Technische Universität Berlin

Fachgebiet Audiokommunikation

Masterarbeit Audiokommunikation und -technologie

Tweaked Speech Codec to Inhibit Howling Suppression

vorgelegt von

David Ditter

[redacted]

Abgabetermin

26. November 2014

Betreuer

Asst. Prof. Edgar Berdahl

contact

[redacted]

Abstract

Acoustical feedback is present whenever a speaker signal gets redirected to a microphone that feeds its input signal directly or indirectly into this very speaker. If the gain of such a closed feedback loop is close to or higher than 1, unpleasant acoustical artifacts will occur and will often times lead to the sinusoidal howling noise, that probably everybody has experienced at a presentation or a concert, when a microphone was accidentally directed to a speaker. This research work tried to suppress these unwanted effects of acoustical feedback by the insertion of a modified speech codec into the signal path of the feedback loop. Two basic concepts were utilized: The *ADPCM* codec as well as the *CELP* codec, which were both implemented as `C++` real-time audio modules and tweaked for the purpose of feedback suppression. A total of 13 algorithm configurations were tested for their ability to suppress feedback within a self-developed software simulation framework as well as in a live test session, which was conducted in a conference room and a large lecture hall. Results showed, that the *Speex* open-source speech codec was successfully tweaked to increase the *maximum stable feedback gain* by 3 dB, but which goes along with a reduction of the audio quality within the loop.

Zusammenfassung

Akustische Rückkopplung ist immer dann präsent, wenn ein Lautsprechersignal zu einem Mikrofon gelangt, dessen Ausgangssignal zu ebendiesem Lautsprecher (direkt oder indekt) weitergeleitet wird. Wenn der Verstärkungsfaktor einer solchen geschlossenen Rückkopplungsschleife nahezu oder größer als 1 ist, so treten unerwünschte akustische Artefakte auf. In vielen Fällen ein sinusartiger, heulender Ton, den vermutlich jeder bereits einmal wahrgenommen hat, wenn bei einer Präsentation oder einem Konzert ein Mikrofon aus Versehen auf einen Lautsprecher gerichtet wurde. Diese Forschungsarbeit hat den Versuch unternommen, diese unerwünschten Effekte zu unterdrücken und zwar durch das Einfügen eines modifizierten Sprach-Codierungs-Algorithmus in den Signalpfad der Rückkopplungsschleife. Zwei grundlegende Konzepte wurden hierfür eingesetzt: Die *ADPCM*- und die *CELP*-Codierung, die beide als `C++` Echtzeit-Audiomodule implementiert und für die Zwecke der Feedback-Unterdrückung optimiert wurden. Insgesamt 13 verschiedene Konfigurationen der implementierten Algorithmen wurden in der Hauptmessung, die sowohl in einem Konferenzraum, als auch in einem großen Hörsaal ausgeführt wurde, vermessen. Die Resultate der Messung zeigen, dass der *Speex* Open-Source Codec erfolgreich für den Zweck der Feedback-Unterdrückung optimiert wurde. Der Anstieg des *maximalen, stabilen Verstärkungsfaktors* liegt bei 3 *dB* und mehr, welcher allerdings mit einer Reduktion der Audioqualität einhergeht.

Contents

Overview

The thesis at hand is a summary of the research work of tweaking speech codecs to inhibit howling suppression.

Chapter 1 introduces to the topic of acoustical feedback and to the common research approaches and methods in this field. Furthermore, an insight is given on the primal motivation for the approach of using speech codecs to inhibit howling suppression.

As this work relies heavily on the basic concepts of speech coding – especially on the *ADPCM* and *CELP* codec –, Chapter 2 gives a summary of the concepts that have been utilized in the software implementation part of this project.

Chapter 3 explains in detail, how the algorithms under test were implemented and configured. Four main approaches were realized as *JACK* applications, written in `C++`. Further, a description of the implementation of the frequency shifter as a baseline reference for the measurements is included to this chapter.

The testings of the effects on the system feedback have been carried out in a software simulation framework as well as in a real-world usage scenario in two different rooms: a conference room and a large lecture hall. The detailed description of how these measurements were conducted as well as the measurement data presenting the characteristics of the utilized rooms, can be found in Chapter 4.

The results of all measurements are presented in the following Chapter 5 with an attached discussion of the presented data. Last, Chapter 6 completes this thesis with the conclusions of this research project.

1. Acoustical feedback and howling

In basically all audio systems that have a microphone and a speaker running at the same time, acoustical feedback is an issue. Such systems are installed at events like conferences of any size, presentations and live readings or concerts. Even in the case of the microphone and the speaker not being within the same room, the effect can still be present, for example in landline or *voice over IP* telephony, especially when using PC speakers or table speakerphones for reproducing the transmitted audio signal on the receiver side.

All these systems usually have a *closed feedback loop*, meaning there is an acoustical path within the system from the speaker to the microphone which feeds its signal (possibly indirectly) into this very speaker. Whenever the feedback loop is closed, it is possible for the system to go unstable all by itself, depending on the gain of the feedback loop, which can mostly be controlled by an amplifier or mixing unit within the signal chain. Everybody should have experienced this – for example at a presentation or at a concert –, that all of a sudden a *howling* noise appears that increases its energy very fast and hence, is a very annoying, (and often times physiologically dangerous,) acoustical phenomenon.

The target of this work is to approach this problem of an unstable, closed feedback loop with the basic concepts of speech coding.

1.0 Terminology

As there is no consistent use of the terms *acoustical feedback* and *howling* in the literature, a definition for the scope of this work is needed:

- *acoustical feedback*: Acoustical coupling between one or more microphones and one or more speakers which leads to various artifacts in the speaker signals. With this definition, there is no necessity for audible, acoustical artifacts, to apply this term.

- *howling*: Result of acoustical coupling where one ore more frequencies are unstable in the closed loop and hence, create the certain *howling* sound.

1.1 Problem description

1.1.1 Feedback analysis for time-invariant systems

As mentioned in the introduction to this chapter, probably everybody has made the observation, that directing a microphone to speaker can lead to *howling*. To get a more theoretical understanding of this phenomenon, we should look at the analytical ways to determine the stabilty of a system and then hold it against our observation.

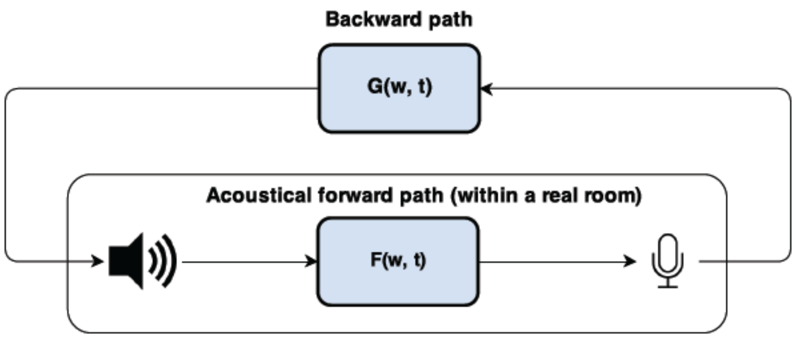

Figure 1.1: Diagram of the *closed feedback loop*.

Figure 1.1 shows the signal path in the *closed feedback loop* which allows us to obtain the *loop transfer function* of the system with no active time-varying block, according to [WM09, p. 292]:

$$\frac{U(\omega,t)}{V(\omega,t)} = \frac{G(\omega,t)}{1 - G(\omega,t)\cdot F(\omega,t)} \tag{1.1}$$

where $\omega \in [0, 2\pi]$ represents the radial frequency variable, $U(\omega,t)$ and $V(\omega,t)$ denote the short-term frequency-spectra of the loudspeaker and source signal and $G(\omega,t)$ and $F(\omega,t)$ are the short-term frequency responses of the forward and feedback path.

In this case, the time-invariant system can be analyzed with the *Nyquist stability criterion* that was introduced by [Nyq32] and furthermore explained in [Lev96, Ch. 8.2]. According to [WM09, p. 292], this critierion can be forumlated as follows:

A linear time-invariant system is unstable, if we can find a frequency for that the following two conditions are met:

$$|G(\omega, t) \cdot F(\omega, t)| \geq 1 \tag{1.2}$$

$$\angle G(\omega, t) \cdot F(\omega, t) = n \cdot 2 \cdot \pi, \quad n \in \mathbb{Z} \tag{1.3}$$

In words, a linear-time invariant system is unstable, if there exists a frequency f_0 for that the follwing is true:

- The feedback loop gain at frequency f_0 is equal to or greater than 1.
- The phase shift for this frequency f_0 is a mulitple of 2π, i.e. there is no interference in the feedback loop for this frequency and hence, the waves will not cancel but add up maximally.

This explains the phenomenon from above: When a directional microphone is directed to a speaker, the gain of the transfer function $G(\omega, t)$ will be significantly higher. So here, the variables of the first criterion are varied and hence, lead to an unstable system and by this to the noticable *howling* sound.

1.1.2 Feedback analysis for time-varying systems

As this work will approach to modify the feedback loop by the insertion of a modified speech codec into the signal path, we should also know about the theoretical stabilty criteria for time-varying systems, because we will see in Chapter 2 that the *ADPCM* codec as well as the *CELP* codec are time-varying systems.

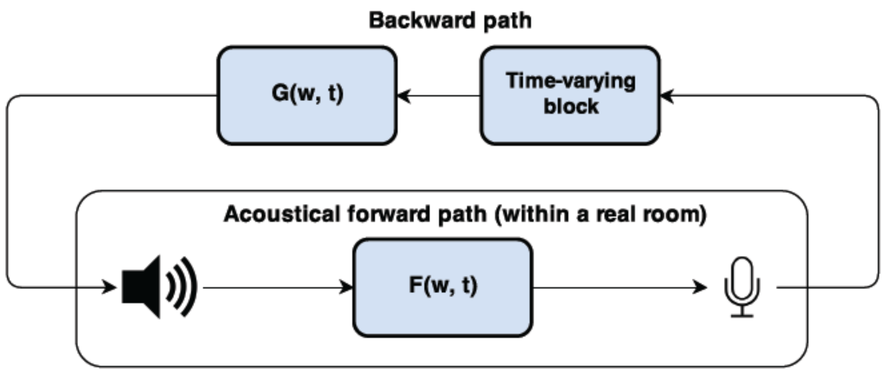

Figure 1.2: Diagram of the *closed feedback loop* with an inserted time-varying block.

In the case of time-varying systems, we can not utilize the *Nyquist stabilty criterion*, but the so-called *circle criterion* instead, which can not be linked to a single author, as [Wil71, Ch. 5.4, p. 130] states: ”There are several more or less independent and simultaneous sources for the stabilty criterion.” These sources include [San64], [San65] as well as [Zam66] and [Zam64].

Summarizing the explanations by [Wil71, Ch. 5.4, p. 130] to the circle criterion, a time-varying system is stable, if

- (as a precondition) the open-loop[1] system is stable
- there exists a so-called *critical disk* (see [Wil71, Ch. 5.4]) within the z-plane such that for almost all time instances, the transfer function of the time-varying block lies within this disk.

1.2 Common approaches to feedback suppression

Three common approaches to suppress acoustical feedback are introduced in the following paragraphs. A more comprehensive and historically complete overview over the *DSP*-based approaches can be found in [WM09].

Phase modulation methods

Phase modulation methods attempt to skirt the *revised Bode stability criterion* (see [Lev96, Ch. 8]) by continually varying the phase for all frequencies in the system. If the microphone and the loop signal are not in-phase for a sufficiently long time, *howling* can be effectively suppressed.

One way to implement this concept, is to apply a pitch or frequency shift to the signal. If we shift a certain frequency, we change its wavelength and hence also change its phase at the microphone position. This method was introduced by [Sch64] and was recently evaluated by [BH10]. They showed that the *maximum stable gain* can be increased by as much as 4 dB or more by using a frequency-shift of up to 8 Hz on the signal. However, the *increase in maximum stable gain* is limited in practice by artifacts introduced by the frequency shifter.

[1] *open loop system* stands for a system where no round-trip is possible

Gain reduction methods

Algorithms from this category try to detect frequencies that are amplified in the feedback loop. If such a frequency is detected, gain will be reduced until the feedback effect stops. The gain can be reduced

1. at all frequencies,
2. at a narrowband around the detected frequency using a notch filter,
3. by a wider notch filter at the detected frequency.

While the first solution is a simple and robust one, it is surely not the best idea in terms of audio quality. The same can be said about the second approach. Hence, the third type – gain reduction by a notch-filter – is recently the most used one. However, stability problems are often requiring multiple notch filters, which may effectively reduce the volume as in the first gain reduction method.

Room modeling methods

Room modeling methods measure the frequency response of the system (either online or offline) and afterwards try to flatten it by convoluting the inverted response with the input signal. As *howling* is likely to occur at frequencies with an above-average amplification, this inversion is able to suppress feedback. This method is a suitable solution in systems where there is one microphone and one loudspeaker. But it gets difficult to execute, when there are multiple microphones and speakers, as they will lead to a variety of measurable system responses that can not be flattened all at once. Furthermore, in a practical context of a room, the effect is similar to introducing notch filters to flatten out peaks in the loop transfer function.

Now that we have seen the common approaches to feedback suppression, we want to make the approach of tweaking speech codecs reasonable.

1.3 Approach by tweaking speech codecs

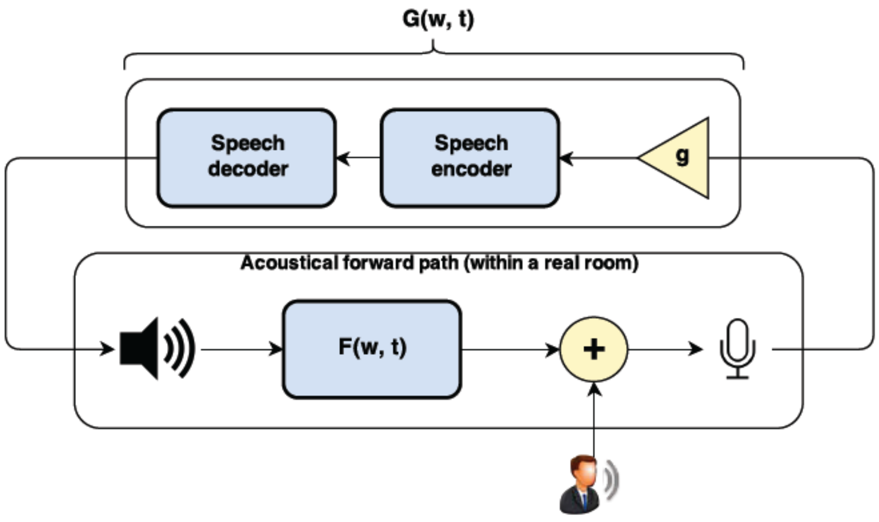

Figure 1.3: Closed feedback loop with an inserted spech encoder and decoder (*codec*).

Figure 1.3 allows us to understand the theoretical idea, that was the primal motivation to approach the problem of unstable feedback with a speech codec, which can be summarized in the following ***working hypothesis* for this thesis**:

> *"The value of $|G(\omega, t)|$ is smaller for acoustic feedback ringing (howling) than it is for speech. This implies, that for speech signals we can increase the volume gain g without violating first part of the Nyquist stability criterion* $|G(\omega, t) \cdot F(\omega, t)| \leq 1$.*"*

The reasoning here is, that a well tuned speech codec should transmit all actual speech signals, but should not transmit signals that are not speech-like, like the feedback ringing sound.

Besides this theoretical concept, there were three further reasons to approach the problem of acoustical feedback with the signal processing concepts of speech coding:

1. Preliminary tests on the *Speex*[2] open-source speech codec had suggested that running a signal through the codec effects the stable feedback gain positively, especially for the low and ultra-low quality settings of *Speex*.

[2] see http://www.speex.org/

2. In contrast to some of the common approaches to suppress feedback, which reduce the output gain of the signal, speech codecs have no mean to change the gain of the signal[3]. Furthermore, as we will see, the *CELP* codec for example does a complex resynthesis of the signal which might lead to time-varying changes in the phase of the output signal which could be beneficial for feedback suppression.

3. Within a practical scenario where a signal has to be transmitted through a digital channel, some codec scheme has to be applied in any case. If acoustical feedback is an issue in that use-case, one could use the knowledge about the effect on feedback as an additional criterion for selecting the optimal codec for that scenario. By this, one could suppress feedback without having to add an additional feedback suppression algorithm and by this saving development effort and *CPU load*.

[3]This is correct for the core speech codec technologies. Often times, speech codec implementations are enhanced by a an *automatic gain control* in the preprocessing phase. None of such algorithms were utilized within this work.

2. Elementary speech codec technologies

As in the course of this work we will make heavy use of standard speech coding techniques, this chapter explains the basic concepts that were adapted and that will be crucial to understand the tweaking modifications explained in the following chapter.

2.1 Linear Predictive Coding (*LPC*)

All of the adapted speech codecs in this work depend on the concept of *Linear Predictive Coding.* Its general idea is to minimize the variance of the transmitted signal by transmitting the error signal of a linear prediction, instead of transmitting the signal itself (see [KK89, p. 399]). As the receiver can make the same predictions as the sender by using his reconstructed previous signal samples, it can use the transmitted prediction error signal for a full reconstruction of the signal.

