Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

Dennis Crantz

Aktive Regelung von Schalldruck und –schnelle mit einem digitalen Signalprozessor

Fakultät Technik und Informatik

Faculty of Engineering and Computer Science

Dennis Crantz

Aktive Regelung von Schalldruck und -schnelle mit einem digitalen Signalprozessor

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Mechatronik
an der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Erstprüfer: Prof. Dr.-Ing. habil. Thomas Kletschkowski
Zweitprüfer: Prof. Dr. rer. nat. Rasmus Rettig

Abgegeben am 07.10.2014

Zusammenfassung

Dennis Crantz

Thema der Bachelorthesis

Aktive Regelung von Schalldruck und -schnelle mit einem digitalen Signalprozessor

Stichworte

Active Noise Control, Active Noise Reduction, ANC, Schalldruck, Schallschnelle, digitaler Signalprozessor, DSP

Kurzzusammenfassung

Im Zuge dieser Arbeit wurde ein System entwickelt, um unterschiedliche Algorithmen zur aktiven Lärmkompensation auf einem digitalen Signalprozessor zu implementieren und deren Wirksamkeit gegenüberzustellen. Dabei wird neben der aktiven Beeinflussung des Schalldrucks auch die Beeinflussung des Schalldruckgradienten und eine damit theoretisch zu erzielende Vergrößerung des beruhigten Bereichs untersucht.

Dennis Crantz

Title of the paper

Active control of sound pressure and sound velocity using a digital signal processor

Keywords

active noise control, active noise reduction, ANC, sound pressure, sound velocity, digital signal processor, DSP

Abstract

In the course of this thesis a system to implement and compare the effectiveness of different algorithms for the task of active noise control was engineered. The goal is to cancel not only the sound pressure but also the pressure gradient to see if a larger zone of quiet can be achieved.

Inhaltsverzeichnis

Abbildungsverzeichnis **VI**

Abkürzungsverzeichnis **IX**

Symbolverzeichnis **X**

1 Einführung **1**
1.1 Einleitung . . . 1
1.2 Stand der Technik . . . 2
1.3 Grundlagen der technischen Akustik . . . 3
1.4 Grundlagen der Least-Mean-Squares-Algorithmen . . . 6
1.4.1 Least-Mean-Squares-Algorithmus . . . 6
1.4.2 Normalized Least-Mean-Squares-Algorithmus . . . 8
1.4.3 Filtered-Reference-Least-Mean-Squares-Algorithmus . . . 9
1.4.4 Feedback Least-Mean-Squares-Algorithmen . . . 10
1.4.5 Stabilität von Feedback LMS-Algorithmen . . . 12
1.5 Active Noise Control . . . 14
1.5.1 Grundlagen des Active Noise Control . . . 14
1.5.2 Einfluss der Sekundärstrecke . . . 17
1.6 Ziel der Arbeit . . . 18

2 Vorüberlegungen und Versuchsaufbau **19**
2.1 Verwendete Hardware . . . 19
2.2 Vergleichen der Algorithmen und Erfassen der Eingangssignale . . . 21

3 Implementierte adaptive Algorithmen **24**
3.1 Einkanalige adaptive Algorithmen . . . 24
3.1.1 Sekundärstreckenidentifikation bei einkanaligen Systemen . . . 24
3.1.2 Einkanalige adaptive Steuerung . . . 25
3.1.3 Einkanalige adaptive Regelung . . . 26
3.2 Mehrkanalige adaptive Algorithmen . . . 27
3.2.1 Sekundärstreckenidentifikation bei mehrkanaligen Systemen . . . 27
3.2.2 Mehrkanalige adaptive Steuerung . . . 28
3.2.3 Mehrkanalige adaptive Regelung . . . 29

4 Versuchsdurchführung **33**
4.1 Vergleich zwischen einkanaliger Steuerung und Regelung 33
4.2 Vergleich zwischen Druck-Druck und Druck-Druckgradient Steuerung 36
4.3 Vergleich zwischen Druck-Druck und Druck-Druckgradient Regelung . 39

5 Zusammenfassung **42**

Literaturverzeichnis **43**

Anhang **45**

A Gemessene Schallfelder ohne und mit ANC **46**
A.1 Schallfelder beim Vergleich zwischen einkanaliger Steuerung und Regelung 46
A.2 Schallfelder beim Vergleich zwischen Druck-Druck und Druck-Druckgradient Steuerung 48
A.3 Schallfelder beim Vergleich zwischen Druck-Druck und Druck-Druckgradient Regelung 50

B Zeitverläufe und Frequenzspektren ohne und mit ANC **52**
B.1 Zeitverläufe und Frequenzspektren beim Vergleich zwischen einkanaliger Steuerung und Regelung 52
B.2 Zeitverläufe und Frequenzspektren beim Vergleich zwischen Druck-Druck und Druck-Druckgradient Steuerung 54
B.3 Zeitverläufe und Frequenzspektren beim Vergleich zwischen Druck-Druck und Druck-Druckgradient Regelung 58

C C-Code implementierter Algorithmen **62**
C.1 C-Code einkanalige adaptive Steuerung 62
C.2 C-Code einkanalige adaptive Regelung 68
C.3 C-Code mehrkanalige adaptive Steuerung 75
C.4 C-Code mehrkanalige adaptive Regelung 82

Abbildungsverzeichnis

1.1 Blockdiagramm eines LMS-Filters 6
1.2 Sekundärstrecke in Reihe zum Filter 9
1.3 Blockdiagramm eines FxLMS-Filters 10
1.4 Blockdiagramm eines LMS-Filters mit Feedback 11
1.5 Blockdiagramm eines FxLMS-Filters mit Feedback 12
1.6 Blockdiagramm zur Verdeutlichung der Rückkopplung 13
1.7 Blockdiagramm Feedback FxLMS mit Filter zur Sicherstellung der Stabilität 14
1.8 Beispiel destruktiver Interferenz 15
1.9 Schematische Darstellung eines ANC-Systems mit Referenzsignal . . . 16
1.10 Schematische Darstellung eines ANC-Systems ohne Referenzsignal . . 17
1.11 Komponenten der Sekundärstrecke 17
2.1 Bild des Instrumentengehäuses mit der Elektronik zur Signalverstärkung 20
2.2 Bild einer verwendeten Sekundärquelle 21
2.3 Approximation der partiellen Ableitung nach einer Raumrichtung . . 22
3.1 Blockdiagramm der Sekundärstreckenidentifikation bei einkanaligen adaptiven Algorithmen 25
3.2 Blockdiagramm der einkanaligen adaptiven Steuerung 26
3.3 Blockdiagramm der einkanaligen adaptiven Regelung 27
3.4 Blockdiagramm der adaptiven Steuerung mit zwei Eingängen 29
3.5 Blockdiagramm der Sekundärstrecken bei zwei Ein- und Ausgängen . 30
3.6 Blockdiagramm der adaptiven Regelung mit zwei Ein- und Ausgängen 32
4.1 Versuchsanordnung beim Vergleich zwischen einkanaliger Steuerung und Regelung 34
4.2 Änderung des Schalldruckpegels bei der einkanaliger Steuerung 34
4.3 Änderung des Schalldruckpegels bei der einkanaliger Regelung 35
4.4 Vergleich der um 10 dB beruhigten Zonen zwischen einkanaliger Steuerung und Regelung 35
4.5 Versuchsanordnung beim Vergleich zwischen Druck-Druck und Druck-Druckgradient Steuerung 37
4.6 Änderung des Schalldruckpegels bei der Druck-Druck Steuerung . . . 37
4.7 Änderung des Schalldruckpegels bei der Druck-Druckgradient Steuerung 38
4.8 Vergleich der um 10 dB beruhigten Zonen bei der mehrkanaligen Steuerung 38

4.9 Versuchsanordnung beim Vergleich zwischen Druck-Druck und Druck-Druckgradient Regelung . . . 40
4.10 Änderung des Schalldruckpegels bei der Druck-Druck Regelung . . . 40
4.11 Änderung des Schalldruckpegels bei der Druck-Druckgradient Regelung 41
4.12 Vergleich der um 10 dB beruhigten Zonen bei der mehrkanaligen Regelung . . . 41
A.1 Gemessenes Schallfeld ohne ANC . . . 46
A.2 Gemessenes Schallfeld bei einkanaliger Steuerung . . . 47
A.3 Gemessenes Schallfeld bei einkanaliger Regelung . . . 47
A.4 Gemessenes Schallfeld ohne ANC . . . 48
A.5 Gemessenes Schallfeld bei Druck-Druck Steuerung . . . 49
A.6 Gemessenes Schallfeld bei Druck-Druckgradient Steuerung . . . 49
A.7 Gemessenes Schallfeld ohne ANC . . . 50
A.8 Gemessenes Schallfeld bei Druck-Druck Regelung . . . 51
A.9 Gemessenes Schallfeld bei Druck-Druckgradient Regelung . . . 51
B.1 Ausschnitt des Zeitverlaufs des Schalldrucks am Fehlermikrofon . . . 52
B.2 Frequenzspektrum des Schalldruckpegels am Fehlermikrofon . . . 53
B.3 Ausschnitt des Zeitverlaufs der Schalldrücke am Fehlermikrofon bei der Druck-Druck Steuerung . . . 54
B.4 Frequenzspektrum des Schalldruckpegels am ersten Fehlermikrofon bei der Druck-Druck Steuerung . . . 55
B.5 Frequenzspektrum des Schalldruckpegels am zweiten Fehlermikrofon bei der Druck-Druck Steuerung . . . 55
B.6 Ausschnitt des Zeitverlaufs der addierten Schalldrücke bei der Druck-Druckgradient Steuerung . . . 56
B.7 Ausschnitt des Zeitverlaufs der subtrahierten Schalldrücke bei der Druck-Druckgradient Steuerung . . . 56
B.8 Frequenzspektrum des Schalldruckpegels der addierten Schalldrücke bei der Druck-Druckgradient Steuerung . . . 57
B.9 Frequenzspektrum des Schalldruckpegels der subtrahierten Schalldrücke bei der Druck-Druckgradient Steuerung . . . 57
B.10 Ausschnitt des Zeitverlaufs der Schalldrücke am Fehlermikrofon bei der Druck-Druck Regelung . . . 58
B.11 Frequenzspektrum des Schalldruckpegels am ersten Fehlermikrofon bei der Druck-Druck Regelung . . . 59
B.12 Frequenzspektrum des Schalldruckpegels am zweiten Fehlermikrofon bei der Druck-Druck Regelung . . . 59
B.13 Ausschnitt des Zeitverlaufs der addierten Schalldrücke bei der Druck-Druckgradient Regelung . . . 60
B.14 Ausschnitt des Zeitverlaufs der subtrahierten Schalldrücke bei der Druck-Druckgradient Regelung . . . 60
B.15 Frequenzspektrum des Schalldruckpegels der addierten Schalldrücke bei der Druck-Druckgradient Regelung . . . 61

