The spectrum still shows increased amplitudes in the frequency range below 100 Hz in reference to the target signal, but the increase is not that pronounced as in the spectrum with learning rate 1. From an auditive perspective this led to an inferior noise suppression, but a higher sound quality as the low frequency noise components were barely audible. Consequently, and as expected from the analyses in the previous chapters also in the original scenario the application of the LMS led to an improvement of the corrupted signal, even though the noise could not be removed totally without distorting the output.

6 Conclusion

The aim of the project described in this seminar paper was to implement a functioning LMS filter and to apply it to real data, modelling a simplified echo cancellation scenario. While the chapters 2 to 4 laid the foundations for a better understanding of digital filters in general and the LMS adaptive filter in particular, chapter 5 described the implementation process as well as the functioning of the implementation. The presented results showed that the LMS filter, implemented in a normalised form, was able to reduce the assumed noise in all scenarios. As in chapter 5.1 and 5.2 became clear, it showed the expected convergence behaviour and frequency response. However, especially the set-up in the original scenario, described in chapter 5.3, brought to light the limitations of the algorithm. In all scenarios 0.001 was chosen as learning rate. Higher learning rates would have led to a higher convergence speed and an even better noise reduction. The reason why μ nevertheless was chosen to be relatively low was that a higher learning rate added low-frequent noise to the filter's output. This noise distorted the output to a higher extent than the original noise and greatly reduced sound quality. With lower learning rates this effect was not observed. An analysis showed that the distorting frequencies mainly laid in the frequency range below 100 Hz. It was hypothesized that the reason for this low-frequent noise could lie in an amplitude overdrive of the filter after silent periods in the narrator signal. This, however, does not explain why only low-frequencies are enhanced in a way to distort the filter's output. Moreover, the amplitude overdrive did not occur right after the silent periods, but shortly afterwards. The explanations given here seem to be insufficient for explaining the observed low-frequent noise distortions. Thus, this question must remain unanswered and could be interesting for future works. Moreover, also other improvements and extensions are imaginable. The model underlying the present project was a very simplistic one. In the original set-up the echo was modelled as a modified version of the music delayed by one sample and with reduced amplitude. Naturally occurring echoes are far more complex. Future projects could test the filter in more natural conditions. Another possible improvement concerns the learning rate. In many Machine Learning algorithms the learning rate is not fixed, but flexible. A common solution, for example, is to start with a higher learning rate decreasing over time. But also more complex approaches exist, as presented for example by Zeiler (2012) [14]. To conclude, the project described in this seminar paper was successful in two respects: Central concepts learnt in the seminar where revised and advanced. Moreover, completely new knowledge, for example regarding gradient descent, could be acquired, highly relevant for future seminars.

7 References

[1] John R. Deller, JR., John H. L. Hansen, and John G. Proakis. *Discrete-Time Processing of Speech Signals*. IEEE Press, New York, 2000.

[2] Paulo S.R. Diniz. *Adaptive Filtering: Algorithms and Practical Implementation*. Springer Science+Business Media, New York, 3 edition, 2008.

[3] Minh N. Do. Linear regression and least-squares filtering. `https://courses.engr.illinois.edu/ece420/fa2017/AdaptiveSignalProcessing.pdf`, 2017.

[4] Keith Johnson. *Acoustic and Auditory Phonetics*. Wiley-Blackwell, Oxford, 3 edition, 2012.

[5] Karl-Dirk Kammeyer and Kristian Kroschel. *Digitale Signalverarbeitung: Filterung und Spektralanalyse mit MATLAB-Übungen*. Springer Vieweg, Wiesbaden, 9 edition, 2018.

[6] E. Kaymak, M. Atherton, K. Rotter, and B. Millar. Block diagram of an adaptive filter as a noise canceller. `https://www.researchgate.net/figure/Block-diagram-of-an-adaptive-filter-as-a-noise-canceller_fig4_267150925`, 08.2007.

[7] Nishant Kumar. Block diagram of an iir filter. `https://www.researchgate.net/figure/Block-Diagram-of-an-IIR-Filter_fig2_307577369`, 07.2016.

[8] National Instruments Website. Adaptive echo cancellation (adaptive filter toolkit). `http://zone.ni.com/reference/en-XX/help/372357B-01/lvaftconcepts/aft_aec/`.

[9] Steven W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, San Diego, California, 2 edition, 1999.

[10] B. Widrow and M. E. Hoff. *Adaptive switching circuits*. Stanford University, Stanford, CA, 1960.

[11] Norbert Wiener. *Extrapolation, Interpolation and Smoothing of Stationary Time Series, with Engineering Applications*. Technology Press and Wiley, New York, 1949.

[12] Wikipedia. Finite impulse response. `https://en.wikipedia.org/wiki/Finite_impulse_response`, 01.07.2019.

[13] W. Wokurek. Speech signal processing and speech enhancement: Course manuscript. 04.07.2019.

[14] Matthew D. Zeiler. Adadelta: An adaptive learning rate method. `https://arxiv.org/pdf/1212.5701`, 2012.

8 List of Figures