2.1.1 Analysis

The first assumption is, that we can find coefficients $p_1, p_2, ..., p_N$ so that a linear combination of the previous N signal samples is a good estimate $\hat{x}[n]$ for the current sample $x[n]$, in formula:

$$p_1 \cdot x[n-1] + p_2 \cdot x[n-2] + ... + p_N \cdot x[n-N] = \hat{x}[n] \approx x[n]. \tag{2.1}$$

With this, the prediction signal $\hat{x}(n)$ is generated from the original signal $x(n)$ by applying a linear filter with impulse response

$$p(n) = \sum_{k=1}^{N} p_k \cdot \delta(n-k), \tag{2.2}$$

where δ denotes the Dirac function. From this formula we can obtain the transfer function $P(z)$ for the system between input and prediction signal:

$$P(z) = p_1 \cdot z^{-1} + ... + p_N \cdot z^{-N} \tag{2.3}$$

Now, we can rewrite the formula for the difference signal $d(n)$ between the actual and the predicted signal which is represented in the linear system seen in figure 2.1:

$$d(n) = x(n) - \hat{x}(n) \tag{2.4}$$

$$\iff d(n) = x(n) - \big(p_1 \cdot x(n-1) + ... + p_N \cdot x(N-n)\big). \tag{2.5}$$

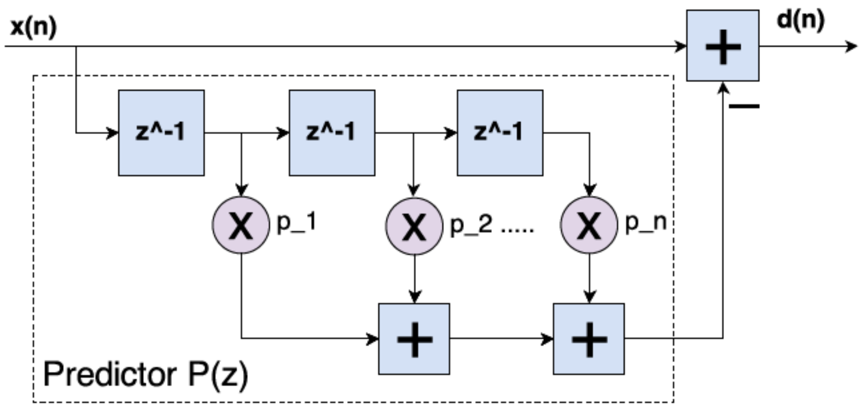

Figure 2.1: *LPC analysis filter*, see equation 2.5. Diagram adapted from [Hay96, p.243].

From equation 2.5 we can derive the transfer function for the linear system between the input signal and the error/difference signal:

$$D(z) = X(z) - \big(p_1 \cdot X(z) \cdot z^{-1} + ... + p_N \cdot X(z) \cdot z^{-N}\big) \tag{2.6}$$

$$\iff D(z) = X(z) \cdot \big(1 - (p_1 \cdot z^{-1} + ... + p_N \cdot z^{-N})\big) \tag{2.7}$$

$$\iff \frac{D(z)}{X(z)} = 1 - \underbrace{(p_1 \cdot z^{-1} + ... + p_N \cdot z^{-N})}_{=P(z)}. \tag{2.8}$$

So the system transfer function for the signal analysis is:

$$H_{\text{Analysis}}(z) = \frac{D(z)}{X(z)} = 1 - P(z) \tag{2.9}$$

The linear system H_{Analysis} is often referred to as the *LPC analysis filter* .

2.1.2 Optimal prediction for a deterministic signal

To find optimal prediction coefficients $p_1, p_2, ..., p_N$ for a fixed N and a deterministic signal, we have to minimize the variance of the prediction error signal $d(n)$:

$$\sigma_D^2 = E[D^2(n)] = E[\{X(n) - \hat{X}(n)\}] \overset{!}{=} Min. \tag{2.10}$$

As shown in [KK89, p.356f], this is equivalent to solving the system of linear equations:

$$\mathbf{R}_{XX} \cdot \mathbf{p}_{opt} = \mathbf{r}_{XX} \tag{2.11}$$

where $\mathbf{R}_{XX}$ is the auto-correlation matrix, $\mathbf{r}_{XX}$ is the auto-correlation vector of the signal $x(n)$ and $\mathbf{p}_{opt}$ is the vector consisting of the N optimal prediction coefficients. As $\mathbf{R}_{XX}$ is a *Toeplitz matrix*[1] and the right hand side vector consists of the unique values of the left hand side Toeplitz matrix, the system can be efficiently solved for $\mathbf{p}_{opt}$ by the *Levinson-Durbin recursion* (see [Lev47] and [Dur60]).

As [KK89, p.351f] shows, the variance of the transmitted signal $d(n)$ for $\mathbf{p}_{opt}$ is

$$\sigma_D^2 = \sigma_X^2 - \mathbf{r}_{XX}^T \cdot \mathbf{R}_{XX}^{-1} \cdot \mathbf{r}_{XX} \tag{2.12}$$

which implies, that $\mathbf{r}_{XX}^T \cdot \mathbf{R}_{XX}^{-1} \cdot \mathbf{r}_{XX}$ is the decrease in variance compared to the input signal $x(n)$.

2.1.3 Synthesis

The original signal $x(n)$ can be resynthesized, if only $d(n)$ and $P(z)$ are known. This is done by the the reciprocal analysis filter, the so-called *LPC synthesis filter* with the following system transfer function:

$$H_{Synthesis}(z) = \frac{1}{H_{Analysis}(z)} = \frac{1}{1 - P(z)}. \tag{2.13}$$

[1] *Toeplitz matrix* definition: $A_{i,j} = A_{k,l}$, if $i - j = k - l$ for all $i, j, k, l = 1, ..., N$, i.e. all entries on each upper-left to bottom-right diagonal coincide.

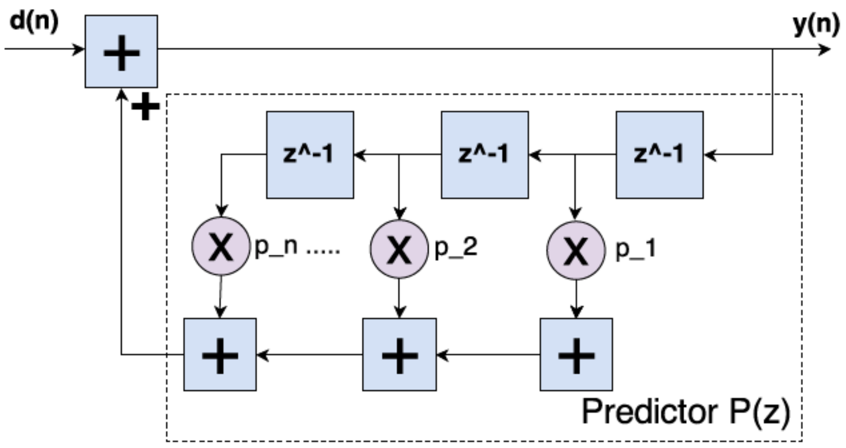

Figure 2.2: Block diagram of the *LPC synthesis filter* $H_{Synthesis}$, according to equation 2.13. Diagram adapted from [Hay96, p.243].

2.1.4 Differential Pulse Code Modulation *(DPCM)*

The basic speech codec that utilizes the concepts of *LPC* is called *Differential Pulse Code Modulation* (*DPCM*) (cf. [GR10, chapter 7.3.1]). In this codec (see figure 2.3), sender and receiver have to agree on the prediction coefficients $p_1, p_2, ..., p_N$ prior to the transmission. After *LPC analysis* on the sender side, the signal $d(n)$ is transmitted in PCM^2 representation through the channel. The receiver performs the *LPC analysis* on the received signal and obtains the reconstructed signal.

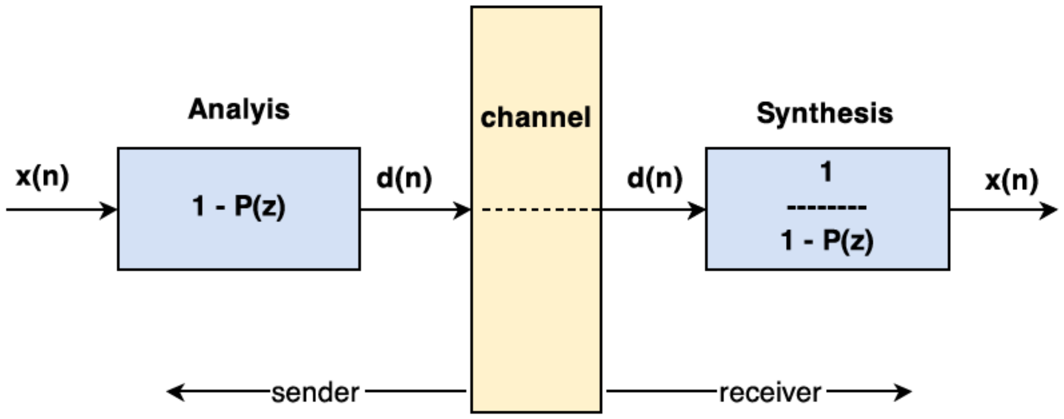

Figure 2.3: DPCM codec. Diagram adapted from [Nol12].

[2] *Pulse Code Modulation* (PCM) is a digital representation of a signal, where the coded values refer to the actual amplitude of the signal.

2.2 Adaptive *DPCM* (*ADPCM*)

The *Adaptive Differential Pulse Code Modulation* (*ADPCM*) is an extension to the *DPCM* codec and the concepts of *LPC* (see [GR10, chapter 7.4.2]). As mentioned in 2.1.2, finding optimal prediction coefficients requires a deterministic signal. But in general and especially in the case of speech, we can not assume a deterministic signal. The short-term power spectrum of the signal might differ from the long-term power spectrum, as spectral formants can shift or the signal spectrum will differ for the different phonemes in speech.

In all cases where the short-term power spectrum differs from the long-term one, the former optimal prediction coefficients $\mathbf{p}_{opt}$ are no longer optimal for short-term prediction. To solve this problem, we have to adapt the vector of prediction coefficients $\mathbf{p}$ constantly in time to match the current short-term power spectrum, which is why this codec is called *Adpative DPCM*. The adaption of the prediction coefficients can be executed in either a *backward* or a *forward* fashion.

2.2.1 Forward adaption

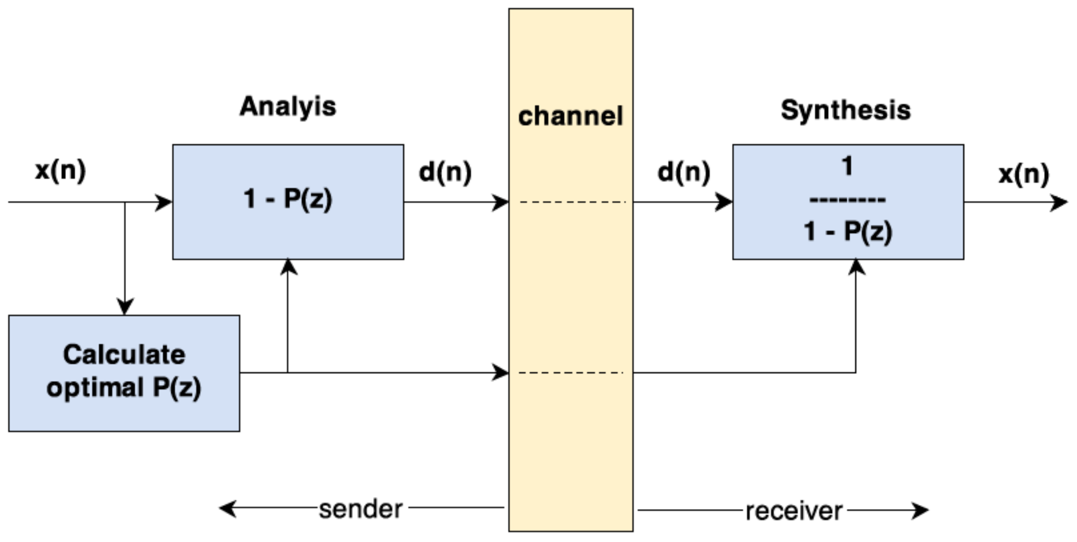

Figure 2.4: *Forward ADPCM*, frame-by-frame processing. Diagram adapted from [Nol12].

One way to realize an *ADPCM* codec is to modify the *DPCM* by using adaptive prediction with *forward adaption*. For this, the signal has to be processed in a frame-by-frame manner. Instead of using constant prediction coefficients on both sides, for each frame of audio we calculate a current $\mathbf{p}_{opt,current}$ on the sender side, using the methods described in Section 2.1.2. After the calculation, $\mathbf{p}_{opt,current}$ gets passed to the *LPC analysis* filter as well as the *LPC synthesis* filter (see Figure 2.4) and a traditional DPCM is performed.

For this process, additional data has to be transmitted through the channel. Usually, these coefficients are coded before channel transmission using so-called *reflection coefficients* (see [Mak77]), because the filter $P(z)$ is very sensitive to quantization errors in the pure prediction coefficients. Alongside the additional channel throughput, this codec has a forced frame-by-frame processing, which implies a non-zero latency of the codec.

2.2.2 Backward adaption

A second way to realize an *ADPCM* codec is by using a backwards structure, meaning that it only relies on past signal frames and therefore has zero delay (see [Kon04, p. 222]). The aim of the *LPC* coefficient adaption in this case is to minimize the energy E_d of the transmitted signal $d(n)$.

To get an easy idea on how to minimize this energy, let us assume that we have only a single *LPC* coefficient p_1, meaning $N = 1$. For a fixed time instance n, we can obtain a formula for the energy:

$$E[d] = d(n)^2 = (x_n - p_1 \cdot x_{n-1})^2. \tag{2.14}$$

As we have fixed the time to n and we are interested in minimizing the term over p_1, we should rewrite equation (2.14) as a function of p_1:

$$\begin{aligned} d_n(p_1)^2 &= (x_n - p_1 \cdot x_{n-1})^2 && (2.15) \\ &= x_{n-1}^2 \cdot p_1^2 - 2 \cdot x_n \cdot x_{n-1} \cdot p_1 + x(n)^2. && (2.16) \end{aligned}$$

So, the energy of the transmitted signal is a parabolic function in p_1, as schematically drawn in Figure 2.5.

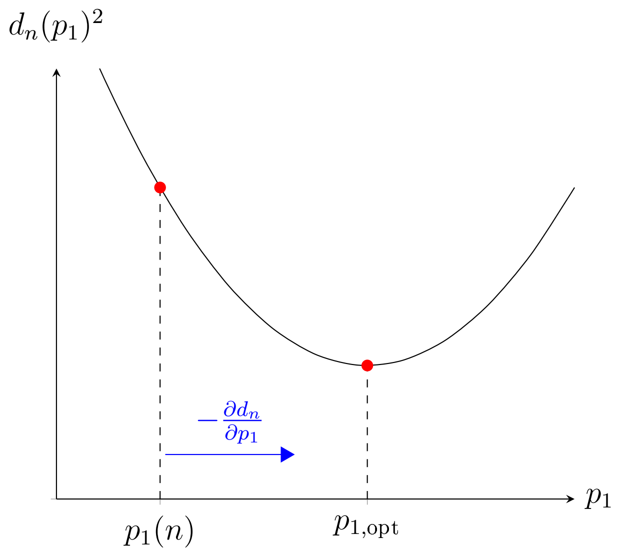

Figure 2.5: Schematic curve of the parabolic error function $d_n(p_1)^2$. The optimal LPC coefficient $p_{1,\text{opt}}$ lies at the minimum point of the curve. To minimize the prediction error, we have to shift $p_1(n)$ in the direction of the negative gradient $-\frac{\partial d_n}{\partial p_1}$. Diagram adapted from [Nol12].
.

Since it is expensive (especially for $N > 1$) to find $p_{1,\text{opt}}$ analytically, we try to get closer to it by walking in the opposite direction of the gradient of $E[d]$ at the postion of $p_1(n)$ instead. Imagine the current prediction coefficient in a position left to $p_{1,\text{opt}}$ as in figure 2.5. The derivative/gradient would have a negative value because the graph decreases at $p_1(n)$ and so it would be correct to walk in the opposite (positive) direction to get closer to $p_{1,\text{opt}}$.

Accordingly, if $p_1(n)$ were to be on the right side of $p_{1,\text{opt}}$, the gradient at $p_1(n)$ would be positive, and again, walking in the opposite (negative) direction would be the right decision. Note that, if the step in the correct direction is too large, it may happen that the value of $d_n(p_1)^2$ is even worse than before. Finding an optimal step size is a main part of configuring an *ADPCM* codec.

So, we need a formula for the derivative/gradient of $E[d]$ to get the correct step direction, which we obtain by differentiating equation (2.14):

$$\frac{\partial E[d]}{\partial p_1} = \frac{\partial d_n(p_1)^2}{\partial p_1} = 2 \cdot x_{n-1} \cdot p_1 - 2 \cdot x_n \cdot x_{n-1} \tag{2.17}$$

$$= -2 \cdot x_{n-1} \cdot \underbrace{(x_n - p_1 \cdot x_{n-1})}_{=d_n}. \tag{2.18}$$

As explained above, the step direction has to be contrary to this gradient. Furthermore, there is no knowledge on the step *size* that would bring us closest to $p_{1,\text{opt}}$, which is why we need a parameter $\lambda > 0$ that controls the step size. In total, we get a formula for the updated prediction coefficient:

$$p_1(n+1) = p_1(n) - \lambda \cdot \frac{\partial E[d]}{\partial p_1} = p_1(n) + \lambda \cdot x_{n-1} \cdot d(n). \tag{2.19}$$

(Notice that the factor 2 from equation 2.18 has been integrated into λ here.)

Examining Figure 2.5 once more, we see that the absolute value of the gradient $\frac{\partial E[d]}{\partial p_1}$ is proportional to the distance from $p_1(n)$ to $p_{1,\text{opt}}$. We conclude, that equation (2.19) implies a way to apply different step sizes depending on this distance.

A difficult task is to find an optimal setting for λ. Too small values will lead to a slow adaption speed and hence poor prediction. Too high values might for some time instances lead to a step too far in the correct direction and so the new $p_1(n+1)$ leads to an amount of $d(n)^2$ that is even larger than the one for $p_1(n)$. Analytically speaking, this happens when the overall step size $\lambda \cdot x_{n-1} \cdot d(n)$ is twice as large as the distance $|p_1(n) - p_{1,\text{opt}}|$.