B.16 Frequenzspektrum des Schalldruckpegels der subtrahierten Schalldrücke bei der Druck-Druckgradient Regelung 61

Abkürzungsverzeichnis

ANC	Active Noise Control, Active Noise Cancellation
ANR	Active Noise Reduction
DFT	discrete Fourier transform
DSP	digitaler Signalprozessor
DTFT	discrete-time Fourier transform
FIR	finite impulse response
FPGA	Field Programmable Gate Array
FxLMS	Filtered-Reference-Least-Mean-Squares
IIR	infinite impulse response
LMS	Least-Mean-Squares
MSE	mean squared error
NLMS	Normalized Least-Mean-Squares-Algoirithmus

Symbolverzeichnis

c	Schallgeschwindigkeit $= 343\,\frac{\mathrm{m}}{\mathrm{s}}$	$\frac{\mathrm{m}}{\mathrm{s}}$
f	Frequenz	Hz
L_p	Schalldruckpegel	dB
λ	Wellenlänge	m
$\boldsymbol{\nabla}$	Nabla-Operator $\boldsymbol{\nabla} = \left[\frac{\partial}{\partial x} \quad \frac{\partial}{\partial y} \quad \frac{\partial}{\partial z}\right]^\mathsf{T}$	
p	Schalldruck	$\frac{\mathrm{N}}{\mathrm{m}^2}$
$\widetilde{p}$	Effektivwert des Schalldrucks	$\frac{\mathrm{N}}{\mathrm{m}^2}$
ρ_0	Schalldichte $\approx 1{,}184\,\frac{\mathrm{kg}}{\mathrm{m}^3}$ bei $25\,^\circ\mathrm{C}$	$\frac{\mathrm{kg}}{\mathrm{m}^3}$
t	Zeit	s
$\boldsymbol{v}^\mathsf{T}$	Transponierte des Vektors $\boldsymbol{v}$	
$\boldsymbol{v}$	Schallschnelle $\boldsymbol{v} = [v_x \quad v_y \quad v_z]^\mathsf{T}$	$\frac{\mathrm{m}}{\mathrm{s}}$

$\boldsymbol{x}$	Ortsvektor $\boldsymbol{x} = [x \quad y \quad z]^\mathsf{T}$	m
$\boldsymbol{\xi}$	Schallauslenkung $\boldsymbol{\xi} = [\xi_x \quad \xi_y \quad \xi_z]^\mathsf{T}$	m

1 Einführung

1.1 Einleitung

Active Noise Control (ANC), auch Active Noise Cancellation, Active Noise Reduction (ANR) oder Aktive Lärmminderung, ist ein Verfahren der aktiven Minderung von Schall durch das Aussenden von Gegenschall und dient der Reduktion von Lärm nach dem Prinzip der destruktiven Interferenz.

Lärm tritt in nahezu allen Bereichen des Alltags auf. Er gilt als Umweltverschmutzung und wird als störend empfunden. Er kann sich belastend oder sogar gesundheitsschädlich auf den Menschen auswirken. Ein angestrebtes Ziel bei der Entwicklung von Produkten ist es deshalb oft, Lärm zu minimieren, sei es in der Produktion von Güter aus Gründen der Arbeitssicherheit oder zum Beispiel im Personenverkehr zur Steigerung des Komforts. Typische Störquellen im Zusammenhang mit ANC sind zum Beispiel Motoren, Lüftungen, Gebläse oder Kompressoren. Vibrationen führen oft zum ungewollten Abstrahlen von Schall. Dieses Problem wird durch den vermehrten Einsatz des Leichtbaus noch verstärkt. ANC kann oder wird bereits in vergleichsweise kleinen, leichter zu beeinflussenden Umgebungen, wie zum Beispiel in Kopfhörern beziehungsweise Gehörschützern, als auch in größeren Umgebungen, wie zum Beispiel in Flugzeugen, eingesetzt.

Traditionell wird die Ausbreitung von störendem Schall durch passive Maßnahmen reduziert. So werden zum Beispiel absorbierende Materialien eingesetzt, um zu verhindern, dass Schallleistung an einer Wand reflektiert wird, sondern im absorbierenden Material in Wärme umgewandelt wird.

Eine ähnliche Problemstellung stellt auch die aktive Reduktion von Körperschwingungen dar, das Active Vibration Control. Im Unterschied zum ANC kommen hier aber anstelle von Lautsprechern elektromagnetische oder hydraulische Aktoren zum Einsatz, welche Kräfte in die Struktur einleiten. Mit diesen so genannten Shakern können dann Kräfte erzeugt werden, welche die Schwingungen reduzieren. Dies kann auch im Bereich der aktiven Lärmminderung eine Option sein, wenn direkt der Körperschall reduziert werden kann und somit der abgestrahlte, sich in der Luft ausbreitende Schall vermindert wird.

Handelt es sich beim Lärm um statische Störungen, so ist es möglich ein Modell des Systems zu bestimmen und mit Hilfe dessen einen optimalen Regler zu errechnen. Dies ist jedoch aufgrund der Komplexität nicht immer möglich. Weiter sind die Störungen oftmals nicht stationär. Wenn man zum Beispiel den Verbrennungsmotor eines Autos als Störquelle betrachtet, so ändert sich über die Zeit sowohl die Amplitude als auch die Frequenz des verursachten Schalldrucks. Um auf solche Änderungen reagieren zu können, sind adaptive Verfahren notwendig. Die wohl verbreitetste Form solcher Systeme stellen adaptive Filter, welche auf dem Least-Mean-Square-Problem basieren, dar.

ANC Systeme zeichnet aus, dass sie mitunter einen Vorteil gegenüber passiven Systemen in Hinblick auf Größe, Gewicht und Kosten aufweisen, vor allem im Bereich niedriger Frequenzen, in welchem sie am effektivsten eingesetzt werden können. Es gilt zu prüfen, ob eventuell sogar eine Integration in bereits vorhandene Hardware (zum Beispiel Infotainmentsysteme im Automobil) möglich ist.

Ein Problem klassischer ANC Systeme ist die Eigenschaft, nur sehr lokal einen markanten Erfolg in der Reduktion des Lärms vorweisen zu können. So kann bereits mit kleinem Abstand zum Punkt, an dem der Schall geregelt wird, der Effekt der Schallreduktion nicht mehr gegeben sein. Aufgrund dessen ist ein Ziel der Forschung, den beeinflussten Bereich weiter zu vergrößern, um den Einsatz der Technik attraktiver zu machen.

In der Literatur und im Sprachgebrauch hat sich im Zusammenhang mit ANC die Verwendung englischer Begriffe (zum Beispiel Active Noise Control, Least-Mean-Square) durchgesetzt. Aus diesem Grund wird in dieser Arbeit bei der Verwendung solcher Begriffe zumeist auf diese zurückgegriffen, anstatt die deutsche Übersetzung (zum Beispiel aktive Lärmminderung, kleinste mittlere quadratische Abweichung) zu nutzen.

1.2 Stand der Technik

Wie bereits erwähnt, kommt ANC bereits in einigen Bereichen zum Einsatz. Einer, in dem die Technologie bereits seit einiger Zeit eingesetzt wird, ist der von Kopfhörern. Anfangs wurden ANC-Kopfhörer für den Einsatz in Cockpits von Hubschraubern und Flugzeugen entwickelt. Diese sollten sicherstellen, dass trotz lauter Umgebungsgeräusche die Kommunikation mit anderen möglich ist. Seit einiger Zeit kommen immer mehr Kopfhörer mit dieser Technologie für den privaten Gebrauch auf den Markt. Die Herausforderung liegt hier darin, das Ganze zu vergleichsweise geringen Preisen bereit zu stellen, damit die Kopfhörer nicht übermäßig teurer werden und auf dem Markt bestehen können. Inzwischen haben alle großen Hersteller

solche Kopfhörer im Angebot. Derartige Kopfhörer sind in erster Linie für längere Reisen gedacht, um die wahrgenommenen Umgebungsgeräusche, zum Beispiel in Bahn oder Flugzeug, zu vermindern und so den Reisekomfort zu steigern.

Weitere Einsatzorte von ANC sind mittelgroße Propellermaschinen. Hier ist bereits eine größere Anzahl solcher Systeme im Einsatz, und Klimaanlagen in großen Bürokomplexen. Besonders Klimaanlagen sind ein Anwendungsgebiet, in dem monotone Störgeräusche vorliegen, welche sich gut für aktive Maßnahmen eignen.

Auch wenn die Idee hinter Active Noise Control vergleichsweise einfach ist (siehe Kapitel 1.5), so ist die praktische Umsetzung eines solchen Systems doch mit einigen Schwierigkeiten verbunden. Aus diesem Grund ist der Einsatz von ANC in verschiedensten Bereichen auch immer noch Bestandteil der Forschung und es existieren lediglich Labormodelle oder gar nur rein theoretische Ansätze. Da es quasi noch keine flexibel einsetzbaren Systeme gibt, muss für jeden Anwendungsfall eine Neuentwicklung durchgeführt werden, was sich im Preis der Integration einer ANC Lösung bemerkbar macht. Auf Grund dessen und dadurch, dass es mitunter schwierig ist, den Erfolg der Integration einer aktiven Lösung zur Lärmminderung voraus zu sagen, ist die Verwendung von ANC-Systemen immer noch recht überschaubar.

Die Algorithmen, welche zur Umsetzung nötig sind, können bei komplexeren Problemstellungen sehr schnell sehr rechenintensiv werden. Im Moment werden derartige Algorithmen vorzugsweise auf einem digitalen Signalprozessor (DSP) implementiert. Diese sind so gestaltet, dass sie sich besonders gut für Aufgaben der digitalen Signalverarbeitung eignen. Aber auch hier gilt es, schon im Voraus abzuschätzen, ob der DSP den Algorithmus in Echtzeit abarbeiten kann.

1.3 Grundlagen der technischen Akustik

In diesem Kapitel soll einmal auf alle für die Arbeit relevanten akustischen Größen eingegangen werden, welche zum Lösen der Problemstellung nötig sind, so wie deren Zusammenhänge gezeigt werden.

Bei Schall handelt es sich um Schwingungen von Gasteilchen. Diese führen zu einem Schalldruck p (auch Schallwechseldruck), welcher dem statischen Druck p_{stat} überlagert ist, so dass für den Gesamtdruck p_{ges} gilt:

$$p_{\text{ges}} = p_{\text{stat}} + p \tag{1.1}$$

Der Schalldruck stellt für die Arbeit eine sehr wichtige physikalische Größe dar, da das Trommelfell des menschlichen Ohres, aber auch die Membran von Mikrofonen, der Druckschwankung folgt, indem es auf die Druckdifferenz zwischen Vorder- und

Rückseite reagiert, und somit der Schalldruck die wahrgenommene beziehungsweise gemessene Größe darstellt. Beim Schalldruck handelt es sich um eine skalare Größe, welche vom Ort $\boldsymbol{x}$ und von der Zeit t abhängig ist.

$$p = p\left(\boldsymbol{x}, t\right) \qquad \text{mit} \qquad \boldsymbol{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Weiter ist im Zusammenhang mit dem Schalldruck p der Effektivwert des Schalldrucks $\widetilde{p}$ von Interesse, insbesondere bei der Betrachtung sinusförmiger Größen. Mit ihm wird der Schalldruckpegel L_p gebildet:

$$L_p = 10 \log_{10} \left(\frac{\widetilde{p}^{\,2}}{p_0^2} \right) \text{ dB} = 20 \log_{10} \left(\frac{\widetilde{p}}{p_0} \right) \text{ dB} \tag{1.2}$$

Der Schalldruckpegel gibt das logarithmierte Verhältnis des Effektivwertes zum Bezugswert $p_0 = 2 \cdot 10^{-5}\,\text{Pa}$ an. Der Bezugswert p_0 gibt ungefähr die menschliche Hörschwelle eines Tones von 1 kHz an.

Betrachtet man den Schalldruck an verschiedenen Orten gleichzeitig, spricht man von einem Schallfeld. Dieses bezeichnet das Gebiet, in dem sich Schallwellen ausbreiten. Um ein Schallfeld vollständig beschreiben zu können, muss man an allen Orten des Schallfeldes zu jedem Zeitpunkt neben dem Schalldruck auch die Schallschnelle kennen. Die Schallschnelle $\boldsymbol{v}$ ergibt sich aus der zeitlichen Ableitung der Schallauslenkung $\boldsymbol{\xi}$.

$$\boldsymbol{v} = \frac{\partial \boldsymbol{\xi}}{\partial t} = \dot{\boldsymbol{\xi}} \tag{1.3}$$

Bei beiden Größen handelt es sich um vektorielle Größen, die ebenfalls wieder von Ort und Zeit abhängen.

$$\boldsymbol{v} = \boldsymbol{v}\left(\boldsymbol{x}, t\right) \qquad \boldsymbol{\xi} = \boldsymbol{\xi}\left(\boldsymbol{x}, t\right)$$

Die Schallauslenkung beschreibt die Auslenkung eines Teilchens im Übertragungsmedium aus seiner Ruhelage und zeigt in Richtung der Auslenkung. Die Schallschnelle beschreibt die zeitliche Änderung der Auslenkung und somit die Geschwindigkeit des Teilchens und zeigt in die Richtung, in welche sich das Teilchen bewegt.

$$\boldsymbol{v} = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \qquad \boldsymbol{\xi} = \begin{bmatrix} \xi_x \\ \xi_y \\ \xi_z \end{bmatrix}$$

Nicht verwechselt werden darf die Schallschnelle mit der Schallgeschwindigkeit c, welche die Ausbreitungsgeschwindigkeit von Schall in einem Medium angibt und für Luft näherungsweise mit $c = 343\,\frac{\text{m}}{\text{s}}$ gegeben ist. Mit Hilfe der Schallgeschwindigkeit kann über die Frequenz f einer Schallwelle die Wellenlänge λ bestimmt werden.

$$\lambda = \frac{c}{f} \tag{1.4}$$

Die Wellenlänge beschreibt den kleinsten Abstand zweier Punkte einer Welle gleicher Auslenkung und Bewegungsrichtung.

Einige der hier angesprochenen Größen hängen über die Dichte des Übertragungsmediums ρ_0 zusammen, welche als annähernd konstant angenommen wird. Diese ist Temperaturabhängig und beträgt in unserer Atmosphäre bei 25 °C (298,15 K) ungefähr 1,184 $\frac{\mathrm{kg}}{\mathrm{m}^3}$.

Möchte man wissen, wie sich der Druck räumlich ändert, ist der Schalldruckgradient $\boldsymbol{\nabla} p$ von Interesse. Dieser ist eine vektorielle Größe und besteht aus den räumlichen Ableitungen des Schalldrucks. Der Gradient wird mit Hilfe des Nabla-Operators $\boldsymbol{\nabla}$ gebildet, welcher aus den partiellen Richtungsableitungen besteht.

$$\boldsymbol{\nabla} = \begin{bmatrix} \dfrac{\partial}{\partial x} \\ \dfrac{\partial}{\partial y} \\ \dfrac{\partial}{\partial z} \end{bmatrix}$$

$$\boldsymbol{\nabla} p = \begin{bmatrix} \dfrac{\partial p}{\partial x} \\ \dfrac{\partial p}{\partial y} \\ \dfrac{\partial p}{\partial z} \end{bmatrix} = \boldsymbol{\nabla} p\left(\boldsymbol{x}, t\right)$$

Der Schalldruckgradient zeigt in die Richtung der stärksten Änderung des Schalldrucks und sein Betrag ist umso größer, je größer die Änderung des Drucks ist. Er ist ebenfalls abhängig von Ort und Zeit. Würde im Schallfeld ein konstanter Schalldruck herrschen, wäre der Gradient gleich Null.

Der Schalldruckgradient und die Schallschnelle hängen über folgende Beziehung zusammen:

$$\boldsymbol{\nabla} p = -\rho_0 \frac{\partial \boldsymbol{v}}{\partial t} \tag{1.5}$$

Der Schalldruckgradient ist demzufolge proportional zur zeitlichen Ableitung der Schallschnelle.

$$\boldsymbol{\nabla} p \sim \frac{\partial \boldsymbol{v}}{\partial t} \tag{1.6}$$

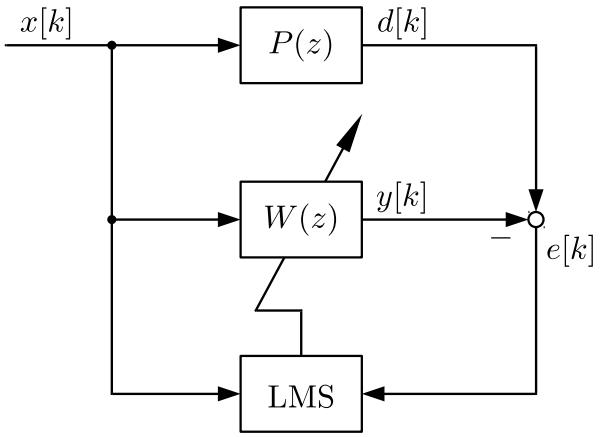

Abbildung 1.1: Blockdiagramm eines LMS-Filters

1.4 Grundlagen der Least-Mean-Squares-Algorithmen

Nach den akustischen Grundlagen wird in diesem Kapitel auf die Grundlagen der Least-Mean-Squares-Algorithmen (LMS-Algorithmen) eingegangen und für die Arbeit relevante Algorithmen vorgestellt.

1.4.1 Least-Mean-Squares-Algorithmus

LMS-Algorithmen bilden die Grundlage für adaptive Filter mit endlicher Impulsantwort (engl. finite impulse response, FIR). Deren Koeffizienten sind nicht konstant, sondern werden fortlaufend angepasst, sprich adaptiert. Adaptive Filter sind dementsprechend nicht zeitinvariant. Anwendung finden LMS-Algorithmen in einer Vielzahl von Bereichen, neben ANC zum Beispiel in der Telekommunikations- und Regelungstechnik.

LMS-Algorithmen beruhen darauf, den mittleren quadratischen Fehler (engl. mean squared error, MSE) zu minimieren. Das Blockdiagramm eines adaptiven Filters ist mit Abbildung 1.1 gegeben. Eine Primärstrecke $P(z)$ erhält als Eingangssignal das Referenzsignal $x[k]$ und gibt das Signal $d[k]$ aus. Das adaptive Filter $W(z)$ erhält ebenfalls das Referenzsignal als Eingang und gibt das Signal $y[k]$ aus. Die Koeffizienten des Filters $W(z)$ werden durch den LMS-Algorithmus eingestellt, welcher die Eingangssignale $x[k]$ und $e[k]$ erhält. Ziel ist es, den Fehler $e[k]$ zu minimieren.

Im Folgenden sollen die Signale und Filter aus Abbildung 1.1 im Frequenzbereich betrachtet werden. Hierbei ist $\circ\!\!-\!\!\bullet$ der Transformationsoperator und steht in diesem Fall für die zeitdiskrete Fourier-Transformation (engl. discrete-time Fourier trans-

form, DTFT), nicht zu verwechseln mit der diskreten Fourier-Transformation (engl. discrete Fourier transform, DFT). Für das optimale Filter $W_{\text{opt}}(f)$ soll gelten, dass der Fehler $E(f)$ gleich Null sei:

$$x[k] \circ\!\!-\!\!\bullet X(f) \quad e[k] \circ\!\!-\!\!\bullet E(f) \quad y[k] \circ\!\!-\!\!\bullet Y(f) \quad d[k] \circ\!\!-\!\!\bullet D(f)$$

$$D(f) = P(f)\,X(f) \tag{1.7}$$

$$Y(f) = W_{\text{opt}}(f)\,X(f) \tag{1.8}$$

$$E(f) = D(f) - Y(f) \tag{1.9}$$

$$E(f) \stackrel{!}{=} 0 \tag{1.10}$$

$$0 = P(f)\,X(f) - W_{\text{opt}}(f)\,X(f) \tag{1.11}$$

$$\Rightarrow W_{\text{opt}}(f) = P(f) \tag{1.12}$$

Aus der Forderung in Gleichung (1.10) folgt Gleichung (1.11). Durch Umformen ergibt sich Gleichung (1.12), aus der erkennbar ist, dass für den optimalen Fall die Frequenzgänge der Primärstrecke und des Filters gleich sein müssen. Handelt es sich bei $P(z)$ um ein FIR-Filter mit gleicher Anzahl an Koeffizienten wie $W(z)$, so ist die Lösung leicht ersichtlich. Das optimale Filter muss die gleichen Koeffizienten wie die Primärstrecke besitzen. Handelt es ich bei $P(z)$ um zum Beispiel ein Filter mit unendlicher Impulsantwort (engl. infinite impulse response, IIR), so kann die Impulsantwort und damit auch der Frequenzgang von $P(z)$ durch $W(z)$ nur angenähert werden.