All conclusions up to this point were done under the assumption of $N = 1$, but they can be executed in a very similar fashion for $N > 1$ and a general prediction coefficient p_i (see [Hay96, p.266ff]). We get:

$$p_i(n+1) = p_i(n) + \lambda \cdot x_{n-i} \cdot d(n), \qquad \text{for } i = 1, 2, ..., N \tag{2.20}$$

As a simplification, equation (2.20) has a constant value λ for the update of each *LPC* coefficient, where it would be possible to have an own paramater λ_i for each p_i.

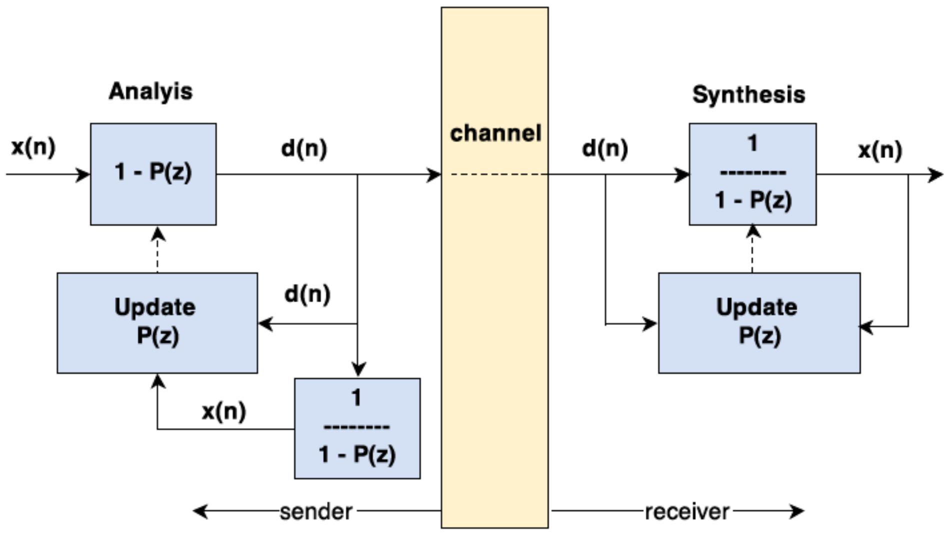

Figure 2.6: *Backward ADPCM*, sample-by-sample processing. Basically a *DPCM* codec, but we have to pass the reconstructed $x(n)$ as well as $d(n)$ to the optimization algorithm. In contrast to the forward *ADPCM*, the channel only has to carry the signal $d(n)$ and no additional data. Diagram adapted from [Nol12].

2.3 Code-Excited Linear Prediction *(CELP)*

As we have seen in the previous section, the error signal that the *LPC analysis produces* not only has minimal energy, it is furthermore decorrelated with the input signal. Hence, for an optimal predictor, the long-term error signal spectrum will be white and as white noise is described by a a fully random and stationary process, no further prediction is possible.

If more data compression is needed than *ADPCM* codecs provide, the Code-Excited Linear Prediction (CELP) allows for that. It is based on the knowledge, that the error signal of *LPC* is close to being white and so audio frames that have run through the *LPC analysis filter* are more similar (spectrum-wise) to each other than unfiltered frames. We infer that we can find a relatively small set of K vectors that is able to represent any frame coming out of the LPC analysis adequately with one of its elements. This set of vectors is called *Codebook* (see [Kon04, p.199]).

2.3.1 Analysis-by-Synthesis

Figure 2.7 shows the processing for a single frame and a single Codebook vector. This is run in a loop over all Codebook vectors to determine which of the vectors produces a synthesized signal frame that is closest to the input signal frame.

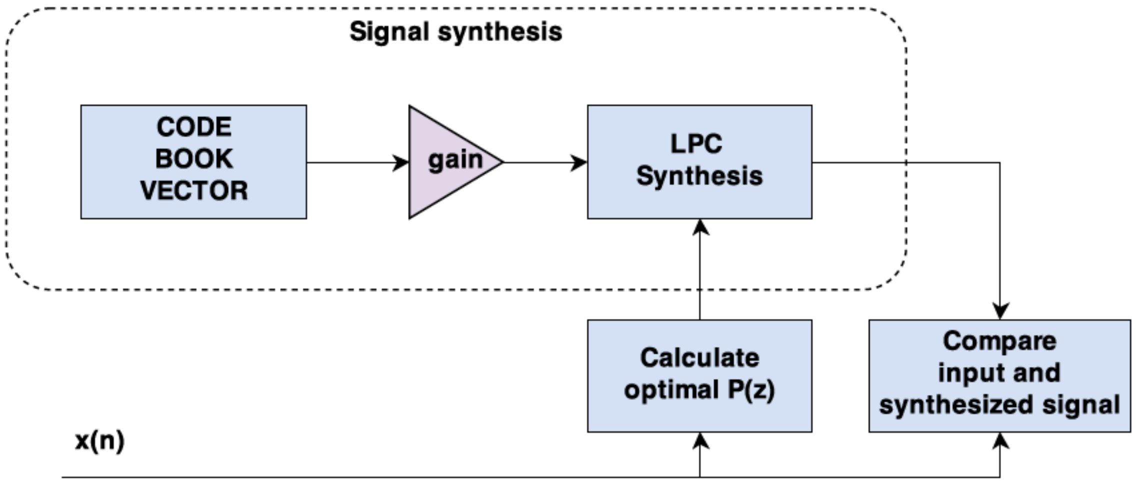

Figure 2.7: CELP, framy-by-frame processing on sender side. This process is **repeated in a loop for each Codebook vector**. The index $\boldsymbol{k}$ of the Codebook vector that produces a synthesized signal frame closest to the input frame gets chosen as optimal and will therefore be transmitted through the channel. Figure adapted from [Kon04, 201].

As can be seen, the algorithm used for the analysis of the optimal Codebook vector and gain already contains a full signal (re-)synthesis and hence, can be used in the same fashion on the receiver side for signal reconstruction. Often, the term *Analysis-by-Synthesis* is used to describe this concept.

2.3.2 Codebooks

Two different types of Codebooks are used in CELP and the two can be used in combination.

Fixed Codebook

A *fixed Codebook* is a constant table of K vectors of the length of the input frames. This table is available to the sender and the receiver side, so that the transmission of the vector index is sufficient to synchronize both sides. Three methods are common to fill the fixed Codebook [Kon04, p.110ff]:

1. Using random values, therefore matching the input vectors with an assumed white spectrum.

2. Finding optimal Codebook vectors in a training phase.
3. Constructing a so-called *sparse Codebook*, which contains multiple zero entries and by that speeds up the Codebook search.

Periodic Codebook

A periodic Codebook is a virtual Codebook, as in an implementation no actual Codebook table is parsed. The underlying idea is, that a pitch might be occurent in the signal after the *LPC analysis* (see [KK89, p. 400]) . So, let us assume that we have a pitch at frequency f_{pitch}. For a sample rate f_s this would correspond to a time periodicity of

$$T_{pitch} = \frac{1}{f_{\text{pitch}}} \tag{2.21}$$

which corresponds to a sample periodicity of

$$P_{pitch} = T_{pitch} \cdot f_s = \frac{f_s}{f_{\text{pitch}}} \tag{2.22}$$

So for a signal with pitch at f_s we would have a good approximation, namely:

$$e[n] \approx e[n - P_{pitch}]. \tag{2.23}$$

For this reason, we construct a virtual Codebook with vectors that are each time shifted variants of the excitation signal for different values of P. Defining this set of vectors as a Codebook, we can then use it for the optimization process shown in Figure 2.7 to find the optimal value of P. The testing range of P should be determined by the selection of a reasonable range of frequencies at which a speaker's pitch can occur.

2.3.3 Source-filter model of speech

The *Signal synthesis* block in 2.7 is an implementation of the source-filter model of speech [Kon04, p.65], which states that human speech generation works in the following way:

1. The vocal chords generate a noise-like excitation signal.
2. For voiced sounds, the vocal chords open and close with the rate of the pitch periodicity, to bring the basic pitch into the sound.

3. The vocal-tract (upper throat and mouth cavity) acts as a filter that shapes the spectral envelope of the excitation signal.

Hence, CELP remodels the source-filter model of speech with the following equivalencies:

- Fixed Codebook $\hat{=}$ Noise generated by vocal chords
- Periodic Codebook $\hat{=}$ Vocal chords open/close periodicity
- LPC Synthesis filter Fixed Codebook $\hat{=}$ Vocal-tract filter

2.3.4 Optimization methods

As mentioned in the caption of Figure 2.7, for each Codebook vector we produce a synthesized signal and try to determine the one that is 'closest' to the input signal. Different approaches are possible for this determination (see [Kon04, p. 208f]).

Maximizing the normalized cross-correlation. One approach to find an optimal Codebook vector is to maximize the normalized cross-correlation between $x(n)$ and $\hat{x}_k(n)$ over the parameter of k, where $\hat{x}_k(n)$ is the synthesized frame with the k-th Codebook vector used for excitation. In formula:

$$\frac{R_{X\hat{X}_k}(0)}{R_{\hat{X}_k\hat{X}_k}(0)} = \frac{x(n)\cdot\hat{x}_k(n)}{\hat{x}_k(n)^2} \overset{!}{=} \text{Maximize over } k \in \{1, 2, ..., K\}. \tag{2.24}$$

This formula produces a white long-term spectrum for $e(n)$: For the optimization process it makes no difference if there is a spectral discrepancy between $x(n)$ and $\hat{x}_k(n)$ at a frequency f_1 or a second frequency $f_2 \neq f_1$. Therefore, errors at all frequencies are weighted in the same way and the long-term spectrum of $e(n)$ is white.

Minimizing the error energy (*LMS*-method). Another approach is a *Least Mean Squares* (LMS) method, trying to minimize the energy of the error signal $e(n)$. In formula (N denotes frame size):

$$E_e = E_{x-\hat{x}_k} \tag{2.25}$$

$$= \frac{\sum_{i=1}^{N}(x(n-i) - \hat{x}_k(n-i))^2}{N} \overset{!}{=} \text{Minimize over } k \in \{1, 2, ..., K\}. \tag{2.26}$$

With the same argument as in the previous section, the outcoming noise will be white.

2.3.5 Perceptual noise shaping

The synthesized signal $\hat{x}(n)$ that CELP produces is not equal to the input signal $x(n)$, because in general none of the Codebook vectors will match the excitation signal exactly. An error noise signal $e(n)$ is added to the input:

$$e(n) = x(n) - \hat{x}(n). \tag{2.27}$$

The spectral shape of $e(n)$ will be determined by the method of finding the optimal Codebook vector. Our goal is to achieve a spectral shape for $e(n)$ that is close to the one of $x(n)$. If that was true, most of the noise energy would occur at frequencies where the energy of $x(n)$ is large, too. Hence, by the principle of auditory masking, the noise would be less audible, while remaining its total energy.

Perceptual noise shaping filters. As in the case without shaping we assume the noise to be white, any noise shaping filter $S(z)$ we apply would lead to a similar spectral shape of the noise itself. As mentioned before, we want this shape to be close to the shape of the signal itself.

The *CELP* processing already provides a filter that describes the spectral envelope of the incoming speech signal, namely the constantly updated *LPC synthesis filter* $\frac{1}{1-P(z)}$. But this filter can have strong spectral peaks which we do not want to see in the noise signal, so we use a smoothed version $\frac{1-P(\frac{z}{\gamma_1})}{1-P(\frac{z}{\gamma_2})}$ of it, where the i-th coefficient $\dot{p}_i$ of $P(\frac{z}{\gamma})$ is calculated with $\dot{p}_i = p_i \cdot \gamma^i$. This implies, that for the corner case $\gamma = 0$ we get $P(\frac{z}{\gamma}) = \mathbf{0}$ and for $\gamma = 1$ we get $P(\frac{z}{\gamma}) = P(z)$. Figure 2.8 shows the frequency responses for different pairs of γ_1 and γ_2 for a given *LPC synthesis filte*r.

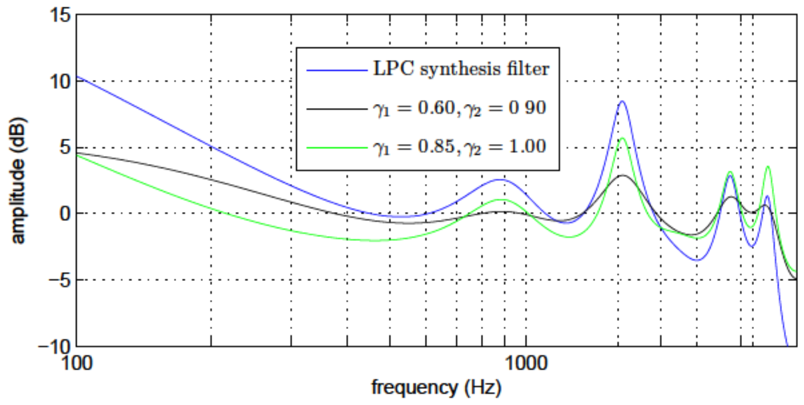

Figure 2.8: Frequency responses of the noise shaping filter $S(z) = \frac{1-P(\frac{z}{\gamma_1})}{1-P(\frac{z}{\gamma_2})}$ for different configurations of γ_1 and γ_2 plotted against the *LPC synthesis filter*. The number of prediction coefficients is 12 at a sampling rate of 8 kHz.

Implementation of perceptual noise shaping As we have seen, the idea of the *LMS* optimization method is to minimize the energy of $e_k(n) = x(n) - \hat{x}_k(n)$ over k and this produces a white spectrum for $e_{k_{opt}}(n)$. But, if we modify the formula and apply $1/S(z)$ to $e_k(n)$ before calculating the mean squares, k will be chosen to synthesize the vector with the least error energy in the domain of $1/S(z)$ or equivalently the most energy in the domain of $S(z)$. This appraoch is explained in [Kon04, p.203ff]

To make understand why this principle works, let us assume that $S(z)$ has a positive peak at frequency f_{peak}. If a Codebook vector produces a synthesis error with a high energy at f_{peak}, then the mean square of $\frac{1}{S(z)} \cdot E_k(z)$ could still be small, because the filter $\frac{1}{S(z)}$ would lower the influence of the value at f_{peak}. So, despite the high error energy of $e_k(n)$ around f_{peak}, its filtered version could still have minimal overall energy and the index k could be chosen as optimal. Hence, this method shifts the error energy to frequencies where the energy of the signal is high as well.

The same principle of noise shaping, that is explained in detail for the *LMS* optimization method, works as well with the cross-correlation method. The equivalency for the two methods is shown in [Kon04, p. 208f].

3. Algorithms under test

This chapter provides a description of the algorithms that were tested for their ability to suppress feedback. The four main approaches are based on the speech coding concepts presented in the previous chapter. Implementational details as well as the specific tweaking modifications that have been introduced to maximize feedback suppression are pointed out for each of the algorithms. Besides a description of the original algorithms there is a brief description of the frequency shifter algorithm that is used as a measurement reference.

At the end of each section, a list of the tested configurations of the specific *algorithm under test* can be found which can be used as references for the *Results* chapter. The value *speech gain* in the configuration tables represents the gain that a certain algorithm applies to a speech signal, i.e. the gain that is occurrent between the input and the output of the algorithm. A negative value suggests, that the algorithm decreases the signal gain and vice versa. This has been measured on 40 speech samples of 3 second length each and for all *algorithm under test* configurations. The mean and variance of the gain that is applied to each of these excerpts is listed in the configuration tables.

General notice on the implemented modules All algorithms under test were implemented as a real-time *JACK*[1] audio module. To be comparable to narrowband *Speex* (see Section 3.2), the processing bandwidth of 4 kHz was chosen for all modules. The frame size was set to 256 samples as this allows to run a fast room impluse response convolution (working only on 2^n frame sizes) which is needed in the measurement simulation framework. For *algorithms under test* that have a different internal processing frame size, a rebuffering was realized within the modules, which

[1]JACK (JACK Audio Connection Kit) provides an Application Programming Interface (API) to write audio processing modules that can run with the audio framework of JACK. A JACK server hosts the processing and is able to load the written JACK applications. The JACK framework provides a simple way to connect the inputs and outputs of the running applications as well as routing audio to the physical connections of the audio interface. For further details, visit http://jackaudio.org/

– in these cases – introduces an additional latency of a single 256 sample frame.

3.1 Original algorithms

3.1.1 Tweaked *ADPCM* (ADPCM-MIX)

As shown in Section *ADPCM* 2.2, the codec provides a way to subtract periodicity from the input signal, namely when applying the analysis filter $1 - P(z)$ to $x(n)$. The resulting signal $d(n)$ has a long-term spectrum that is close to a white spectrum. Feedback tends to be sinusoidal and so the idea was to use the analysis part of the *ADPCM* to attenuate sinusoidal parts of the signal and by this lower the gain of the feedback loop.

Implementation details

The implementation made use of the *backward ADPCM* (see Section 2.2.2), to guarentee for a low-latency codec. The length of $P(z)$ was set to 10. For a fast adaption speed of the filter coefficients, a relatively high value of $\lambda = 0.2$ is used in the implementation. In rare cases this leads to an unstability of the *LPC analysis filter*. As a counteraction, a *forgetting factor* $\varphi = 0.999$ was introduced to formula (2.20) for the prediction coefficient update to ensure stability:

$$p_i(n+1) = \varphi \cdot p_i(n) + \lambda \cdot x_{n-i} \cdot d(n), \qquad \text{for } i = 1, 2, ..., N \tag{3.1}$$

The factor decreases the influence of the old prediction coefficients and by this prevents p from going against infinity.