Es muss folglich eine Funktion gefunden werden, anhand derer der Fehler minimiert werden kann. Hierfür wird der mittlere quadratische Fehler heran gezogen. Das nach Norbert Wiener benannte Wiener Filter ist die Lösung des Problems des kleinsten mittleren quadratischen Fehlers. Der MSE stellt in Abhängigkeit von den Filterkoeffizienten eine quadratische Funktion dar und hat damit genau einen Extremwert. Für das Wiener Filter gilt, dass der MSE gerade sein Minimum annimmt. Für die Bestimmung des Wiener Filters ist die Kenntnis über die Statistik der Signale $x[k]$ und $d[k]$ von Nöten. Dies ist, insbesondere wenn es sich um einen nicht statischen Prozess handelt, in der Regel nicht möglich. Mit dem Verfahren des steilsten Abstiegs kann für die quadratische Funktion des MSE mit Hilfe des Gradienten rekursiv das Minimum gefunden werden, so dass die Lösung zur Wiener Filter Lösung konvergiert. Um nicht mehr auf die Statistik der Signale angewiesen zu sein, wird beim LMS-Algorithmus der Gradient lediglich geschätzt, in dem der MSE durch den aktuellen quadratischen Fehler $e^2[k]$ angenähert wird. So wird das Minimum und damit die Wiener Filter Lösung in der Regel nicht mehr genau, aber hinreichend genau, angenähert. Für eine ausführliche Herleitung des LMS-Algorithmus sei an dieser Stelle auf die Literatur (zum Beispiel [8]) verwiesen. Einen großen Vorteil des LMS-Algorithmus stellt die geringe Komplexität dar, welche ihn attraktiv für den Einsatz in verschiedensten Gebieten macht.

Für das Blockdiagramm in Abbildung 1.1 gelten die folgenden Gleichungen, welche in jedem Zyklus neu berechnet und die Filterkoeffizienten $\boldsymbol{w}[k]$ dadurch rekursiv angepasst werden:

$$y[k] = \boldsymbol{w}^{\mathsf{T}}[k]\,\boldsymbol{x}[k] \tag{1.13}$$

$$e[k] = d[k] - y[k] \tag{1.14}$$

$$\boldsymbol{w}[k{+}1] = \boldsymbol{w}[k] + \mu\, e[k]\,\boldsymbol{x}[k] \tag{1.15}$$

Mit Gleichung (1.13) wird der Filterausgang $y[k]$ berechnet, wobei $\boldsymbol{w}[k]$ einen Vektor darstellt, welcher die N_W Filterkoeffizienten des Filters $W(z)$ zum Zeitpunkt k beinhaltet.

$$\boldsymbol{w}[k] = \begin{bmatrix} w_0[k] & w_1[k] & \cdots & w_{N_W-1}[k] \end{bmatrix}^{\mathsf{T}}$$

Der Fehler $e[k]$ wird, wie in Gleichung (1.14) gezeigt, aus der Differenz der aktuellen Signale $d[k]$ und $y[k]$ gebildet. Gleichung (1.15) zeigt die LMS Updategleichung für die Filterkoeffizienten. Diese berechnen sich aus den alten Filterkoeffizienten und der Multiplikation der Schrittweite μ, des aktuellen Fehlersignals $e[k]$ und dem Vektor der letzten N_W Referenzsignalsampels $\boldsymbol{x}[k]$ mit

$$\boldsymbol{x}[k] = \begin{bmatrix} x[k] & x[k{-}1] & \cdots & x[k{-}N_W{+}1] \end{bmatrix}^{\mathsf{T}}.$$

Bei der Schrittweite ist darauf zu achten, dass diese, wird sie zu groß gewählt, dazu führen kann, dass der Algorithmus instabil wird. Problematisch ist die Tatsache, dass die maximal mögliche Schrittweite vom Referenzsignal $x[k]$ abhängt.

1.4.2 Normalized Least-Mean-Squares-Algorithmus

Um die Schrittweite unabhängig vom Referenzsignal $x[k]$ zu machen, kann eine Normierung durchgeführt werden. Bei einem solchen Normalized Least-Mean-Squares-Algorithmus (NLMS) wird die Schrittweite berechnet, indem ein Faktor β durch das Quadrat der 2-Norm (auch euklidische Norm) des Referenzsignalvektors $\boldsymbol{x}[k]$ geteilt wird. Das Quadrat der 2-Norm ist gleich dem Skalarprodukt des Vektors mit sich selbst.

$$\| \boldsymbol{x}[k] \|_2^2 = \boldsymbol{x}^{\mathsf{T}}[k]\,\boldsymbol{x}[k] \tag{1.16}$$

$$\mu[k] = \frac{\beta}{\gamma + \boldsymbol{x}^{\mathsf{T}}[k]\,\boldsymbol{x}[k]} \qquad 0 < \beta < 2 \tag{1.17}$$

Die kleine, positive Konstante γ sorgt dafür, dass die Schrittweite nicht zu groß wird. Weiter wird so eine Division durch Null verhindert. Negativ macht sich bei dieser Variante bemerkbar, dass eine Division durchgeführt werden muss. Divisionen sind rechentechnisch gesehen sehr aufwendig. Auch hier soll für eine Herleitung des NLMS auf die entsprechende Literatur verwiesen werden (zum Beispiel [8]).

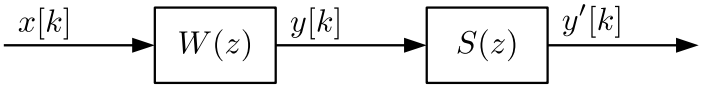

Abbildung 1.2: Sekundärstrecke in Reihe zum Filter

1.4.3 Filtered-Reference-Least-Mean-Squares-Algorithmus

In vielen Fällen wirkt der Filterausgang $y[k]$ nicht direkt auf den Summationspunkt des Fehlersignals, sondern wird selbst noch mit einer so genannten Sekundärstrecke $S(z)$ gefiltert (siehe Abbildung 1.2), welche dann das mit ihr gefilterte Signal $y'[k]$ als Ausgang besitzt. Dieser Fall tritt zum Beispiel beim ANC auf (vergleiche Kapitel 1.5). Um Konvergenz des LMS sicher zu stellen, muss dieser modifiziert werden. Eine Möglichkeit besteht, wenn man den Fall im Frequenzbereich betrachtet, darin, das invertierte Modell der Sekundärstrecke $\widehat{S}^{-1}(f)$ in Reihe zu schalten. Unter der Annahme eines optimalen Modells folgt:

$$\widehat{S}_{\text{opt}}(f) \stackrel{!}{=} S(f) \tag{1.18}$$

$$W(f)\,S(f)\,\frac{1}{\widehat{S}_{\text{opt}}(f)}\,X(f) = W(f)\,X(f) = Y(f) \tag{1.19}$$

Aus Gleichung (1.19) geht hervor, dass, unter Voraussetzung eines gut angenäherten Modells $\widehat{S}(z)$, der Einfluss der Sekundärstrecke ausgeglichen werden kann und wieder das Signal $y[k]$ am Summationspunkt anliegt. Problematisch ist die Tatsache, dass es nicht immer möglich ist, ein invertiertes Modell zu bilden. Aus diesem Grund wird oft eine andere Herangehensweise gewählt. Betrachtet man wieder die Reihenschaltung aus Filter und Sekundärstrecke im Frequenzbereich, so zeigt sich, dass sich der Fehler nun wie folgt zusammensetzt:

$$y'[k] \circ\!\!-\!\!\bullet\, Y'(f)$$

$$E(f) = D(f) - Y'(f) \tag{1.20}$$

$$= D(f) - S(f)\,Y(f) \tag{1.21}$$

$$= D(f) - S(f)\,W(f)\,X(f) \tag{1.22}$$

$$= D(f) - W(f)\,\Big(S(f)\,X(f)\Big) \tag{1.23}$$

In einer Reihenschaltung können Übertragungsfunktionen vertauscht werden, ohne das Verhalten der Reihenschaltung zu ändern. Das Fehlersignal, welches es zu minimieren gilt, besteht aus der Differenz von $d[k]$ und $y'[k]$. Das Signal $x[k]$, gefiltert mit $S(z)$, entspricht somit, aus Sicht des Fehlersignals des LMS $e[k]$, dem neuen Referenzsignal (siehe Gleichung (1.23)). Der LMS wird so angepasst, dass er, anstelle des ursprünglichen Referenzsignals $x[k]$, das mit einem Sekundärstreckenmodell $\widehat{S}(z)$

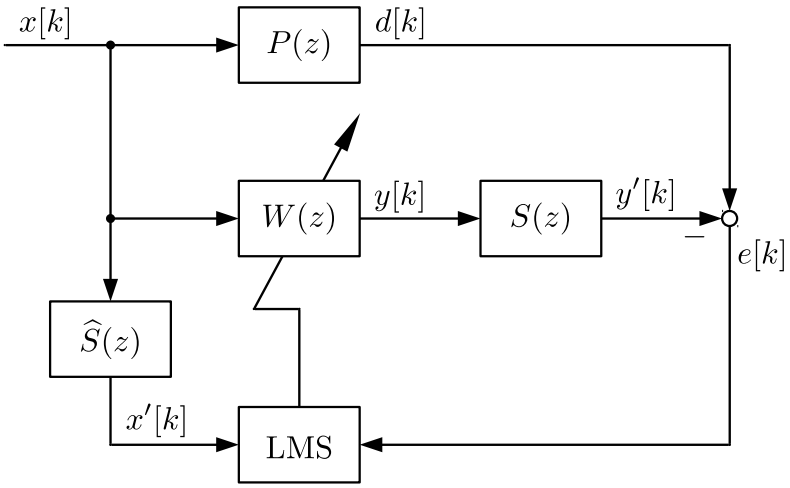

Abbildung 1.3: Blockdiagramm eines FxLMS-Filters

gefilterte Signal $x'[k]$ erhält. Diese Art von Algorithmus wird als Filtered-Reference-Least-Mean-Squares-Algorithmus, abgekürzt FxLMS-Algorithmus, bezeichnet. Die Updategleichung für den Koeffizientenvektor $\boldsymbol{w}[k]$ lautet in diesem Fall wie folgt:

$$\boldsymbol{w}[k{+}1] = \boldsymbol{w}[k] + \mu\, e[k]\, \boldsymbol{x}'[k] \tag{1.24}$$

Das gefilterte Referenzsignal $x'[k]$ wird dabei mit

$$x'[k] = \widehat{\boldsymbol{s}}^{\mathsf{T}}\, \boldsymbol{x}[k] \tag{1.25}$$

gebildet. Der Vektor $\widehat{\boldsymbol{s}}$ beinhaltet die N_S Koeffizienten des Sekundärstreckenmodells $\widehat{S}(z)$:

$$\widehat{\boldsymbol{s}} = \begin{bmatrix} \widehat{s}_0 & \widehat{s}_1 & \cdots & \widehat{s}_{N_S-1} \end{bmatrix}^{\mathsf{T}}$$

Dieses wird in der Regel im Vorfeld eingemessen und ist während des eigentlichen Betriebs zeitinvariant. Das Blockdiagramm eines FxLMS ist in Abbildung 1.3 gezeigt. Das Modell der Sekundärstrecke kann unter Verwendung eines einfachen LMS erstellt werden.