Test configurations

The implemented *JACK* application provides a slider that mixes the prediction error signal $d(n)$ with the input signal $x(n)$. The mix fader setting will be denoted as *MIX*. The formula for the output signal $y(n)$ of the *ADPCM* application is:

$$y(n) = \text{MIX} \cdot d(n) + (\text{MIX} - 1) \cdot x(n) \quad \text{with } 0 \leq MIX \leq 1. \tag{3.2}$$

For high values of *MIX*, the prediction error signal $d(n)$ will be more present in the output signal $y(n)$. For low values, it will be less present and the untouched speech signal will dominate the output. The tweaking was a process of trying to find a *MIX* value that gives significant raise in stable gain but on the other hand does not touch the perceived audio quality too much.

The settings that were selected for the final measurement are:

	parameter		speech gain [dB]		
algorithm	name	value	mean	var	name
ADPCM	MIX	0.2	-0.1	0.0	ADPCM-MIX0.2
ADPCM	MIX	0.5	-0.1	0.0	ADPCM-MIX0.5

Table 3.1: Tweaked *ADPCM* test configurations

3.1.2 Tweaked *Speex* (SPEEX-MODE)

The *Speex*[2] codec is an open source implementation of *CELP* with several modifications and enhancements to the standard *CELP* approach described above (see [Val06]).

Speex comes with 11 different quality modes where each has a certain channel bandwidth. Besides these 11 quality modes, *Speex* can furthermore run in narrow-band (4 kHz bandwidth) or wide-band (8 kHz bandwidth) mode. To reduce the number of the algorithms working withing *Speex* and so reducing the complexity of the system, it was throughout configured as narrow-band. Notice that for narrow-band mode, some of the 11 quality modes are equal up to the bit level, resulting in a total of 7 unique narrow-band quality modes within *Speex*.

It can be heard, that lower quality modes subtract part of the pitch content of the signal and preliminary tests had shown that this affects the feedback loop. For this reason, three of the native *Speex* quality modes (0, 5, 8) are evaluated in the measurements and furthermore an own mode (2) was configured in a tweaking process.

[2] *"The Speex project has been started in 2002 to address the need for a free, open-source speech codec. Speex is based on the Code Excited Linear Prediction (CELP) algorithm and, unlike the previously existing Vorbis codec, is optimised for transmitting speech for low latency communication over an unreliable packet network."* (Quoted from [Val06]) For further details, visit `http://www.speex.org/`

Speex' implementation of standard *CELP* features

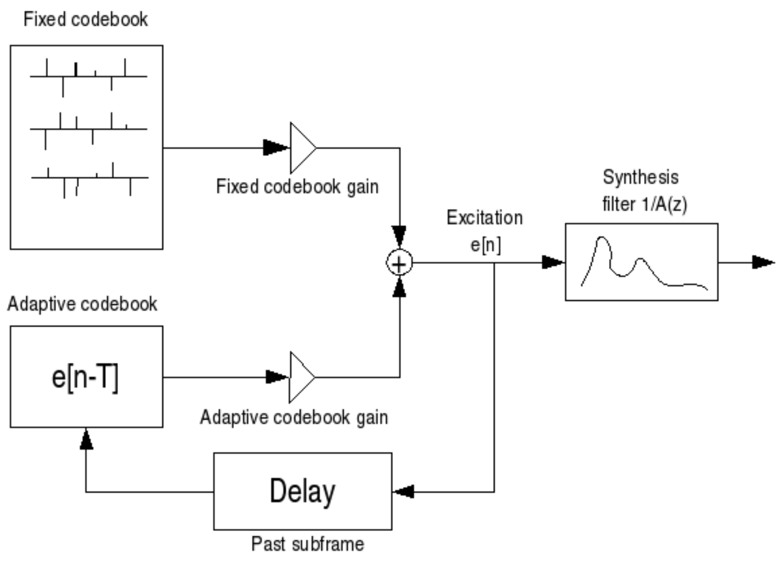

Figure 3.1: Basic concept of signal synthesis in *Speex*. This represents the analysis block of Figure 2.7. The two Codebooks are both represented in the figure: the static (here: fixed) Codebook and the (virtual) periodic (here: adaptive) Codebook. Graphic copied from the official *Speex* paper [Val06, p. 3] which is under public license[4].

- Optimization method: *Speex* utilizes the *LMS* optimization method described in Section 2.3.4.
- Perceptual noise shaping: *Speex* makes use of the noise shaping concepts described in Section 2.3.5. The utilized noise shaping filter is:

$$S(z) = \frac{1 - P(\frac{z}{\gamma_1})}{1 - P(\frac{z}{\gamma_2})}, \qquad \text{with } \gamma_1 = 0.6 \text{ and } \gamma_2 = 0.9 \tag{3.3}$$

Enhancements of *Speex* to standard *CELP* codec

- Multiple pitch gains per frame: *Speex* uses multiple pitch gain values, with a number count from 1 to 3, depending on the quality setting. When using 3 pitch gain values, an optimal pitch period P_{opt} is found and then the estimated excitation signal is build as a linear combination of $e(n - P_{\text{opt}} - 1)$, $e(n - P_{\text{opt}})$ and $e(n - P_{\text{opt}} + 1)$, i.e. also using the adjacent shifted signals for prediction.

[4]Copyright by Jmvalin at en.wikipedia [CC-BY-2.5] http://commons.wikimedia.org/wiki/File:Celp_decoder.svg, from Wikimedia Commons.

- Shortened Codebook: Each *CELP* subframe comes in at a length of 40 samples (5 ms). The common approach is to use a static Codebook with vectors of 40 sample length. *Speex* instead uses Codebooks with a length of 5 samples[5], then dividing the 40 sample subframe into 8 sub-vectors with 5 sample length each and matching these with the vectors of the short Codebook. This reduces the *CPU load* significantly. On the other hand it raises the channel throughput because 8 instead of 1 Codebook vector indices have to be transmitted per subframe.

Test configurations

The implemented *JACK* module utilizes the *Speex* library functions for encoding and decoding a single frame. The application provides a parameter control that can select the *Speex* quality mode, which determines the main configuration parameters of the *CELP* coding procedure. Three of the native *Speex* quality modes were included in the measurements. Notice that *MODE* 2 in contrast, is a tweaked *Speex* configuration, trying to optimize the balance between speech quality and effect on the *maximum stable feedback gain*. This tweaked mode is a mixture of the native quality modes 2 and 3 of *Speex*. The following table shows the main configuration settings in these modes.

algorithm	codebook size	number of pitch gains	pitch gain on level	speech gain [dB]		name
				mean	var	
Speex	0 [6]	1	frame	−3.2	0.6	SPEEX-MODE0
Speex	1024×40	3	frame	−0.4	0.1	SPEEX-MODE2
Speex	512×10	3	subframe	+0.2	0.0	SPEEX-MODE5
Speex	256×5	3	subframe	+0.2	0.0	SPEEX-MODE8

Table 3.2: (Tweaked) *Speex* test configurations

3.1.3 Tweaked original *CELP* (CELP-GAMMA)

Another approach to tweak the feedback suppression effect of a speech codec was to build an own *CELP* implementation from scratch to be able to have a relatively light-weight code base (compared to *Speex*), that can be overseen and configured

[5]This length varies on the setting of the quality mode.

[6]No Codebook is needed, as comfort noise is used as the excitation signal

in a more simple way. Starting point for this original implementation in `C/C++` as a *JACK* application was the *MATLAB CELP* example code written by Sourav Mondal that is accessible on the *MATLAB Central* database [7].

The algorithm operates on 8 kHz sampling rate with a frame size of 160 samples (20 ms) and a subframe size of 40 samples (5 ms). For each frame, the optimal LPC coefficients are calculated with the *Levinson-Durbin recursion*. The subframe analysis is explained in detail in the following section.

Implementation of the subframe analysis

- Optimization method: The implementation utilizes the cross-correlation optimization method described in section 2.3.4.
- Perceptual noise shaping: As in the case of *Speex*, the noise shaping concepts described in Section 2.3.5 are utilized here. The applied noise shaping filter is:

$$S(z) = \frac{1 - P(\frac{z}{\gamma_1})}{1 - P(\frac{z}{\gamma_2})}, \qquad \text{where } \gamma_1 = 0.85 \text{ and } \gamma_2 = 1.0, \tag{3.4}$$

  leading to

$$S(z) = \frac{1 - P(\frac{z}{0.85})}{1 - P(z)}. \tag{3.5}$$

  This relates to the noise weighting filter $W(z)$:

$$W(z) = \frac{1}{S(z)} = \frac{1 - P(z)}{1 - P(\frac{z}{0.85})}. \tag{3.6}$$

  Setting $\gamma_2 = 1$ has a computational advantage. When using *CELP* with perceptual noise shaping, we have to do the following filtering in each round of the CELP analysis-by-synthesis loop:

$$\hat{x}_W(n) = e(n) \cdot H_{\text{Synthesis}} \cdot W(z) = e(n) \cdot \frac{1}{1 - P(z)} \cdot \frac{1 - P(z)}{1 - P(\frac{z}{0.85})}. \tag{3.7}$$

  $\hat{x}_W(n)$ is the full synthesized signal in the noise-weighted domain. As we have

[7] see http://de.mathworks.com/matlabcentral/fileexchange/39038-celp-codec (visited on 2014-11-15)

set $\gamma_2 = 1$, we can now save one of the two filter operations per loop:

$$\hat{x}_W(n) = e(n) \cdot \frac{1}{1 - P(z)} \cdot \frac{1 - P(z)}{1 - P(\frac{z}{0.85})} = e(n) \cdot \frac{1}{1 - P(\frac{z}{0.85})}. \tag{3.8}$$

To shorten the following block diagram, we define

$$A\left(\frac{z}{\gamma}\right) := 1 - P\left(\frac{z}{\gamma}\right), \tag{3.9}$$

which gives us:

$$\hat{x}_W(n) = e(n) \cdot \frac{1}{A(\frac{z}{0.85})}. \tag{3.10}$$

Furthermore, a given input $x(n)$ can be transformed to the noise weighted domain signal $x_W(n)$ as follows:

$$x_W(n) = x(n) \cdot W(z) = x(n) \cdot \frac{1 - P(z)}{1 - P(\frac{z}{0.85})} = x(n) \cdot \frac{A(z)}{A(\frac{z}{0.85})}. \tag{3.11}$$

Figure 3.2 shows the utiliziation of the low-complexity noise shaping in the block diagram of the subframe analysis.

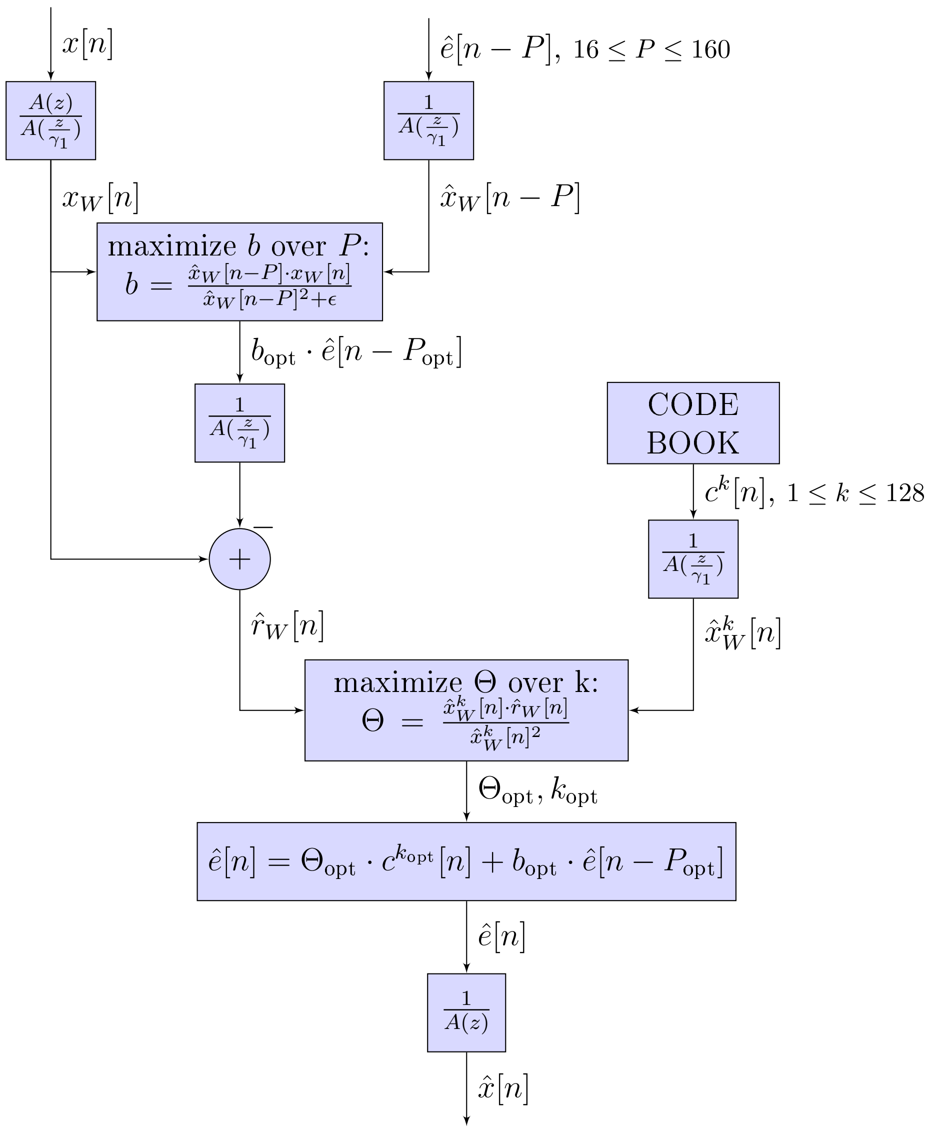
$x[n]$
$\hat{e}[n-P]$, $16 \le P \le 160$
$\frac{A(z)}{A(\frac{z}{\gamma_1})}$
$\frac{1}{A(\frac{z}{\gamma_1})}$
$x_W[n]$
$\hat{x}_W[n-P]$
maximize b over P: $b = \frac{\hat{x}_W[n-P]\cdot x_W[n]}{\hat{x}_W[n-P]^2+\epsilon}$
$b_{\text{opt}} \cdot \hat{e}[n-P_{\text{opt}}]$
CODE BOOK
$c^k[n]$, $1 \le k \le 128$
$-$
$+$
$\hat{r}_W[n]$
$\hat{x}_W^k[n]$
maximize Θ over k: $\Theta = \frac{\hat{x}_W^k[n]\cdot \hat{r}_W[n]}{\hat{x}_W^k[n]^2}$
$\Theta_{\text{opt}}, k_{\text{opt}}$
$\hat{e}[n] = \Theta_{\text{opt}} \cdot c^{k_{\text{opt}}}[n] + b_{\text{opt}} \cdot \hat{e}[n-P_{\text{opt}}]$
$\hat{e}[n]$
$\frac{1}{A(z)}$
$\hat{x}[n]$

Figure 3.2: Original CELP subframe analysis-by-synthesis. The two maximization blocks represent the optimization process for the four parameters per subframe: optimal pitch gain b_{opt}, optimal pitch index P_{opt}, optimal Codebook index k_{opt} and the optimal Codebook gain Θ_{opt}.

To enhance the quality of the synthesized signal, the idea of *Speex* was adapted to use a Codebook that contains vectors smaller than the subframe size (see above). A sub-vector size of 5 samples was chosen and the original Codebook from *Speex* with dimensions 256×5 is used.

Test configurations

The *JACK* application provides a fader for γ_1 to adjust the smoothness of the noise shaping filter where $\gamma_1 = 1$ implies no noise shaping. Figure 2.8 shows an exemplary filter shape for a setting of $\gamma_1 = 0.85$. The tested configurations $\gamma_1 = 0.70$ and $\gamma_1 = 0.85$ are both comparable in the percepted quality of the output signal, even if the difference in the noise between the two settings is audible.

	parameter		speech gain [dB]		
algorithm	name	value	mean	var	name
CELP	γ_1	0.70	+0.5	0.0	CELP-GAMMA$_1$0.70
CELP	γ_1	0.85	+1.3	0.3	CELP-GAMMA$_1$0.85

Table 3.3: Tweaked *CELP* test configurations

3.1.4 Noise shaping mixer (NOISE-GAMMA)

When coding and decoding a signal with *CELP*, it can not be avoided to introduce noise to the signal, because the Codebook is of finite length. The *noise shaping mixer* explained here, emulates the shaped noise that is introduced by the *CELP* algorithms and hence, will be valuable as a comparison in measurements.

Implementation details

As seen above, both *CELP* implementations shape the noise through a constantly updated noise shaping filter

$$S(z) = \frac{1 - P(\frac{z}{\gamma_1})}{1 - P(\frac{z}{\gamma_2})}, \quad \text{with } \gamma_1 < \gamma_2 \tag{3.12}$$

to make the noise less audible (see Section 2.3.5). Therefore, we want an algorithm that generates the same shaping filter $S(z)$. Once we have $S(z)$, we generate white noise, pass it through $S(z)$ and add it to the input signal. This process can be seen in Figure 3.3, which is implemented for measurements as a *JACK* module.