1.4.4 Feedback Least-Mean-Squares-Algorithmen

Es gibt Fälle, in denen es nicht möglich oder schlicht unpraktisch ist, ein Referenzsignal $x[k]$ zu messen. So kommt es beim ANC zum Beispiel, wird das Referenzsignal mittels eines Mikrofons erfasst, zur Rückkopplung des Filterausgangs $y[k]$ auf das Referenzsignalmikrofon. Das Referenzsignal müsste durch zusätzliche Maßnahmen von der Rückkopplung befreit werden.

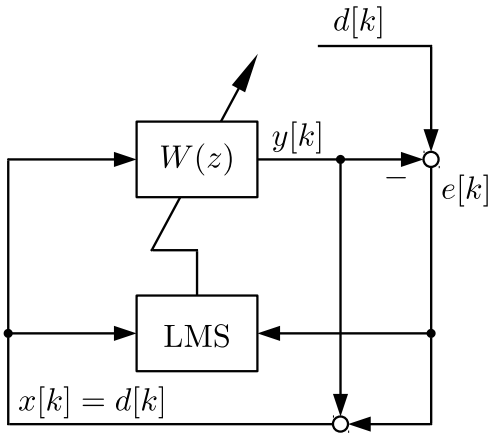

Abbildung 1.4: Blockdiagramm eines LMS-Filters mit Feedback

Eine Möglichkeit, dies zu umgehen, ist ein adaptives Feedback-System (siehe Abbildung 1.4). Bei diesem wird anstelle des Referenzsignals $x[k]$ das Primärsignal $d[k]$ als Referenzsignal verwendet. In vielen Fällen wird das Primärsignal nicht direkt erfasst, sondern lediglich das Fehlersignal $e[k]$. Für den Fall eines einfachen LMS-Systems kann das Primärsignal allerdings, wie die Umformung von Gleichung (1.14) zeigt, durch die Addition des Filterausgangs $y[k]$ zum Fehlersignal zurück gewonnen werden:

$$d[k] = e[k] + y[k] \tag{1.26}$$

$$x[k] = d[k] \tag{1.27}$$

Etwas aufwendiger ist die Umsetzung eines Feedback-Systems bei der Verwendung eines FxLMS-Algorithmus. Hier kann das Primärsignal $d[k]$ nicht durch einfache Addition des Filterausgangs $y[k]$ zum Fehlersignal zurück gewonnen werden. Betrachtet man einmal mehr alle Signale im Frequenzbereich, so ist mit Gleichung (1.21) die Zusammensetzung des Fehlersignals gegeben. Um aus dem Fehlersignal das Primärsignal zurück zu gewinnen, wird das bereits vorhandene Modell der Sekundärstrecke verwendet. Unter Annahme eines optimalen Modells folgt durch Umformen:

$$\widehat{S}_{\mathrm{opt}}(f) \stackrel{!}{=} S(f) \tag{1.28}$$

$$D(f) = E(f) + S(f)\,Y(f) \tag{1.29}$$

$$= E(f) + \widehat{S}_{\mathrm{opt}}(f)\,Y(f) \tag{1.30}$$

Mit Hilfe des Sekundärstreckenmodells $\widehat{S}(z)$ wird, indem das Sekundärsignal $\widehat{y'}[k] \approx y'[k]$ nachgebildet wird, in Näherung das Primärsignal $\widehat{d}[k] \approx d[k]$ aus dem Fehlersignal $e[k]$ zurück gewonnen, welches dann als Referenzsignal verwendet wird:

$$\widehat{y'}[k] = \widehat{\boldsymbol{s}}^{\mathsf{T}}\,\boldsymbol{y}[k] \tag{1.31}$$

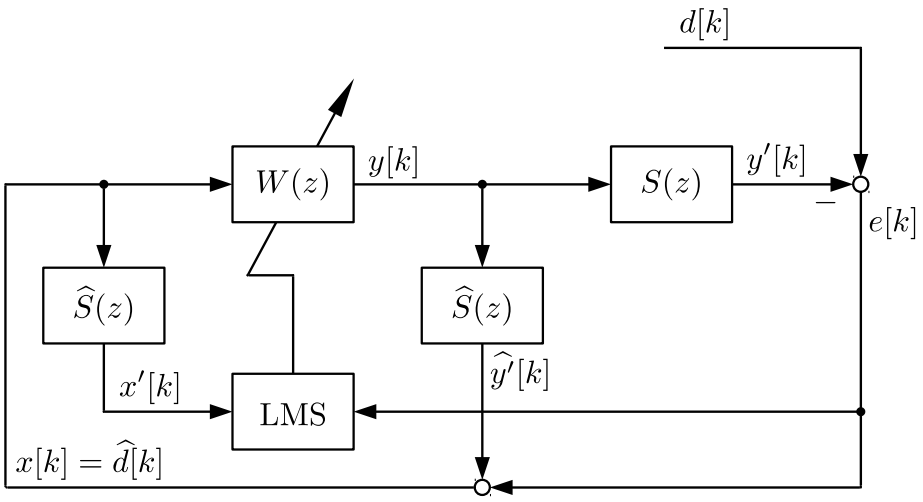

Abbildung 1.5: Blockdiagramm eines FxLMS-Filters mit Feedback

$$\widehat{d}[k] = e[k] + \widehat{y}'[k] \tag{1.32}$$

$$x[k] = \widehat{d}[k] \tag{1.33}$$

$\boldsymbol{y}[k]$ ist der Vektor der letzten N_S Ausgangswerte des Filters $W(z)$:

$$\boldsymbol{y}[k] = \begin{bmatrix} y[k] & y[k-1] & \cdots & y[k-N_S+1] \end{bmatrix}^{\mathsf{T}} \tag{1.34}$$

Das entsprechende Blockdiagramm eines FxLMS mit Feedback ist mit Abbildung 1.5 gegeben.

1.4.5 Stabilität von Feedback LMS-Algorithmen

Im Zusammenhang mit Feedback LMS-Algorithmen müssen zusätzliche Überlegungen bezüglich der Stabilität angestellt werden. Für die folgenden Überlegungen wird das Filter $W(z)$ als zeitinvariant angenommen, was aufgrund der vergleichsweise langsamen Konvergenz der Filterkoeffizienten durchaus zulässig ist. Betrachtet man das Blockdiagramm aus Abbildung 1.5 im Frequenzbereich, so setzt sich das Ausgangssignal $y[k] \circ\!\!-\!\!\bullet\, Y(f)$ wie folgt zusammen:

$$Y(f) = W(f)\left(D(f) + Y(f)\left(\widehat{S}(f) - S(f)\right)\right) \tag{1.35}$$

Durch Umstellen von Gleichung (1.35) kann man erkennen, wie der Filterausgang $Y(f)$ auf Störungen $D(f)$ reagiert:

$$Y(f) = W(f)\left(\widehat{S}(f) - S(f)\right)Y(f) + W(f)\,D(f) \tag{1.36}$$

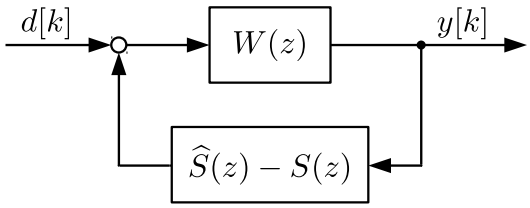

Abbildung 1.6: Blockdiagramm zur Verdeutlichung der Rückkopplung

$$\frac{Y(f)}{D(f)} = \frac{W(f)}{1 - W(f)\left(\widehat{S}(f) - S(f)\right)} \tag{1.37}$$

In Gleichung (1.37) ist zu erkennen, dass das Ausgangssignal auf den Filtereingang zurückgekoppelt wird. Noch deutlicher wird das, wenn man das entsprechende Blockdiagramm betrachtet (Abbildung 1.6). Man kann die Differenz von $\widehat{S}(f)$ und $S(f)$ als ein System auffassen. Es ist bekannt, dass das System mit Rückkopplung immer stabil ist, wenn für alle Frequenzen gilt:

$$\left| W(f)\left(\widehat{S}(f) - S(f)\right) \right| \overset{!}{<} 1 \tag{1.38}$$

Diese Forderung ist auf zwei Arten umsetzbar. Zu erkennen ist, dass für den Fall, dass für alle Frequenzen $S(f) \approx \widehat{S}(f)$ gilt, der Betrag in Gleichung (1.38) klein wird, da die Differenz klein ist. Das bedeutet, wenn das Sekundärstreckenmodell näherungsweise für die gesamte Bandbreite erstellt wurde, kann die Forderung in Gleichung (1.38) erfüllt werden.

Es gibt Fälle, in denen nur schmalbandige Störungen beseitigt werden sollen. In diesen Fällen bietet es sich an, die Sekundärstrecke auch nur für den entsprechenden Frequenzbereich zu identifizieren. Da für alle Frequenzen, für die das Sekundärstreckenmodell nicht identifiziert wurde, $S(f) \approx \widehat{S}(f)$ nicht mehr gilt, muss man sich behelfen. Ein Ansatz wäre es, ein Filter $H(z)$ in die Regelung einzubringen, dass alle zurückgekoppelten Frequenzanteile, für die $S(f) \not\approx \widehat{S}(f)$ gilt, in der Amplitude deutlich senkt, damit die Forderung in Gleichung (1.38) weiterhin erfüllt werden kann. Damit würde Gleichung (1.37) wie folgt um das Filter $H(z)$ erweitert werden:

$$\frac{Y(f)}{D(f)} = \frac{W(f)\,H(f)}{1 - W(f)\,H(f)\left(\widehat{S}(f) - S(f)\right)} \tag{1.39}$$

In Abbildung 1.7 ist das um das Filter $H(z)$ erweiterte Blockdiagramm dargestellt.

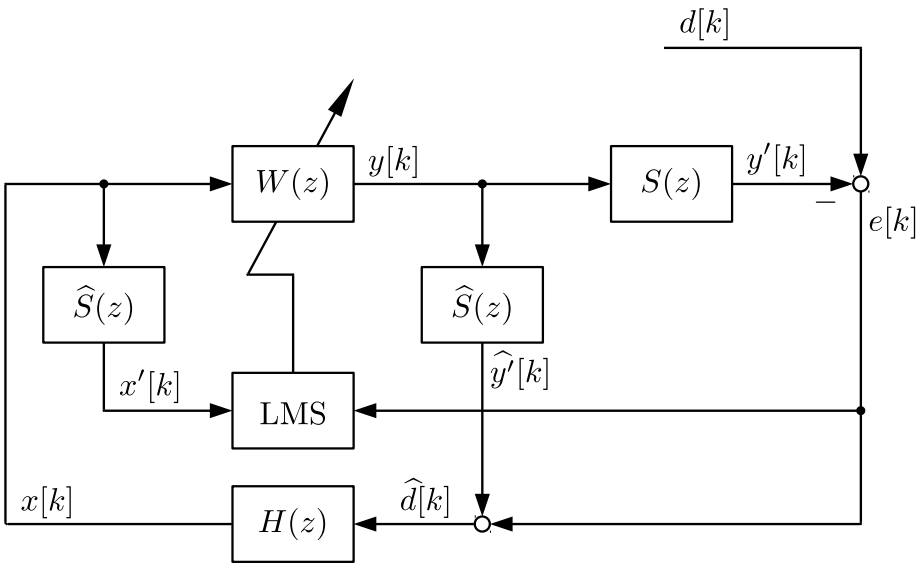

Abbildung 1.7: Blockdiagramm Feedback FxLMS mit Filter zur Sicherstellung der Stabilität

1.5 Active Noise Control

Das folgende Kapitel befasst sich mit der grundlegenden Funktion des Active Noise Control (ANC).

1.5.1 Grundlagen des Active Noise Control

Ziel von ANC ist es, mit Hilfe (mindestens) einer sekundären akustischen Quelle (Lautsprecher) unter Ausnutzung des Superpositionsprinzips dem Lärm einer primären Störquelle ein Signal so zu überlagern, dass es zu destruktiver Interferenz kommt und die Störung im Idealfall komplett kompensiert wird. Der Gegenschall muss, um eine wirksame Störsignalunterdrückung zu erzielen, möglichst genau das negierte Störsignal nachbilden. In Abbildung 1.8 sind ein Störsignal $d(t)$ und ein entsprechendes Signal zur Unterdrückung der Störung $y(t)$ durch destruktive Interferenz gezeigt. Übrig bleibt ein Restfehler $e(t) = d(t) + y(t)$, den es gilt, möglichst klein zu halten.

Ein klassisches ANC-System besteht, wie in Abbildung 1.9 gezeigt, aus einer Gegenschallquelle, der Signalverarbeitung, hier durch den DSP-Block angedeutet und einem Fehlermikrofon. An diesem gilt es, den Lärm durch das Aussenden von Gegenschall zu minimieren. Die Störquelle ist hier als Lautsprecher angedeutet. In der Praxis wird der Lärm zum Beispiel durch ungewollten Schall abstrahlende Struktu-

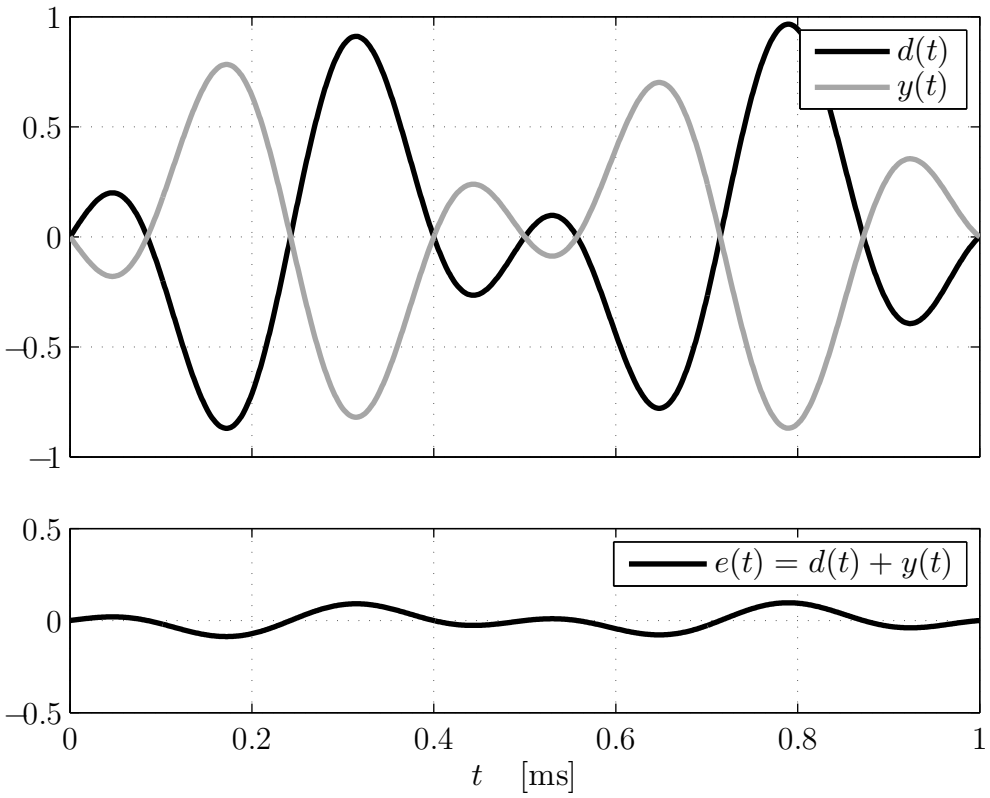

Abbildung 1.8: Beispiel destruktiver Interferenz

ren erzeugt. Über das Fehlersignal hinaus erhält die Signalverarbeitung ein Referenzsignal $x(t)$, anhand dessen der auszusendende Gegenschall geschätzt wird. Für diese Schätzung werden üblicherweise LMS-Algorithmen (siehe Kapitel 1.4) verwendet. Das Referenzsignal kann hierbei mit Hilfe eines Mikrofons, welches nah an der Störquelle platziert wird, erfasst werden, wobei dabei zu bedenken gilt, dass die Gegenschallquelle in diesem Fall teilweise auf das Referenzsignalmikrofon zurückkoppeln kann. Mitunter ist es möglich, ein Referenzsignal, welches mit dem am Fehlermikrofon gemessenen Primäranteil (Anteil der Störquelle) korreliert ist, auf andere Weise zu erfassen. Handelt es sich zum Beispiel durch schwingende Strukturen verursachten Lärm, kann eventuell ein Sensor zur Erfassung der Strukturschwingung eingesetzt werden.

Da sich die Referenzsignalerfassung teilweise als schwierig oder schlicht unpraktisch erweist, existieren so genannte Feedback-Systeme zur Durchführung von ANC (siehe Abbildung 1.10). Hierbei wird lediglich das Fehlersignal $e(t)$ gemessen und auf das direkte Messen eines Referenzsignals verzichtet. Feedback-LMS-Algorithmen sind in Kapitel 1.4.4 erläutert worden. Diese Systeme besitzen den Vorteil, dass sie lediglich ein Fehlermikrofon zur Signalerfassung benötigen, dafür aber höhere Anforderungen an die Stabilität der Algorithmen aufweisen. Da bei Feedback-Systemen das Aus-

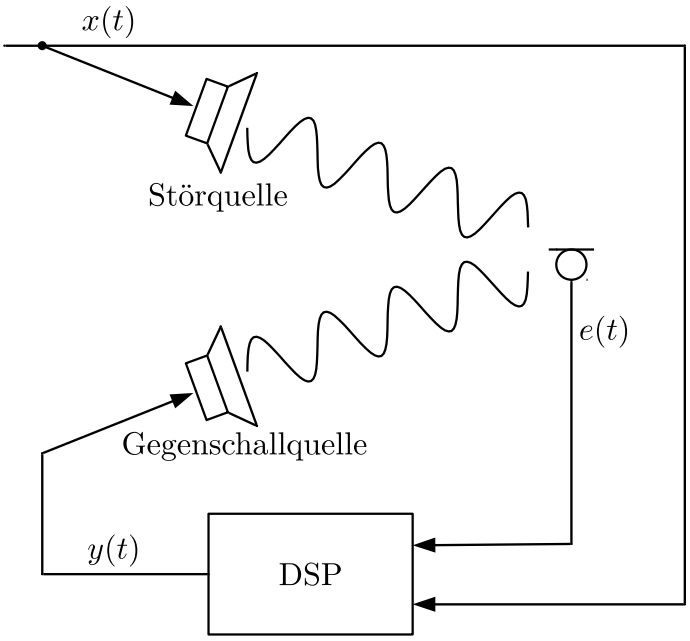

Abbildung 1.9: Schematische Darstellung eines ANC-Systems mit Referenzsignal

gangssignal auf den Eingang des adaptiven Filters zurückgekoppelt wird, spricht man auch von einer ANC-Regelung, wohingegen Feedforward-Systeme als Steuerung bezeichnet werden. Diese Art der Unterscheidung soll auch im weiteren Verlauf der Arbeit verwendet werden. Mitunter werden auch Feedforward-Systeme als Regelung bezeichnet, was man damit begründen kann, dass das Fehlersignal für die Adaption der Filterkoeffizienten zurückgeführt wird.

Bei den bisher besprochenen ANC-Systemen handelte es sich ausschließlich um einkanalige Systeme. Diese beinhalten ein Referenzsignal $x(t)$ (extern zugeführt oder intern erzeugt), ein Fehlersignal $e(t)$ und ein Ausgangssignal $y(t)$. Da mit solchen Systemen nur sehr lokal am Fehlermikrofon eine erkennbare Reduktion des Lärms erzielt werden kann, sind Systeme entwickelt worden, welche mehrere Fehlermikrofone besitzen, um an verschiedenen Stellen, beziehungsweise in einem größeren Bereich, den Lärm zu minimieren. Da solche Systeme an mehreren örtlich voneinander getrennten Punkten den Schalldruck minimieren sollen, kommen diese nicht mehr mit einer Gegenschallquelle aus und besitzen deshalb zusätzlich mehrere Ausgänge. Es existieren dabei Systeme, die entweder ein, aber auch mehrere Referenzsignale verwenden. Zu beachten gilt, dass die Signalverarbeitung für diese Fälle schnell deutlich komplexer werden kann. Mehrkanalige Systeme können abgekürzt als $J\times K\times M$-Systeme bezeichnet werden. Dabei steht J für die Anzahl der Referenzsignale, K für die Anzahl der Gegenschallquellen und M für die Anzahl der Fehlermikrofone. Einkanalige Systeme sind demzufolge $1\times1\times1$-Systeme.