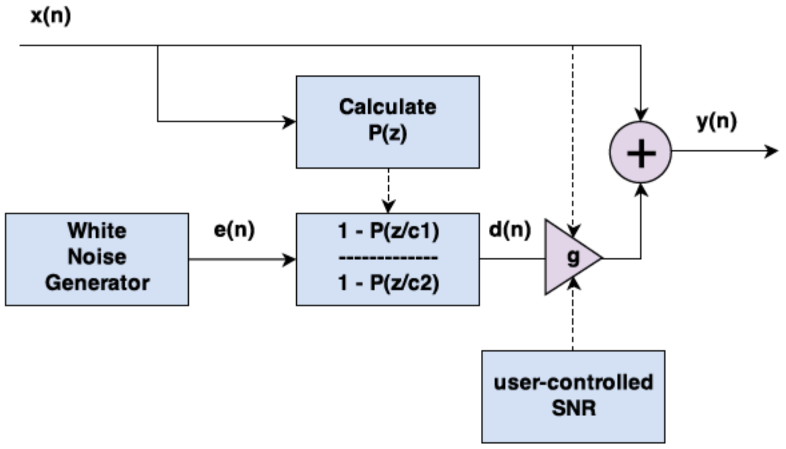

Figure 3.3: Block diagram of the *noise shaping mixer* frame-by-frame processing at a frame size of 160 samples (20ms). The gain value g is dependent on the user controlled SNR and the power of the input signal as described below. Notice that $c_1 = \gamma_1$ and $c_2 = \gamma_2$.

The JACK module provides a slider to set the signal-to-noise ratio (SNR) which controls the gain value that is applied to the added noise in the following way:

$$g = \frac{1}{SNR} \cdot \frac{\sqrt{\sum_{n=1}^{L} x[n]}}{\sqrt{\sum_{n=1}^{L} d[n]}}, \tag{3.13}$$

where $d(n)$ is the shaped noise signal and L denotes the frame length. Equation(3.13) realizes the user-controlled SNR on a frame level, where the SNR is compared on the RMS values. In the experiments, the SNR was constantly set to 2 ($\approx +3\ dB$) and hence leading to:

$$g \approx \frac{1}{2} \cdot \frac{\sqrt{\sum_{n=1}^{L} x[n]}}{\sqrt{\sum_{n=1}^{L} d[n]}}. \tag{3.14}$$

The Gaussian white noise generator is realized by transforming uniformly distributed values provided by the `C++` standard library to normally distributed values using the Box-Muller transform [BM+58].

According to the processing in *CELP*, $P(z)$ is calculated using the *Levinson-Durbin recursion* for each 160 sample (20 ms) frame.

Test configurations

The tested algorithm was set to the shaping filter settings of both the *Speex* codec and the original *CELP* codec from above. This allows us to directly compare the influence of the noise generation between the *noise-only* implementation and the *full-CELP* implementation.

	parameter 1		parameter 2		speech gain [dB]		
algorithm	name	value	name	value	mean	var	name
NoiseShaping	γ_1	0.6	γ_2	0.9	+1.8	0.1	NOISE-GAMMA0.60_0.9
NoiseShaping	γ_1	0.85	γ_2	1.0	+1.8	0.1	NOISE-GAMMA0.85_1.0

Table 3.4: Noise shaping mixer test configurations. Notice that *speech gain* is again the value of the measured gain that is applied between input and output of the *algorithm under test*. The gain value of g in the block diagram is defined as described in Section 3.1.4. The *SNR* was set to +3 *dB* in all experiments.

3.2 Baseline algorithm

3.2.1 Frequency shifter (SHIFTER)

[BH10] report that using a frequency shift algorithm can lead to a significant *increase in maximum stable gain* and that furthermore, this technique does not suppress feedback through a gain modification of the input signal. For this reason the frequency shifter algorithm is used as a reference for the feedback suppression measurements.

Theory

To perform a frequency shift, we can make use the concept of *single sideband modulation (SSB)* which was introduced by [Pet41] and is a refinement of the methods of *amplitude modulation* (*AM*). In *AM*, the input signal $x(n)$ gets multiplied with a carrier wave with frequency $f_{carrier}$. Mathematically, this alone leads to a frequency shift of f_c for any incoming sinusoidal wave.

Still, we can not use pure *AM* for frequency shifting, because the negative part of the spectrum also gets shifted by the value of f_c and so appears in the positve part of the spectrum, on the left hand side to f_c (i.e. in the *left sideband*).

To avoid the appearence of the negative spectrum in the modulated spectrum, the concept of *SSB*, shown in Figure 3.2.1 can be used. As its name indicates, it produces the spectrum only on a single sideband of f_c.

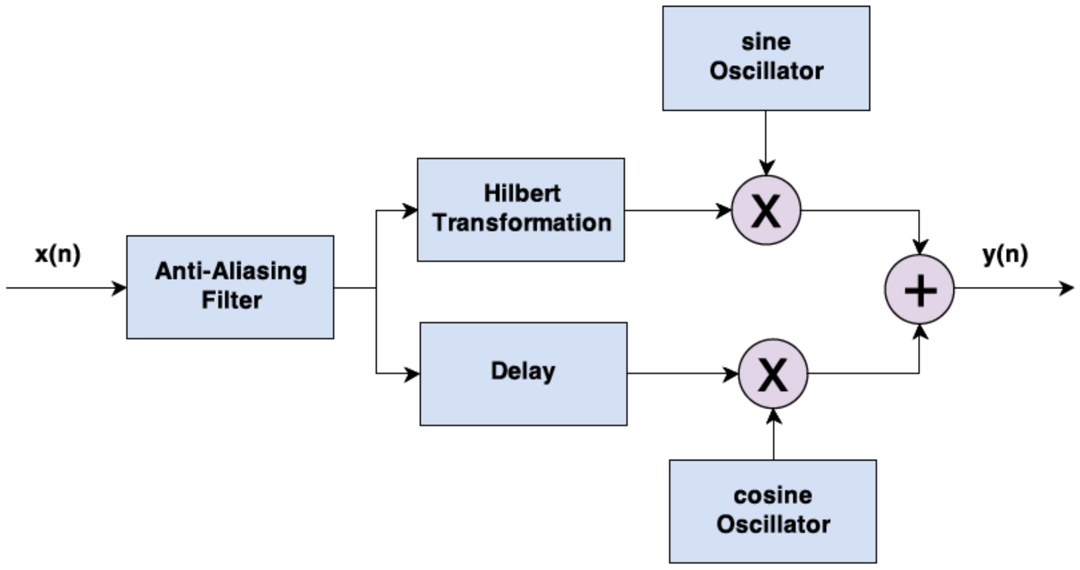

Figure 3.4: Block diagram of the *single sideband modulation* used for frequency shifting. Figure adapted from [ESW06].

Implementation details

The implementation of the algorithm was done by rewriting the *SSB* block diagram in the block diagram interpreter language *FAUST*[8]. It was then compiled as a *JACK* module. The module provides a slider to set the amount of frequency shift, which is accessible via *OSC*[9] control commands.

[8] *"FAUST (Functional Audio Stream) is a functional programming language specifically designed for real-time signal processing and synthesis. It consists in a compiler that translates a FAUST program into an equivalent C++ program, taking care of generating the most efficient code. The FAUST environment also includes various architecture files, providing the glue between the FAUST C++ output and the host audio and GUI environments. The combination of architecture files and FAUST output gives ready to run applications or plugins for various systems, which makes a single FAUST specification available on different platforms and environments without additional cost."* (Quoted from [FOL11]). For further details, visit http://faust.grame.fr/

[9] *"OpenSound Control (OSC) is a protocol for communication among computers, sound synthesizers, and other multimedia devices that is optimized for modern networking technology. OSC has achieved wide use in the field of computer-based new interfaces for musical expression for wide-area and local-area networked distributed music systems, inter-process communication, and even within a single application."* (Quoted from [WFM03]). Visit http://opensoundcontrol.org/for further details.

Test configurations

The value of the pitch shift can be set by a fader in the *JACK* application. The values 2, 4 and 6 *Hz* were chosen for baseline testing as [BH10] also ran tests on these settings and and furthermore note that the audio quality is reasonable within this range.

	parameter		speech gain [dB]		
algorithm	name	value	mean	var	name
FrequencyShifter	SHIFTHZ	2	0.0	0.0	SHIFTER-HZ2
FrequencyShifter	SHIFTHZ	4	0.0	0.0	SHIFTER-HZ4
FrequencyShifter	SHIFTHZ	6	0.0	0.0	SHIFTER-HZ6

Table 3.5: Frequency shifter test configurations

4. Measurement setup

This chapter explains the evaluation procedure of the *algorithms under test*. Therefore, a concept of measuring the *increase in maximum stable gain* is established. This concept is realized in a software simulation framework as well as in the conducted live tests in real rooms. The exact implementation of the concept in both variants is explained here in detail.

4.1 Concept of measuring the *increase in maximum stable gain* (*IMSG*)

To measure the *increase in maximum stable gain* (*IMSG*), we first need to define a method to measure the *maximum stable gain* (*MSG*) of a system. For this task we need an amplifier with a gain control knob. With this knob, the specific maximum gain value $\hat{g}$ has to be found, at which the system is still stable. *Stable* here means, that the energy in an early time frame $[t_1, t_2]$ should be equal to or higher as in a later time frame $[t_3, t_4]$ in any part of the system. Further, for every gain value $g \geq \hat{g}$ the system has to be unstable. For a system with an inserted *algorithm under test* named A, the *MSG* that fulfills these conditions will be noted as $\hat{g}_A$.

For measuring the *IMSG*, we will compare the *algorithm under test* A to the exact same system, but with the difference of the *algorithm under test* bypassed or - equivalently - replaced by a Dirac filter $\delta(0)$. This leads to the **formula for the *increase in maximum stable gain (IMSG)*:**

$$\Delta\hat{g}_A = \hat{g}_A - \hat{g}_{\delta(0)}. \tag{4.1}$$

The important notion that follows from these considerations is: For determining the *IMSG* by experiment, we need a method to determine the *MSG* of a system and a subtraction after two measurements will do the rest to calculate the *IMSG*.

4.2 Simulation framework

As measurements in real rooms are time-consuming, especially the tweaking process of this work created the need for a secondary and more time-efficient method for proper evaluation of the implemented algorithms. Therefore, a software framework was developed to carry out measurements by using pre-recorded impulse responses for simulating various speaker and room characterics. Besides the beneficial aspect of time efficiency, simulations also provide a higher amount of repeatability than real-world experiments.

4.2.1 Experimental setup

The leading thought for the experimental setup of the simulated measurements was to be as close as possible to a real-world measurement scenario. Therefore, the *algorithms under test* still had to run within the *JACK* framework as coded, so recoding them in another language was not considered as an option.

This constraint avoids a different behaviour of the algorithms under the different testing conditions (simulation/real world). Developing two programs in different programming languages that are equal up to the bit-levels is hard, e.g. due to differences in the length of elementary data types. As the algorithms are time-varying, proving equal behavior of two versions that are not bit-exact is also difficult.

For that reason, the decision was made to simulate the whole audio system within the *JACK*[1] framework and to use *MATLAB* as a tool to supervise, control and evaluate the measurements being run within *JACK*. By using the *Open Sound Control (OSC)*[2] communication protocol, *MATLAB* was successfully utilized to control parameters of the implemented *JACK* modules.

Figure 4.1 shows the structure of the evaluation framework and its basic modules, which should be explained briefly.

[1]see http://jackaudio.org/
[2]see http://opensoundcontrol.org/

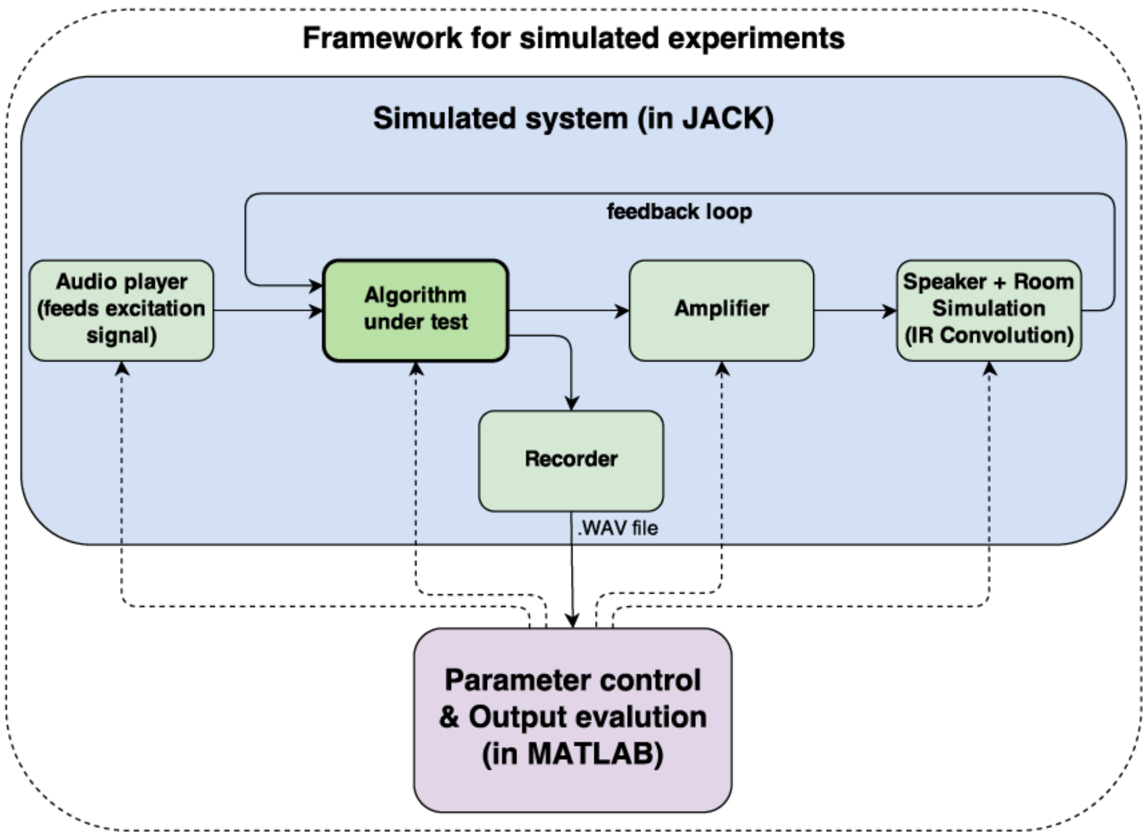
Framework for simulated experiments
Simulated system (in JACK)
feedback loop
Audio player (feeds excitation signal)
Algorithm under test
Amplifier
Speaker + Room Simulation (IR Convolution)
Recorder
.WAV file
Parameter control & Output evalution (in MATLAB)

Figure 4.1: Diagram of the simulation framework. Green blocks represent *JACK* modules, regular lines represent audio paths and dashed lines represent paths for *OSC* control signals. The whole audio system runs within the *JACK* framework which allows for connection management of the modules. *MATLAB* controls all parameter settings of the applications via *OSC* and evaluates the output of the system which is delivered from the *JACK* recorder module in *.WAV*-files.

Audio player. A self-written and lightweight *JACK* module that streams a specified *.WAV*-file frame-by-frame into the *JACK* framework. Its single *OSC* control parameter is an on/off switch, which starts and stops the output. The *audio player* is used to feed the system with an initial excitation signal. The open source library LIBSNDFILE [3] was used for loading the *.WAV*-files into the module.

Amplifier. Has a gain control and an on/off switch. If turned to *ON*, the chosen gain is applied to the signal, otherwise the output is all zeros. The on/off switch is needed to reset the system between two measurements.

[3]see http://www.mega-nerd.com/libsndfile/

Microphone, speaker & room simulation. Loads an impulse response (IR) and convolutes it with the signal to achieve a microphone, speaker and room simulation. As real-time *IR* convolution is a hard problem which is not within the core of this thesis, the open-source program *LV2 Convolution Reverb*[4] written by Tom Szilagyi was used for this task. It allows for fast convolution by transferring the signal to the frequency domain (using the open source library *FFTW*[5]) and hence, replacing the expensive convolution in the time domain with a more efficient multiplication in the frequency domain.

The program is written as an audio plugin under the *LV2* standard. Therefore, the program *Lv2Rack* was used as a wrapper to make the *LV2 Convolution Reverb* accessible within the JACK framework.

For each *IR*, a configuration file for the *LV2Rack* had to be created. Stored configurations can be reloaded to the rack via the system command line and by this, *MATLAB* is able to automatically load a various speaker/room configurations, if a list with the configuration files is provided.

Recorder. The recorder module is needed to let *MATLAB* listen to sinks and sources within the *JACK* framework. The recorder has an on/off switch and writes the current recording in its buffer to a file on disk when turned *OFF*. Again, the LIBSNDFILE library was used for the implementation of this *JACK* module, in this case for writing the *.WAV*-files.

Measurement procedure for a fixed gain value

To determine the system stability for a single parameter setting and one fixed gain value, the following tasks are executed by *MATLAB*:

1. Set *algorithm under test*'s parameters, set current gain in *amplifier* (both via *OSC* signals) and load the speaker/room configuration to the *IR convolution engine* (via shell command).

2. Feed the excitation signal - a three second speech excerpt - with the *player* to the system. The *player* is turned to *ON* via *OSC*.

3. Record the system for another 25 seconds[6] after the excitation signal was

[4]see `http://factorial.hu/plugins/lv2/ir/`

[5]For details, confer [FJ98] and `http://www.fftw.org/`

[6]This is the time needed for real-time experiments. *JACK* can be run in so called *freewheel mode* which means that it runs faster than realtime. Depending on the real-time *CPU load* of the *algorithm under test*, the time of 25 seconds was reduced up to 0.5 seconds.

played. *Recorder* is turned on and off via *OSC* commands.

4. Read the recorded *.WAV*-file and compare the energy P_{late} of a late time frame of the signal against the energy P_{early} of an early time frame. If $P_{early} \geq P_{late}$, then the system is stable for the current gain, otherwise the system is unstable. (For reasonability of this criterion, see Section 4.1)

Measurement procedure to find the *maximum stable gain*

The goal is to find the *MSG* value within a tolerance of $\pm 0.1\ dB$. For this task, we need to run the stability test procedure described in the previous section multiple times for different gain values while keeping every other step fixed in the whole process.