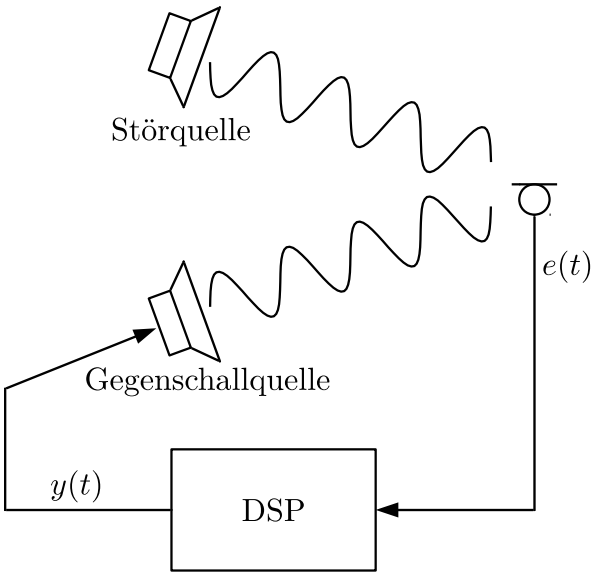

Abbildung 1.10: Schematische Darstellung eines ANC-Systems ohne Referenzsignal

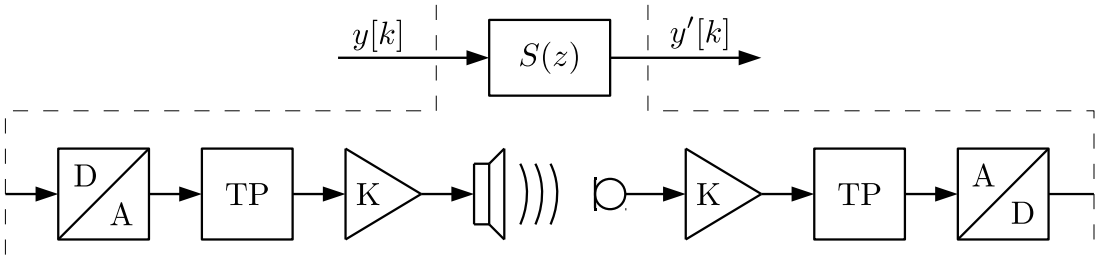

Abbildung 1.11: Komponenten der Sekundärstrecke

1.5.2 Einfluss der Sekundärstrecke

Der einfache LMS-Algorithmus ist für ANC-Systeme eher ungeeignet. Grund hierfür ist die Tatsache, dass das Ausgangssignal des adaptiven Filters nicht direkt auf den Summationspunkt des Fehlersignals wirkt, sondern eine Reihe von Komponenten durchläuft, die das Signal zum Beispiel wandeln oder verstärken sollen. In Abbildung 1.11 ist die Kette der wichtigsten Komponenten einer solchen Sekundärstrecke dargestellt. Diese besteht aus Digital-Analog-Umsetzer, Rekonstruktions-Tiefpass, Verstärker, Lautsprecher mit anschließender Übertragungsstrecke zum Mikrofon, Mikrofonvorverstärker, Anti-Aliasing-Tiefpass und Analog-Digital-Umsetzer. Um den Einfluss dieser Übertragungsstrecke mit zu berücksichtigen, bieten sich FxLMS-Algorithmen an (vergleiche Kapitel 1.4.3).

Komplexer gestaltet sich die Aufgabe der Erstellung eines Sekundärstreckenmodells bei Systemen mit mehreren Fehlermikrofonen und Gegenschallquellen. Einzelne Gegenschallquellen wirken jeweils nicht nur auf ein, sondern in der Regel auf alle

Fehlermikrofone, wobei das Übertragungsverhalten zwischen diesen im Allgemeinen unterschiedlich ist, da sich allein durch unterschiedliche Abstände zwischen Lautsprechern und Mikrofonen unterschiedliche Laufzeiten ergeben. Dies hat zur Folge, dass bei einem System mit M Fehlermikrofonen und K Ausgängen genau M mal K Sekundärstrecken identifiziert werden müssten.

1.6 Ziel der Arbeit

Folgend soll kurz das Ziel der Arbeit erläutert werden.

Wie bereits erwähnt, ist ein negativer Aspekt einer klassischen einkanaligen-ANC-Lösung, dass der Schalldruck nur über einen vergleichsweise kleinen Bereich am Mikrofon so beeinflusst wird, dass sich eine merkliche Reduktion der Schalldruckamplitude einstellt. Ziel dieser Arbeit ist es zu erforschen, inwieweit dieser Bereich durch Regeln einer zweiten physikalischen Größe vergrößert werden kann.

Die Idee, die hinter dieser Arbeit steht, wird in [2] vorgestellt. Darin wird beschrieben, dass ein klassisches ANC-System in einem diffusen Schallfeld eine Reduktion des Schalldruckes um 10 dB ungefähr innerhalb eines Durchmessers von $\frac{\lambda}{10}$ erzielen kann. Die Idee ist es, neben dem Schalldruck p auch den Schalldruckgradienten $\boldsymbol{\nabla} p$ zu regeln. Dieser beinhaltet die ersten Ableitungen nach dem Ort und beschreibt somit die örtliche Änderung des Schalldrucks. Werden sowohl der Schalldruck als auch der Schalldruckgradient minimiert, so sollte der Schalldruck mit zunehmender Entfernung vom Punkt, an welchem geregelt wird, im Vergleich schwächer ansteigen. Theoretisch soll so ein um 10 dB reduzierten Bereich von $\frac{\lambda}{2}$ erzielt werden können.

Im Zuge dieser Arbeit soll ein Versuch entstehen, um mit Hilfe des zur Verfügung stehenden DSP Development Boards *TMS320C6713* verschiedene ANC-Algorithmen zu implementieren und deren Wirksamkeit mit einander zu vergleichen.

2 Vorüberlegungen und Versuchsaufbau

2.1 Verwendete Hardware

Im Folgenden soll kurz die verwendete Hardware vorgestellt werden.

Für die Arbeit steht ein DSP Development Board *TMS320C6713* zur Verfügung. Kernstück des Boards ist der mit 225 MHz getaktete gleichnamig Gleitkomma-DSP der Firma Texas Instruments.

Weiter befindet sich auf dem Board ein Stereo Audio Codec nach dem AIC23 Standard, welcher für das Einlesen und Ausgeben der Signale genutzt wird. Dafür werden die Line-In und Line-Out Anschlüsse, welche über 3,5 mm Klinkenbuchsen mit externen Komponenten verbunden werden können, des Codecs verwendet. Dadurch stehen jeweils zwei Ein- und Ausgangskanäle zur Verfügung. Der Codec übernimmt die Digital-Analog-, so wie die Analog-Digital-Umsetzung der Ein- und Ausgangssignale. Es können unterschiedliche Abtastraten im Bereich zwischen 8 und 96 kHz gewählt werden.

Als Entwicklungsumgebung wird *Code Composer Studio* in der Version 5.5, eine von Texas Instruments entwickelte Umgebung zum Programmieren und Debuggen, verwendet. Auf dem Board befindet sich ein JTAG Emulator mit USB-Interface, über welchen Programme aus der Entwicklungsumgebung auf das Board übertragen und debuggt werden können. Zur schnellen Inbetriebnahme des Boards und einfachen Verwendung des Codecs werden die mit [1] zur Verfügung gestellten Dateien verwendet. Die Hardware zur Vorverstärkung der Mikrofonsignale und der Verstärkung der Lausprechersignale ist in einem Instrumentengehäuse untergebracht. Es stehen folgende Anschlüsse zur Verfügung (vergleiche Abbildung 2.1):

(1) Lautsprecherausgang links

(2) Lautsprecherausgang rechts

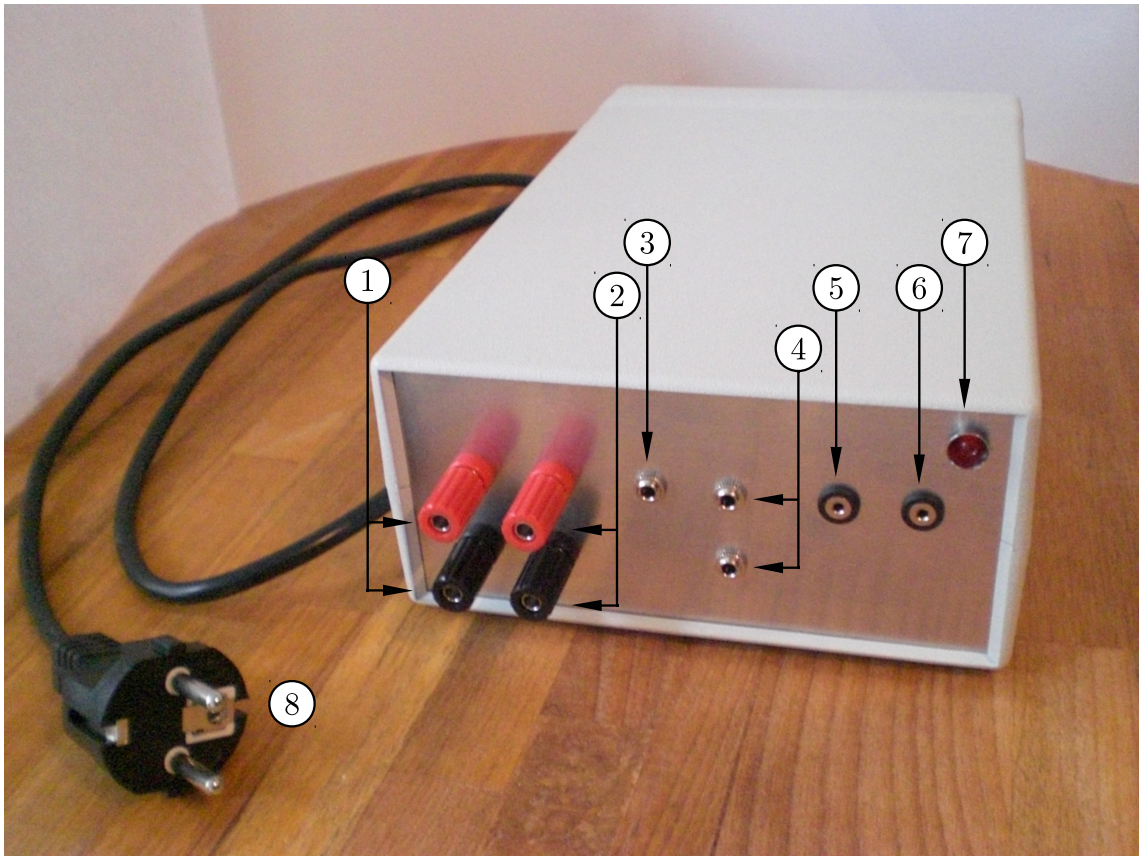

Abbildung 2.1: Bild des Instrumentengehäuses mit der Elektronik zur Signalverstärkung

(3) Lautsprechersignaleingang - 3,5 mm Klinkenbuchse - dreipolig

(4) Mikrofonsignalausgang - 2 × 3,5 mm Klinkenbuchse - dreipolig

(5) Mikrofoneingang 1 - 2,5 mm Klinkenbuchse - zweipolig

(6) Mikrofoneingang 2 - 2,5 mm Klinkenbuchse - zweipolig

(7) Betriebs-LED

(8) Netzstecker

Die Anschlüsse (1) und (2) sind für den Anschluss von Lautsprechern gedacht. Über sie werden die an (3) eingegebenen Audiosignale verstärkt ausgegeben. Als Verstärker wird je Kanal ein 3,5 W Verstärkermodul *M031N* der Firma *Kemo* eingesetzt. Dementgegen kann an die Anschlüsse (5) und (6) jeweils eine Elektretmikrofon-

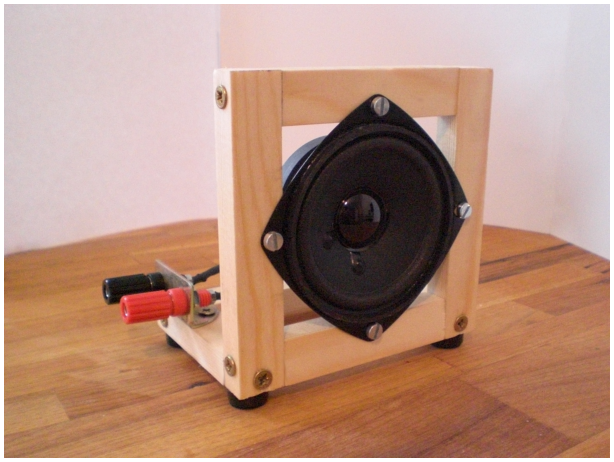

Abbildung 2.2: Bild einer verwendeten Sekundärquelle

kapsel angeschlossen werden, die es gilt vorzuverstärken und mit einer Phantomspannung zu versorgen. Die vorverstärkten Mikrofonsignale werden parallel über die beiden Anschlüsse (4) ausgegeben. Da das Mikrofonsignal von einer monofrequenten Störung von 50 Hz überlagert ist, welche vermutlich aus einer nicht perfekt geglätteten Versorgungsspannung resultiert, die Störung aber nicht im Arbeitsbereich des Systems liegt, wird ein softwaretechnischer Hochpass verwendet, um diese zu unterdrücken. Als Vorverstärker kommt je Kanal ein Mono-Mikrofon Vorverstärker-Bausatz der Firma *Conrad Electronic* zum Einsatz. In Abbildung 2.2 ist eine Sekundärquelle gezeigt. Beim verwendeten Lautsprecher handelt es ich um einen *FR 8* der Firma *Visaton*.