To find the *MSG*, we have to find two gain values g_1 and g_2, with $g_2 = g_1 + 0.1\ dB$ where the system is still stable for g_1 but unstable for g_2. This would directly show, that g_1, is the *MSG* for this specific setting within the specified tolerance.

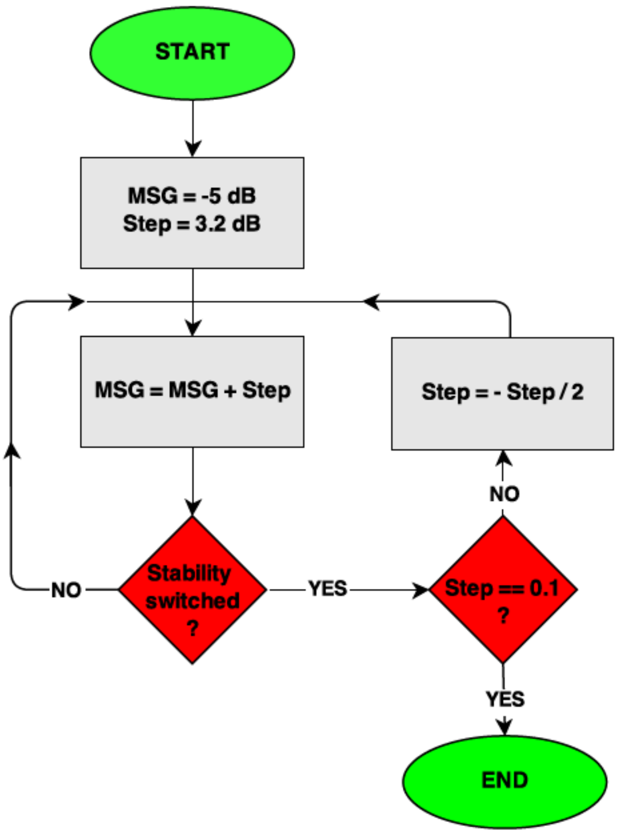

Figure 4.2: Block diagram describing a quick way to find the maximum stable gain (*MSG*) algorithmically.

To find this pair of g_1 and g_2, the algorithm described in the Figure 4.2 was implemented. Basically, we start with a certain *step size* and a certain gain value for that we are sure that the system is stable. Once started, we check for stability (see steps explained above) of the system with the current gain and keep raising the gain step by step. As long as the system keeps in the same state (stable/unstable), we keep going in the same direction, waiting for the system to switch its state. When the system actually switches, we invert the step direction (see minus in formula), halfen the *step size* and wait again for the system to switch and so on and so on. By this method, we guarentee, that after 5 state switches the *step size* is as small as 0.1 dB and this will just happen when we are stepping around the pair of g_1 and g_2.

4.3 Testing live in real rooms

As there may always be weaknesses in simulated experiments, it is necessary to verify the results of the simulation with a live test in real rooms. The presumably most important difference between simulation and the live test in this experiment is a time-varying room transfer function, e.g. due to temperature changes. Acoustical feedback is a fragile phenomenon and could hence be influenced by tiny, but steady variations in the room transfer function. The setup of the live test is explained in the following paragraphs.

4.3.1 Experimental setup

The concept of the measurement setup was built up on the idea of flexibility, repeatability and to be as close as possible to a real world use-case. These aims led to the setup shown in Figure 4.3.

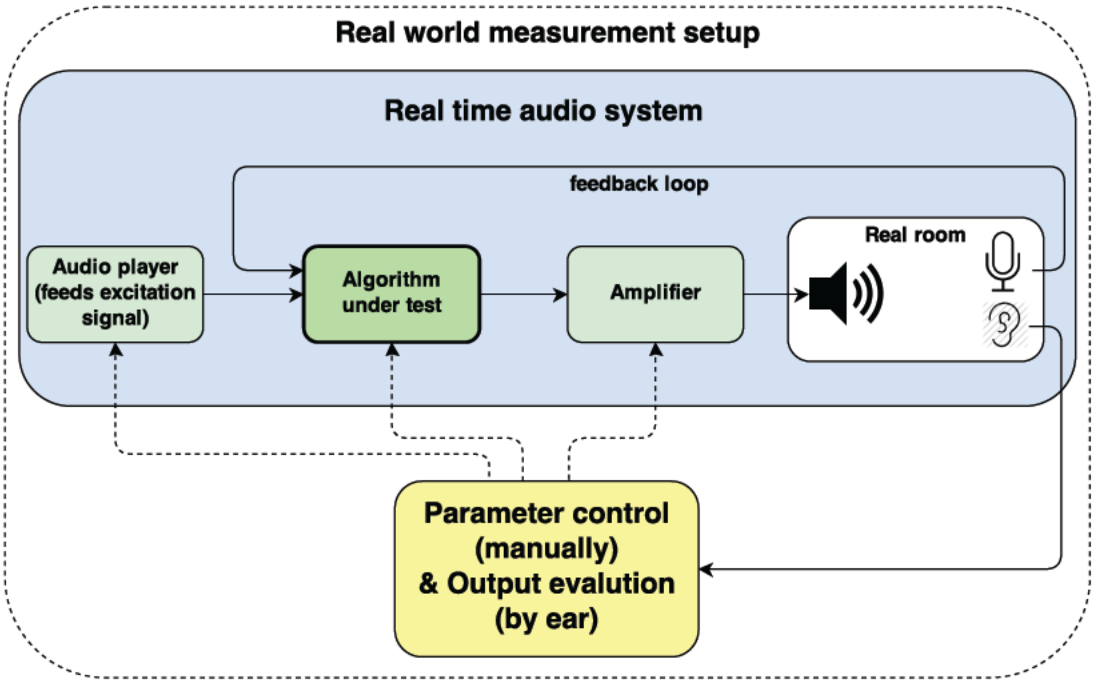
Real world measurement setup
Real time audio system
feedback loop
Audio player (feeds excitation signal)
Algorithm under test
Amplifier
Real room
Parameter control (manually) & Output evalution (by ear)

Figure 4.3: Diagram of the real-world measurement setup. Green blocks represent *JACK* modules, regular lines represent audio paths and dashed lines represent paths for parameter control. The whole audio processing runs within the *JACK* framework which handles the connection management and provides access to physical input and output of the sound card. The parameter setting and finding of the *MSG* is done manually.

Several parts of the audio chain were realized within the *JACK* framework. These parts are:

- the excitation signal feed run by the *audio player*,
- the *algorithm under test*,
- the *amplifier* which allows to adjust the gain. Notice that there was an external amplifier used for actual electrical amplification of the signal, but this amplifier was set to one fixed gain setting.
- The handling of all internal audio connections,
- management of physical input and output.

(For a brief description of all utilized *JACK* modules, see section Section 4.2.1.) Using the *audio player* to feed the system with the exact same excitation signal for each test run increases the repeatability compared to an actual human speaker in the room used for system excitation. Using a digital gain control through the *amplifier* application, allows for exact control of the volume knob (in our case up to ± 0.1 dB) and so avoids the need for an expensive, external amplifier with a sensitive volume knob and display. Compared to the speaker, microphone and room, an amplifier is the most linear part of an audio chain when used within a reasonable dynamic range, which should allow us to replace it by a digital model, even if we want to match a real world scenario.

Besides this digital processing, the rest of the audio chain was realized in real world conditions, including the following parts:

- speaker,
- room,
- microphone.

Process of finding the *MSG* for a fixed setting

To determine the *MSG* for a specific setting the following iterative procedure was executed:

1. Set initial *amplifier* gain.
2. Feed excitation signal from *player* to system.
3. Wait about 10 seconds after excitation and listen if system gets unstable.
4. If the system is stable, raise *amplifier* gain. If it is unstable, decrease *amplifier* gain.
5. Repeat steps 2 to 4 until two gain values $\hat{g}_1$ and g_2 have been found, with $g_2 = \hat{g}_1 + 0.1dB$ and where the system is still stable for $\hat{g}_1$ but unstable for g_2.

4.3.2 Measurement equipment and configurations

The following audio equipment was used to carry out the measurements:

- *E-MU 0404 USB* audio interface
- *Technics SA-AX540* stereo amplifier
- *Teufel Ultima 40* speaker

Furthermore, the *MSG* for all algorithms under test was evaluated for:

- 2 different microphones
  1. *Sennheiser E835* cardioid, dynamic, hand-held microphone
  2. *Shure SM58* cardioid, dynamic, hand-held microphone

- 2 different rooms
  1. Medium-sized conference room (short name: medium)
     ($4m \times 6m \times 3m$, $V = 72\ m^3$, $T_{60} = 0.7\ s$)
  2. Large lecture hall (short name: large)
     ($21m \times 29m \times 7m$, $V = 4623\ m^2$, $T_{60} = 1.6s$)

- 3 microphone postions per room

This results in a total of $2 \times 2 \times 3$ data points per test configuration.

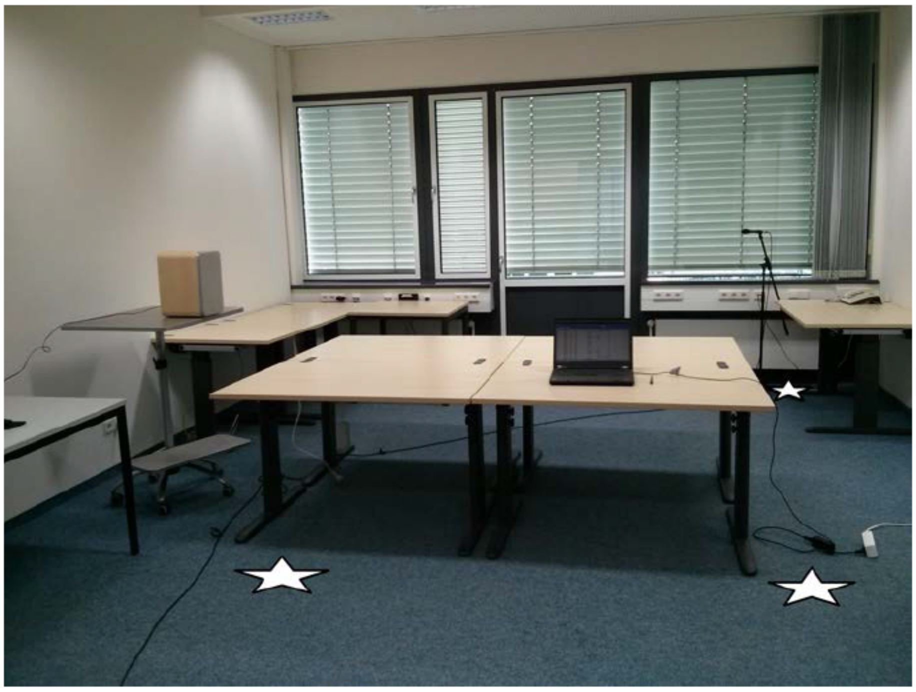

Figure 4.4: Conference room with marked microphone positions.

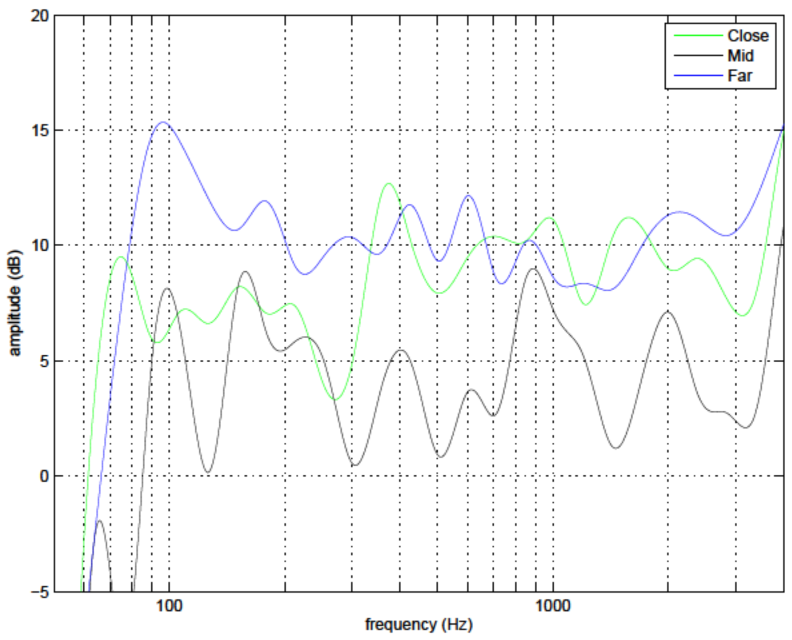

Figure 4.5: **Smoothed** room impulse responses for three microphone positions in the conference room (all measured with *Sennheiser E835*).Notice that a 1/2-octave smoothing has been applied to the curves to enhance the readability.

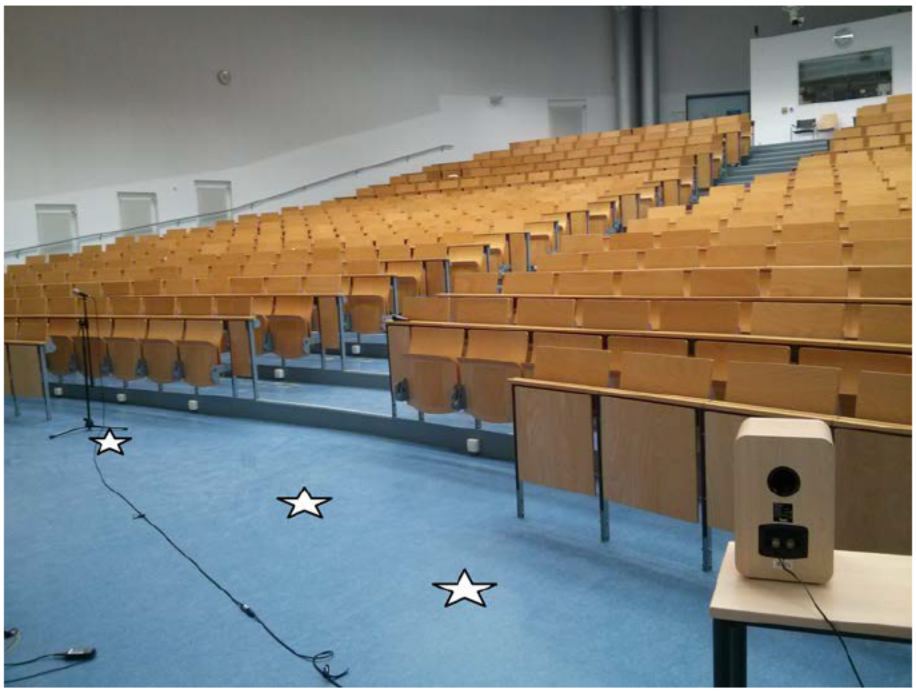

Figure 4.6: Large lecture hall with marked microphone positions.

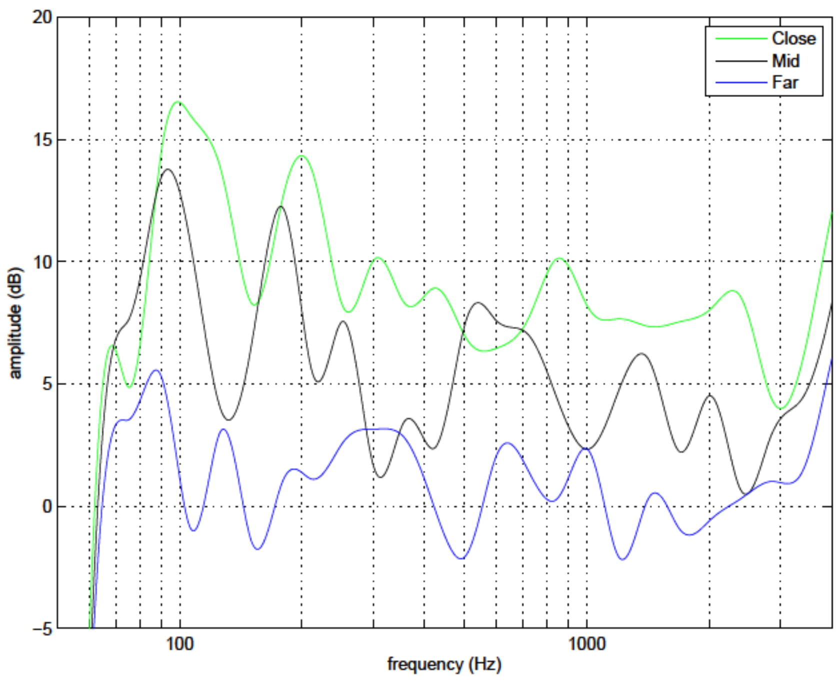

Figure 4.7: **Smoothed** room impulse responses for three microphone positions in the lecture hall (all measured with *Sennheiser E835*). Notice that a 1/2-octave smoothing has been applied to the curves to enhance the readability.

Additional impulse responses were recorded for one more room – a small living room – at three different positions and with 2 different microphone models to have another room model available for testing within the simulation framework. Figure 4.8 shows the frequency response at three different positions of the *Sennheiser E835* microphone. The room's size is $4m \times 3m \times 2.5m$ and the reverberation time is $T_{60} = 0.35s$.

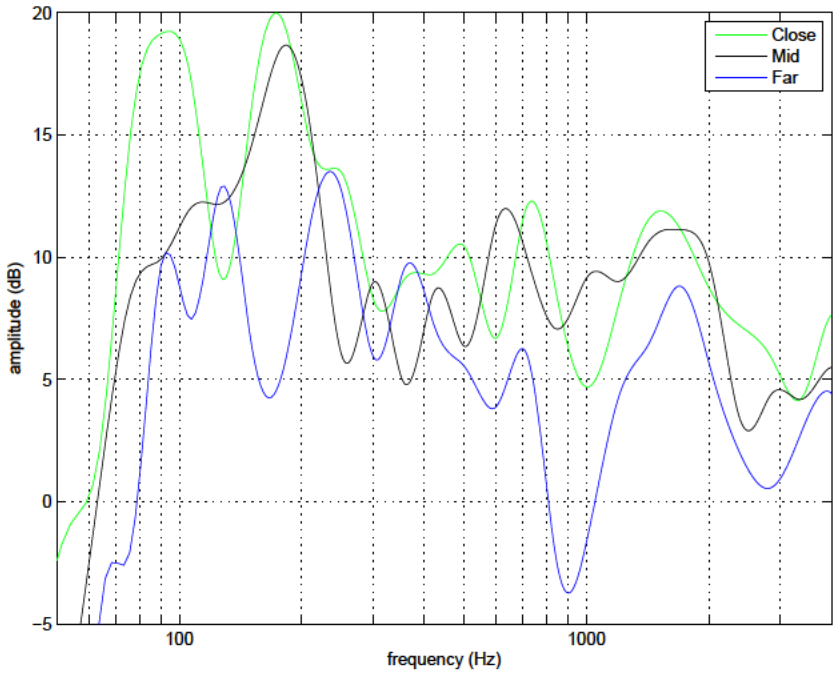

Figure 4.8: **Smoothed** room impulse responses for three microphone positions in the small living room (all measured with *Sennheiser E835*) that is used in the simulation measurements only. Notice that a 1/2-octave smoothing has been applied to the curves to enhance the readability.

5. Results

This chapter provides the results of the live test session as well as the simulation measurements described in the previous chapter. After the presentation of the data, a discussion is attached. But first, a few words about how the data is presented.

Result tables The result tables list the mean values of the *increase in maximum stable gain* (*IMSG*) in *dB* for a certain *algorithm under test* configuration and for all data points that fall under the specified category. Notice that the *speech gain* for each *algorithm under test* (which can be found in the configuration tables in Chapter 3) was removed as a bias from the measurement data of each configuration.[1]

As in total there are 12 data points for each of the 13 *algorithm under test* configurations, the number of data points that go into the mean and standard devitation statistics range from 4 (for three groups) to 12 (overall statistics).

Furthermore, the following hypothesis was tested for significance on the overall results:

- H_1: *"The mean value of the increase in maximum stable gain is greater than 0"*

Which gives us the counter-hypothesis:

- H_0: *"The mean value of the increase in maximum stable gain is less or equal to 0"*

The statistical significance p (i.e. the probability of H_0) was calculated under the untested assumption, that the values follow the *Gaussian normal distribution.*

[1] If for example the actually measured *IMSG* was 3.4 *dB* for an algorithm with a speech gain bias of -1.2 *dB* (i.e. it lowers the volume of the speech signal), in that case, the resulting *IMSG* is presented in the result tables and plots as $4.6\ dB = 3.4\ dB - (-1.2\ dB)$. This is done, because in an actual application of the algorithm, we would also compensate at some place for the gain loss introduced by the algorithm.

Result plots The plots visualize the mean and the unbiased standard deviation listed in the tables. The mean values are marked with an 'o' symbol and the unbiased standard deviation is represented by the length of the bar around the mean. The total length of each bar matches exactly two unbiased standard deviations (measured from one to the other end of a bar).

5.1 Live tests

5.1.1 Overall results

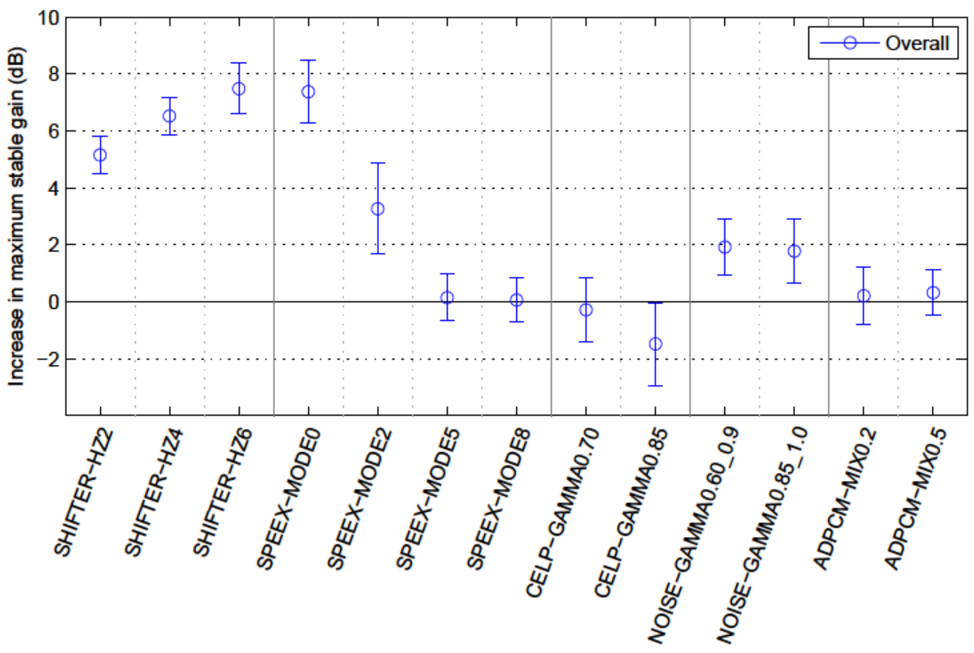

Figure 5.1: Increase in maximum stable gain in [dB]. Mean and standard deviation plot for overall results of live tests.

Algorithm under test	Mean	StDev	$p(H_0)$
SHIFTER-HZ2	+5.2	0.7	0.00
SHIFTER-HZ4	+6.5	0.7	0.00
SHIFTER-HZ6	+7.5	0.9	0.00
SPEEX-MODE0	+7.4	1.1	0.00
SPEEX-MODE2	+3.3	1.6	0.00
SPEEX-MODE5	+0.1	0.8	0.23
SPEEX-MODE8	+0.1	0.8	0.23
CELP-GAMMA$_1$0.70	**−0.3**	1.1	0.98
CELP-GAMMA$_1$0.85	**−1.5**	1.5	1.00
NOISE-GAMMA0.60_0.9	+1.9	1.0	0.00
NOISE-GAMMA0.85_1.0	+1.8	1.1	0.00
ADPCM-MIX0.2	+0.2	1.0	0.03
ADPCM-MIX0.5	+0.3	0.8	0.02

Table 5.1: Increase in maximum stable gain in [dB]. Overall results of live tests.

5.1.2 Results by room

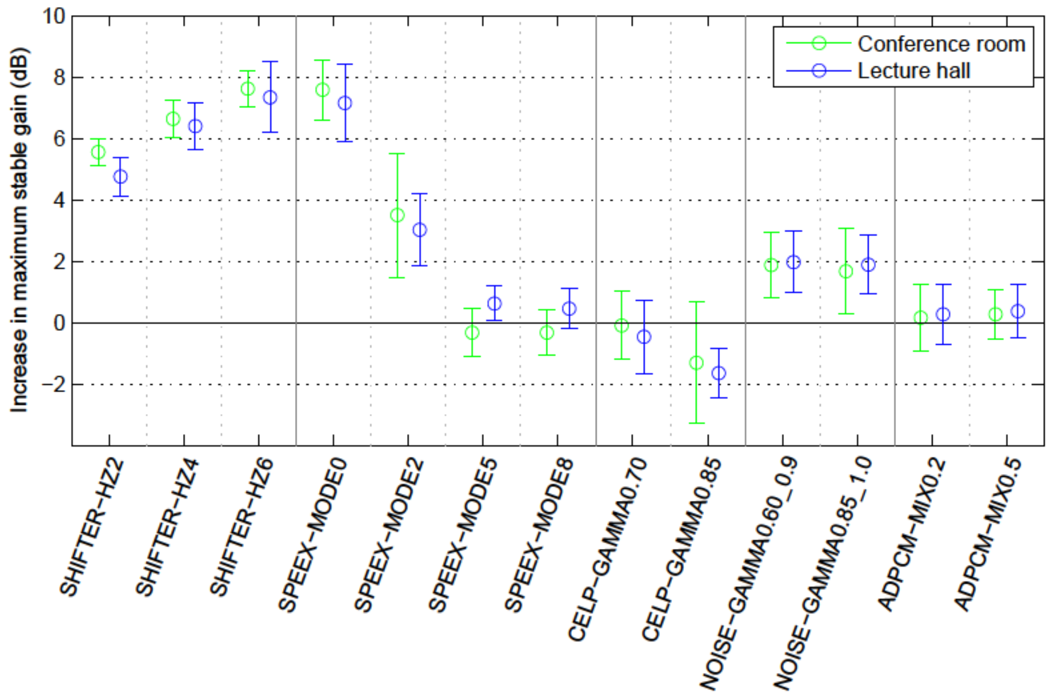

Figure 5.2: Increase in maximum stable gain in [dB]. Mean and standard deviation plot for live test results separated by room.

Room	Conference		Lecture hall	
Algorithm under test	Mean	StDev	Mean	StDev
SHIFTER-HZ2	+5.6	0.4	+4.8	0.6
SHIFTER-HZ4	+6.6	0.6	+6.4	0.8
SHIFTER-HZ6	+7.6	0.6	+7.3	1.2
SPEEX-MODE0	+7.6	1.0	+7.2	1.3
SPEEX-MODE2	+3.5	2.0	+3.0	1.2
SPEEX-MODE5	−0.3	0.8	+0.6	0.6
SPEEX-MODE8	−0.3	0.7	+0.5	0.6
CELP-GAMMA$_1$0.70	−0.1	1.1	−0.5	1.2
CELP-GAMMA$_1$0.85	−1.3	2.0	−1.7	0.8
NOISE-GAMMA0.60_0.9	+1.9	1.0	+2.0	1.0
NOISE-GAMMA0.85_1.0	+1.7	1.4	+1.9	1.0
ADPCM-MIX0.2	+0.2	1.1	+0.3	1.0
ADPCM-MIX0.5	+0.3	0.8	+0.4	0.9

Table 5.2: Increase in maximum stable gain in [dB]. Live test results separated by room.

5.1.3 Results by microphone model

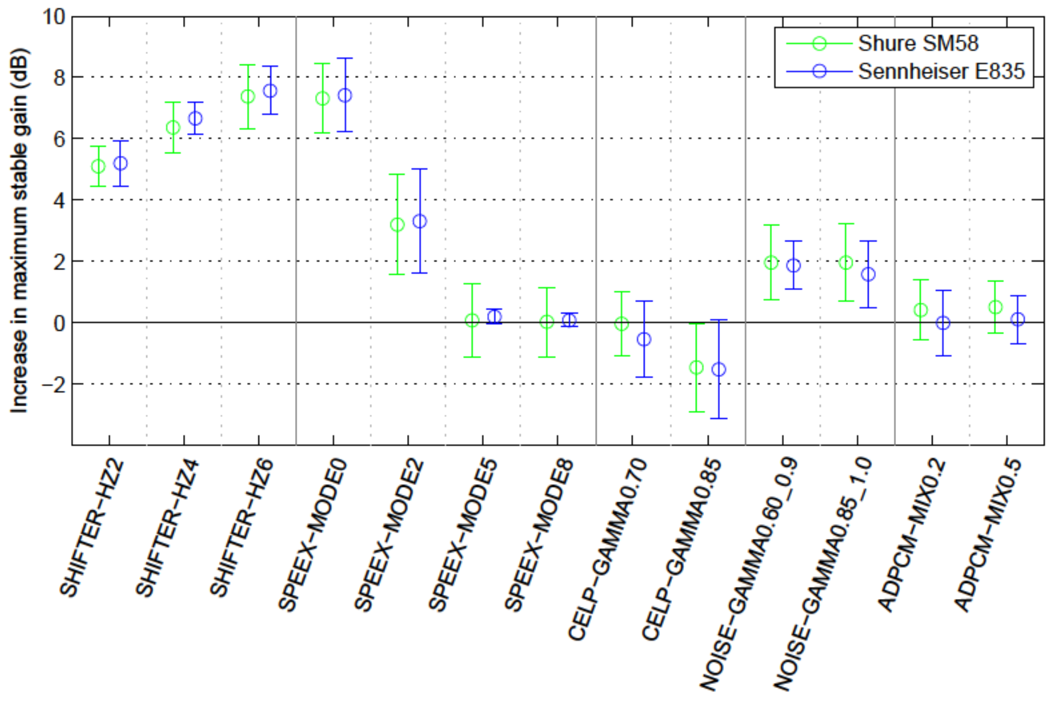

Figure 5.3: Increase in maximum stable gain in [dB]. Mean and standard deviation plot for live test results separated by microphone. model.

Microphone	Shure SM58		Sennheiser E835	
Algorithm under test	Mean	StDev	Mean	StDev
SHIFTER-HZ2	+5.1	0.6	+5.2	0.7
SHIFTER-HZ4	+6.4	0.8	+6.7	0.5
SHIFTER-HZ6	+7.4	1.0	+7.6	0.8
SPEEX-MODE0	+7.3	1.1	+7.4	1.2
SPEEX-MODE2	+3.2	1.6	+3.3	1.7
SPEEX-MODE5	+0.1	1.2	+0.2	0.2
SPEEX-MODE8	+0.0	1.1	+0.1	0.2
CELP-GAMMA$_1$0.70	+0.0	1.1	**−0.5**	1.2
CELP-GAMMA$_1$0.85	**−1.5**	1.4	**−1.5**	1.6
NOISE-GAMMA0.60_0.9	+2.0	1.2	+1.9	0.8
NOISE-GAMMA0.85_1.0	+2.0	1.3	+1.6	1.1
ADPCM-MIX0.2	+0.4	1.0	+0.0	1.1
ADPCM-MIX0.5	+0.5	0.8	+0.1	0.8

Table 5.3: Increase in maximum stable gain in [dB]. Live test results separated by microphone model.

5.1.4 Results by microphone-to-speaker distance

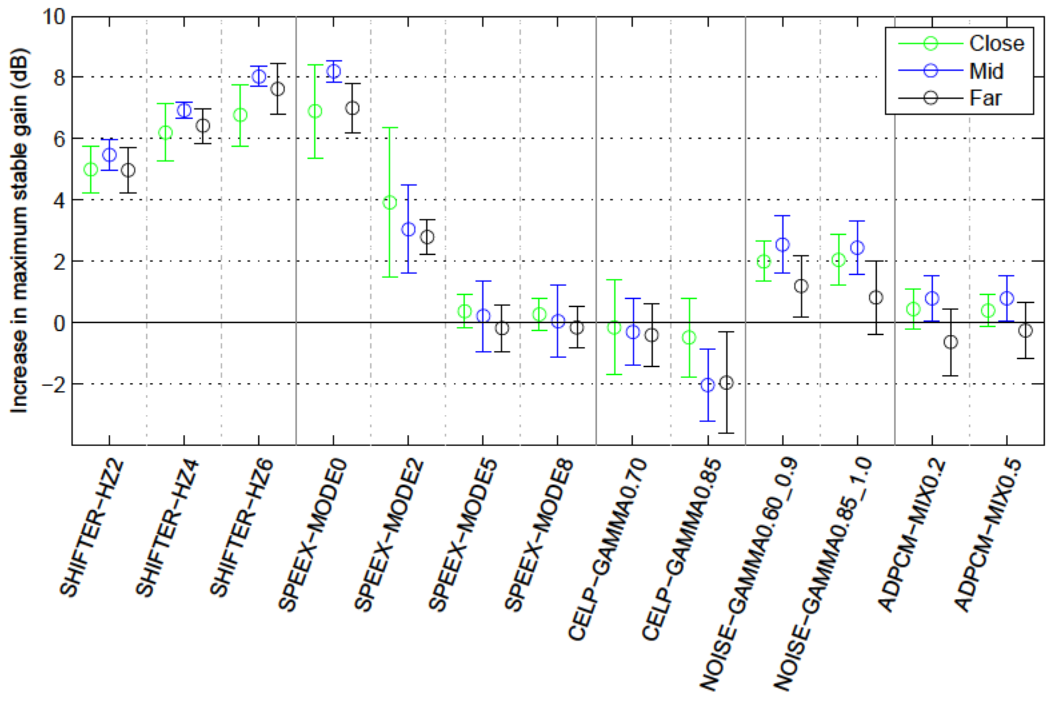

Figure 5.4: Increase in maximum stable gain in [dB]. Mean and standard deviation plots for live test results separated by microphone-to-speaker distance.

Distance	Close		Mid		Far	
Algorithm under test	Mean	StDev	Mean	StDev	Mean	StDev
SHIFTER-HZ2	+5.0	0.8	+5.5	0.8	+5.0	0.7
SHIFTER-HZ4	+6.2	0.9	+6.9	0.9	+6.4	0.6
SHIFTER-HZ6	+6.8	1.0	+8.0	1.0	+7.6	0.8
SPEEX-MODE0	+6.9	1.5	+8.2	1.5	+7.0	0.8
SPEEX-MODE2	+3.9	2.5	+3.1	2.5	+2.8	0.6
SPEEX-MODE5	+0.4	0.5	+0.2	0.5	**−0.2**	0.8
SPEEX-MODE8	+0.3	0.5	+0.1	0.5	**−0.2**	0.7
CELP-GAMMA$_1$0.70	**−0.2**	1.5	-0.3	1.5	**−0.4**	1.0
CELP-GAMMA$_1$0.85	**−0.5**	1.3	-2.0	1.3	**−2.0**	1.6
NOISE-GAMMA0.60_0.9	+2.0	0.7	+2.6	0.7	+1.2	1.0
NOISE-GAMMA0.85_1.0	+2.1	0.8	+2.5	0.8	+0.8	1.2
ADPCM-MIX0.2	+0.5	0.7	+0.8	0.7	**−0.6**	1.1
ADPCM-MIX0.5	+0.4	0.5	+0.8	0.5	**−0.3**	0.9

Table 5.4: Increase in maximum stable gain in [dB]. Live test results separated by microphone-to-speaker distance.

5.2 Software simulated measurements

5.2.1 Simulation results for additional room

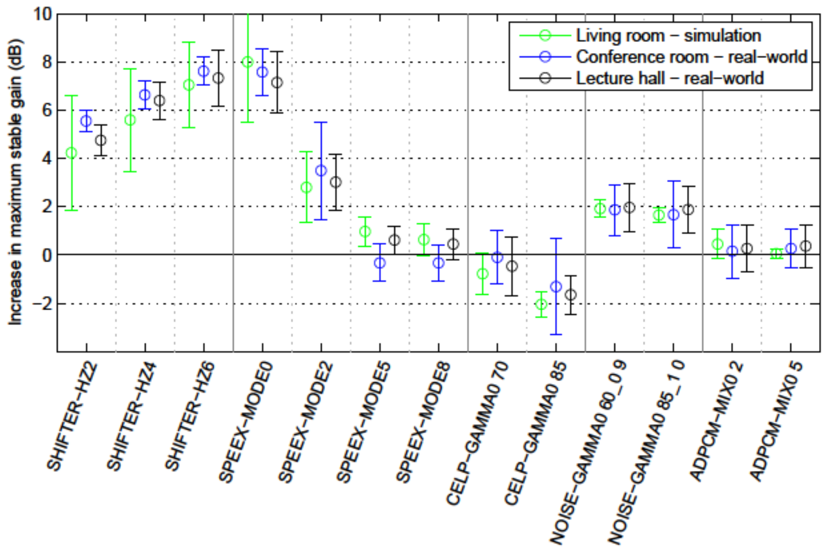

Figure 5.5: Increase in maximum stable gain in [dB]. Mean and standard deviation plot for results of simulated measurements for the living room condition. For visual comparison, the plot also shows the results for the rooms utilized in live tests.

Room	Living room	
Algorithm under test	Mean	StDev
SHIFTER-HZ2	+4.3	2.3
SHIFTER-HZ4	+5.6	2.1
SHIFTER-HZ6	+7.0	1.7
SPEEX-MODE0	+8.0	2.5
SPEEX-MODE2	+2.8	1.4
SPEEX-MODE5	+1.0	0.6
SPEEX-MODE8	+0.6	0.7
CELP-GAMMA10.70	**−0.8**	0.8
CELP-GAMMA10.85	**−2.0**	0.5
NOISE-GAMMA0.60_0.9	+1.9	0.4
NOISE-GAMMA0.85_1.0	+1.6	0.3
ADPCM-MIX0.2	+0.4	0.6
ADPCM-MIX0.5	+0.0	0.2

Table 5.5: Increase in maximum stable gain in [dB]. Overall results of simulated measurements for the living room condition.

5.2.2 Validation of the simulation framework

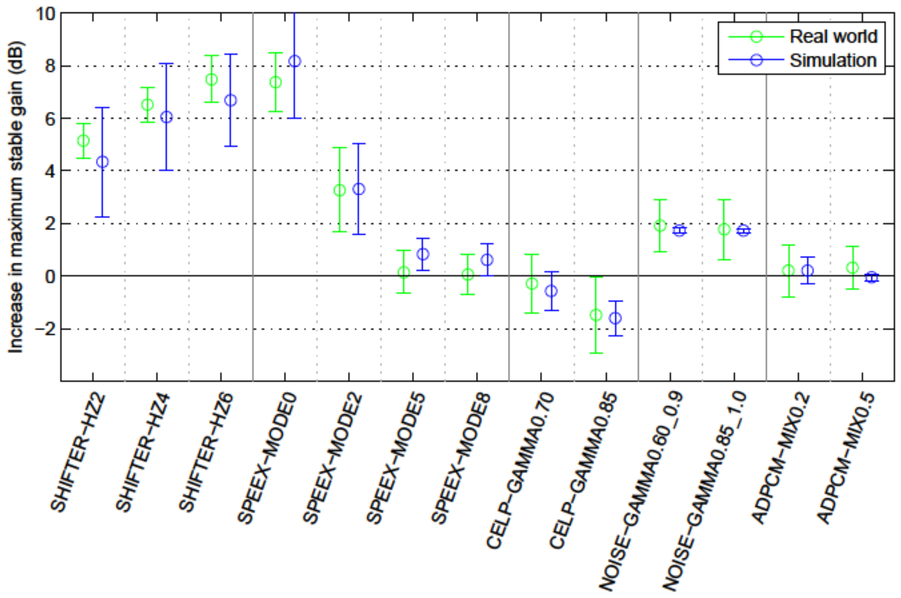

Figure 5.6: Increase in maximum stable gain in [dB]. Mean and standard deviation plots for live test results separated by measurement condition.

Measurement condition	Real-world		Simulation	
Algorithm under test	Mean	StDev	Mean	StDev
SHIFTER-HZ2	+5.2	0.7	+4.3	2.1
SHIFTER-HZ4	+6.5	0.7	+6.0	2.0
SHIFTER-HZ6	+7.5	0.9	+6.7	1.8
SPEEX-MODE0	+7.4	1.1	+8.2	2.2
SPEEX-MODE2	+3.3	1.6	+3.3	1.7
SPEEX-MODE5	+0.1	0.8	+0.8	0.6
SPEEX-MODE8	+0.1	0.8	+0.6	0.6
CELP-GAMMA$_1$0.70	**−0.3**	1.1	**−0.6**	0.7
CELP-GAMMA$_1$0.85	**−1.5**	1.5	**−1.6**	0.7
NOISE-GAMMA0.60_0.9	+1.9	1.0	+1.7	0.1
NOISE-GAMMA0.85_1.0	+1.8	1.1	+1.7	0.1
ADPCM-MIX0.2	+0.2	1.0	+0.2	0.5
ADPCM-MIX0.5	+0.3	0.8	**−0.1**	0.1

Table 5.6: Increase in maximum stable gain in [dB]. Live test results separated by measurement condition. While the variances between the two measuremen conditions differ, the mean values are comparable and hence show, that the simulation is a valid testing method, at least for pre-testing feedback suppression algorithms.

5.3 Discussion

In this section, the foregoing results will be analyzed, interpreted and evaluated with respect to the aim of this work of tweaking speech codecs to inhibit howling suppression.

No group differences in measurement data. Looking at the data in Tables 5.2, 5.3, 5.4, which allow us to compare the measurement data between different system setups, we can infer, that none of the setup categories was influencial to the measurements despite the low count of 12 data points per tested algorithm. Therefore, we will mainly rely on the overall results (Table 5.1) for the rest of this discussion.

No difference between live test results to the results from simulation. The validated simulation framework allowed for proper testing of an additional room (living room) within the simulation. The resulting mean values are comparable to the live test data and hence, do not lead to additional findings.

***Frequency shifter* reference has higher *IMSG* than reported in paper.** The frequency shifter algorithm that is used as a baseline for these measurements shows an *IMSG* from $+5.2dB$ up to 7.5 for frequency shifts between 2 Hz and 6 Hz. The range that is reported by [BH10] for these shift amounts is 2.5 dB to 3.5 dB. A possible explanation could be the difference in the stable feedback criterion that was applied in measurements.

As explained above, in the scope of this thesis, the only criterion determining if a system has a stable feedback depends on the energy in a late time frame. This criterion works very well in a system with no *algorithm under test* running, as the feedback mostly occurs at a single frequency that adds up or decreases its energy over time. This can be evaluated easily.

In the case of the *frequency shifter*, even if the mentioned criterion is not fullfilled, the actual listening experience is the one of a very strange signal, when the gain of the feedback loop is high but still stable. It might be, that in a conservative measurement setup one would have determined the feedback to be unstable in these cases, as it is the feedback loop that creates these sounds, that are very unpleasant. In the case of this work, we needed consistency with the simulation framework and so had to choose a criterion that is easy to evaluate, which might explain the differences in the measurements.

***Speex* shows a significant *IMSG* for low quality settings.** For the ultra-low quality mode 0 of *Speex*, the preliminary tests were successfully verified as at it has proven to provide an *IMSG* of 7.4 *dB*. In this mode, no pitch predictor is used. This will subtract most of the pitch information out of the signal and hence, no sinusoidal feedback can build up. This is exactly what can be heard when bringing the algorithm to a gain of unstable feedback: The perceived sound is not sinusoidal and instead sounds more like a pink noise signal that adds up in the feedback loop.

The reason for the *IMSG* of 3.5 *dB* for the modified *Speex* quality mode 2 should be very similar. In this mode, a pitch prediction is active, but it is weak and more error prone than the prediction algorithms running at high quality modes. This might be the reason, why the *IMSG* is lower than for quality mode 0, but higher than for the modes 5 and 8.

Last to mention, that the tested high quality modes 5 and 8 of *Speex* do not modify the stable feedback gain.

Negative effect on *IMSG* with original *CELP* implementation. In contrast to *Speex*, the original implementation of the *CELP* codec shows a negative effect on the *IMSG*. Compared to *Speex*, a main differnence in the implementation is the way that the pitch predicor works. When fed with an enduring sinusoidal signal, the original *CELP* produces a periodic clicking noise. This clicks might add additional energy to the system when a sinusoidal feedback is occurrent and by this accelerating the build up of signal energy in the feedback loop.

***Noise shaping mixer* increases the *maximum stable gain*, but this effect is not substantive.** As well as the low quality *Speex* modes, the *noise shaping mixer* shows an *increase in maximum stable gain*, here by 1.8 *dB* with the *Speex* noise shaping filter settings and 1.9 *dB* with the orignial *CELP* noise shaping filter settings.

The reason for this positive value lies in the method of removing the *speech gain* bias (See begining of this chapter on the calculation of the *speech gain* value.) from the measurement results. As the *speech gain* for this algorithm is 1.8 *dB*[2], it implies, that in actual measurements, the measured effect was only 0.1 dB and then, after removing the bias, we get to the high value of 1.9 *dB IMSG*. So actually, adding the noise does not actually affect the feedback loop, it only changes the *speech gain*

[2]1.8 dB is exactly the additional gain for an output signal, when uncorrelated noise with an SNR of 3 *dB* SNR is added to an input signal, which can be proven by adding up the powers of the to signals.

of the tested module, which is why the numbers with removed bias show a positive effect.

On the other hand, it **is** interesting to see, that for adding noise at 3 *dB SNR*, the *IMSG* is at 1.8 *dB*. This means, that even if the low qualtiy *Speex* modes had an *SNR* of 3 *dB*, we could not explain the feedback suppression effect only by the introduced noise, because the effect is significantly higher than 1.8dB. Hence, there definitely must be a working principle besides adding noise to the signal, that causes the feedback suppression within *Speex* .

No significant *IMSG* for the *tweaked ADPCM* module. The *tweaked ADPCM* module shows an *IMSG* of 0.2 and 0.3dB for the two tested configurations. With a standard deviation around 1.0 *dB* this is not significant. It is interesting to see, that the idea of flattening out the spectrum does not affect the feedback loop significantly. From a theoritical point of view, this should affect the *MSG* in a positive way.

An explanation why this did not work out in the measurements might be the low number count of 10 filter coefficients that was implemented. Such a low number might not be sufficient to subtract sine waves from the signal that are at the (usually high) frequency of sinusoidal feedback. For a sampling rate of 8 kHz, 10 samples correspond to a pitch periodicity of 800Hz, so sine waves above this frequency can not be canceled out by the analysis filter.

So, even if in this work the *ADPCM* did not help with feedback suppression, further research might lead to better results by using a longer prediction filter.

6. Conclusion

This work tried to utilize the basic concepts of speech coding to suppress acoustical feedback. The core result of this research is the following:

- The low quality settings of the *Speex* open-source speech codec increase the *maximum stable gain* by more than 3 dB. The goal of maximizing the possible effect of the codec on feedback suppression while also maintaining a reasonable audio quality was reached by configuring a new *Speex* quality mode that is a mixture of the native *Speex* quality modes 2 and 3.

Besides these main results there are a few other noticable findings regarding the research topic of feedback suppression:

- Each of the 13 algorithm under test configurations showed a standard deviation lower than 1.6 dB for the increase of maximum stable gain. Therefore, only a low number of measurements was needed to statistically verify that an algorithm has a positive effect on the increase in maximum stable gain.
- There was no significant difference in the measurements for different system configurations, meaning the variation of room size, microphone model or microphone positioning. We infer, that an algorithm that works well for feedback suppression in only a few conditions will also work well in comparable system setups.
- The frequency shifter algorithm might have an even higher affect on the increase in maximum stable gain than reported by [BH10]. Further tests should be carried out, in the best case with an enhanced stability detection criterion that includes the perceived annoyance of the signal in the loop.

Now that this thesis has – hopefully – answered several questions about the research topic of feedback suppression, I am happy to end this work with two questions that will might be able to drive further research in this field:

- Is there a way to find the exact working principle of the feedback suppression in the *Speex* low quality modes?
- And if yes, can these findings be utilized to tweak the codec even more than within this project?

■

Bibliography

[BH10] Berdahl, Edgar ; Harris, Dan: Frequency Shifting for Acoustic Howling Suppression. In: *Proceedings of the 13th International Conference on Digital Audio Effects, Graz, Austria*, 2010

[BM$^+$58] Box, George E. ; Muller, Mervin E. et al.: A note on the generation of random normal deviates. In: *The annals of mathematical statistics* 29 (1958), No. 2, P. 610–611

[Dur60] Durbin, James: The fitting of time-series models. In: *Revue de l'Institut International de Statistique* 28 (1960), P. 233–244

[ESW06] Estephan, Habib ; Sawyer, Scott ; Wanninger, Daniel: Real-Time Speech Pitch Shifting on an FPGA. In: *Departemen Teknik Elektro & Komputer. Universitas Villanova* (2006)

[FJ98] Frigo, Matteo ; Johnson, Steven G.: FFTW: An adaptive software architecture for the FFT. In: *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on* Bd. 3 IEEE, 1998, P. 1381–1384

[FOL11] Fober, Dominique ; Orlarey, Yann ; Letz, Stéphane: FAUST architectures design and OSC support. In: *14th Int. Conference on Digital Audio Effects (DAFx-11)*, 2011, P. 231–216

[GR10] Goldberg, Randy ; Riek, Lance: *A practical handbook of speech coders.* CRC press, 2010. – ISBN 9780849385254

[Hay96] Haykin, Simon S.: *Adaptive filter theory.* 3. Upper Saddle River, NJ, USA : Prentice-Hall, Inc., 1996. – ISBN 0133227600

[KK89] Kammeyer, Karl D. ; Kroschel, Kristian: *Digitale Signalverarbeitung*. Wiesbaden : Teubner, 1989. – ISBN 9783834806109. – 7. Auflage, erw. und korr. Auflage, Vieweg+Teubner

[Kon04] Kondoz, Ahmet M.: *Digital speech: coding for low bit rate communication systems*. 2. John Wiley & Sons, Ltd, 2004. – ISBN 9780470870075

[Lev47] Levinson, Norman: The Wiener RMS (root mean square) error criterion in filter design and prediction. In: *Selected papers of Norman Levinson* 2 (1947), P. 163

[Lev96] Levine, William S.: *The control handbook*. 2. Boca Raton FL : CRC press/IEEE press, 1996. – ISBN 9781420073669

[Mak77] Makhoul, John: Stable and efficient lattice methods for linear prediction. In: *Acoustics, Speech and Signal Processing, IEEE Transactions on* 25 (1977), No. 5, P. 423–428

[Nol12] Noll, Peter: *Quellencodierung WS 2011/12*. Vorlesungsskript der TU Berlin, unpublished, 2012

[Nyq32] Nyquist, Harry: Regeneration theory. In: *Bell System Technical Journal* 11 (1932), No. 1, P. 126–147

[Pet41] Peterson, Eugene: *Single side-band modulation*. http://www.google.com/patents/US2248250. Version: 1941. – US Patent 2,248,250

[San64] Sandberg, IW: On the-Boundedness of Solutions of Nonlinear Functional Equations. In: *Bell System Technical Journal* 43 (1964), No. 4, P. 1581–1599

[San65] Sandberg, Irwin W.: Some Results on the Theory of Physical Systems Governed Nonlinear Functional Equations. In: *Bell System Technical Journal* 44 (1965), No. 5, P. 871–898

[Sch64] Schroeder, MR: Improvement of Acoustic-Feedback Stability by Frequency Shifting. In: *The Journal of the Acoustical Society of America* 36 (1964), No. 9, P. 1718–1724

[Val06] Valin, Jean-Marc: Speex: a free codec for free speech. In: *Australian National Linux Conference, Dunedin, New Zealand* Citeseer, 2006

[WFM03] Wright, Matthew ; Freed, Adrian ; Momeni, Ali: Opensound control: State of the art 2003. In: *Proceedings of the 2003 conference on New interfaces for musical expression* National University of Singapore, 2003, P. 153–160

[Wil71] Willems, Jan C.: *The analysis of feedback systems.* The MIT Press, 1971. – ISBN 0262230461

[WM09] Waterschoot, Toon van ; Moonen, Marc: 50 years of acoustic feedback control: state of the art and future challenges. In: *Proc. IEEE* (2009), P. 08–13

[Zam64] Zames, George: On the stability of nonlinear, time-varying feedback systems. In: *Proc. 1964 Natl. El. Conf* 20 (1964), P. 725–730

[Zam66] Zames, George: On the input-output stability of time-varying nonlinear feedback systems part one: Conditions derived using concepts of loop gain, conicity, and positivity. In: *Automatic Control, IEEE Transactions on* 11 (1966), No. 2, P. 228–238