2.2 Vergleichen der Algorithmen und Erfassen der Eingangssignale

Wie beschrieben, sollen verschiedene ANC-Algorithmen mit einander verglichen werden. Um dies zu tun, wird das Schallfeld um den Punkt, an dem der Schalldruck gezielt beeinflusst wird, erfasst. Dies geschieht, indem mit einer externen Mikrofonkapsel der Schalldruckpegel an mehreren Messstellen erfasst wird. Die Messstellen sind in einem Radius von 0 bis 200 mm in Abständen von 25 mm , jeweils um 22,5 ° versetzt, angeordnet. Die Messpunkte liegen alle in einer Ebene. Der Punkt $\boldsymbol{x}_0 = [\, x_0 \quad y_0 \,]^\mathsf{T}$ stellt den Mittelpunkt des Kreises dar.

Die Messungen erfolgen in einem Raum mit der ungefähren Abmessung von 3,2 m × 5,5 m × 2,5 m .

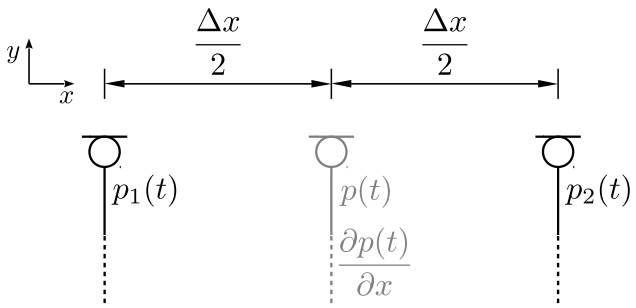

Abbildung 2.3: Approximation der partiellen Ableitung nach einer Raumrichtung

Alle Messungen werden mit monofrequenten Störungen durchgeführt. Die Frequenz der Störung liegt bei 570 Hz . Daraus ergibt sich, dass lediglich zwei Filterkoeffizienten pro Filter nötig sind, da Filter 1. Ordnung monofrequente Signale unabhängig in Amplitude und Phase beeinflussen können.

Bei den in der Arbeit verwendeten Algorithmen gibt es zwei verschiedene physikalische Größen, welche aktiv beeinflusst werden sollen und dafür erfasst werden müssen.

Die Erste ist der Schalldruck $p(t)$. Dieser kann direkt mit einem Mikrofon gemessen werden, da das mit dem Mikrofon erfasste Signal diesem entspricht (siehe Kapitel 1.3).

Bei der zweiten physikalischen Größe handelt es sich um den Druckgradienten $\boldsymbol{\nabla} p(t)$. In der Arbeit wird sich dabei auf eine Raumrichtung beschränkt, womit die zu erfassende Größe eine skalare ist:

$$\frac{\partial p(t)}{\partial x}$$

Die Raumrichtung, in welcher der Druck partiell abgeleitet wird, kann hierbei erst einmal als frei wählbar betrachtet werden. Die direkte Erfassung des Druckgradienten oder der mit dem Gradienten in Beziehung stehenden Schallschnelle gestaltet sich relativ schwierig. Eine vergleichsweise einfache Methode ist es, den Differentialquotienten durch einen Differenzenquotienten anzunähern. Für die Ableitung entlang einer Raumachse werden dafür zwei Mikrofone benötigt. Das Prinzip ist in Abbildung 2.3 dargestellt. Es werden die zwei Drück $p_1(t)$ und $p_2(t)$ erfasst, aus denen dann der Druck und die partielle Ableitung nach der Raumrichtung an einem gedanklich mittig von beiden Mikrofonen platzierten Mikrofon, in der Abbildung in Grau dargestellt, errechnet werden kann.

$$p(t) \approx \frac{p_1(t) + p_2(t)}{2} \tag{2.1}$$

$$\frac{\partial p(t)}{\partial x} \approx \frac{p_2(t) - p_1(t)}{\Delta x} \tag{2.2}$$

Die Mikrofone müssen einem von der zu messenden Frequenz abhängigen Abstand Δx aufweisen. Dieser muss zum einen so gewählt werden, dass ein Unterschied erfasst werden kann, darf aber auf der anderen Seite nicht zu groß werden, da ansonsten der Differentialquotient nicht mehr zuverlässig angenähert wird. Diese Methode eignet sich folglich nur für die Erfassung eher schmalbandiger Signale. Eine Faustregel für das Verhältnis von Abstand zu Wellenlänge besagt:

$$\frac{\Delta x}{\lambda} \leq \frac{1}{6} \tag{2.3}$$

Die Achse, welche durch beide Mikrofone geht, bestimmt die Raumrichtung, in welcher die Ableitung approximiert wird.

Da es bei der Regelung von Druck und Druckgradient darum geht, diese zu minimieren, werden als Eingangssignale lediglich die Summe und die Differenz heran gezogen, welche jeweils proportional zu den zu minimierenden Größen sind.

$$\frac{p_1(t) + p_2(t)}{2} \sim p_1(t) + p_2(t) \tag{2.4}$$

$$\frac{p_2(t) - p_1(t)}{\Delta x} \sim p_2(t) - p_1(t) \tag{2.5}$$

Der Zusammenhang von Schalldruckgradient und Schallschnelle ist mit Gleichung (1.5) beziehungsweise (1.6) gegeben. Der Schalldruckgradient ist proportional zur zeitlichen Änderung der Schallschnelle. Minimiert man den Schalldruckgradienten, beeinflusst man dadurch indirekt die Schallschnelle, da man deren zeitliche Änderung minimiert.

Für alle verwendeten Mikrofone wurde ein Mikrofon Kalibrator genutzt, um von den erfassten Signalen auf Schalldrücke schließen zu können.

3 Implementierte adaptive Algorithmen

3.1 Einkanalige adaptive Algorithmen

Nachfolgend sind die beiden implementierten einkanaligen ANC-Algorithmen beschrieben, sogenannte 1×1×1-Systeme. Diese Algorithmen verwenden lediglich ein Fehlermikrofon und eine Gegenschallquelle.

3.1.1 Sekundärstreckenidentifikation bei einkanaligen Systemen

Die Sekundärstreckenidentifikation bei einkanaligen Systemen wird, unabhängig davon, ob es sich um eine Steuerung oder Regelung handelt, in beiden Fällen gleich durchgeführt. Die Identifikation erfolgt mittels eines NLMS-Algorithmus. Ein Blockdiagramm ist mit Abbildung 3.1 gegeben. $S(z)$ ist die Übertragungsfunktion, welche das Verhalten der Sekundärstrecke beschreibt. Ziel ist es, ein Modell $\widehat{S}(z)$ selbiger zu erstellen. Alle von der gestrichelten Linie umschlossenen Operationen werden auf dem DSP durchgeführt. Es wird ein internes Referenzsignal $x[k]$ gleichzeitig über die Gegenschallquelle ausgegeben und auf das Modellfilter gegeben. Das Filterausgangssignal $y[k]$ wird von dem am Fehlermikrofon eingelesenen erwünschten Signal $d[k]$ abgezogen und so das Fehlersignal $e[k]$ gebildet, welches neben dem Referenzsignal in die LMS-Updategleichung eingeht. Folgende Berechnungen werden während der Identifikation zyklisch auf dem DSP durchgeführt:

$$y[k] = \widehat{\boldsymbol{s}}^{\mathsf{T}}[k]\,\boldsymbol{x}[k] \tag{3.1}$$

$$e[k] = d[k] - y[k] \tag{3.2}$$

$$\mu[k] = \frac{\beta}{\gamma + \boldsymbol{x}^{\mathsf{T}}[k]\,\boldsymbol{x}[k]} \tag{3.3}$$

$$\widehat{\boldsymbol{s}}[k{+}1] = \widehat{\boldsymbol{s}}[k] + \mu[k]\,e[k]\,\boldsymbol{x}[k] \tag{3.4}$$

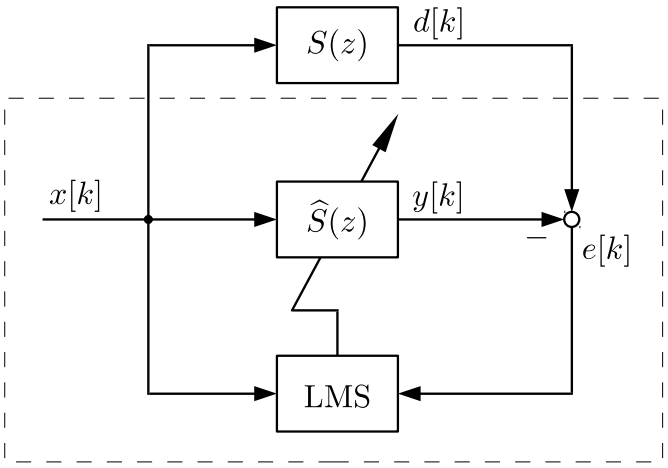

Abbildung 3.1: Blockdiagramm der Sekundärstreckenidentifikation bei einkanaligen adaptiven Algorithmen

β bestimmt hierbei die Schrittweite der Updategleichung, γ ist eine Sicherheitskonstante, um unter anderem eine Division durch Null zu vermeiden (siehe Kapitel 1.4.2).

3.1.2 Einkanalige adaptive Steuerung

Der erste implementierte adaptive Algorithmus ist eine einkanalige Steuerung, sprich ein 1×1×1-Feedforward-Algorithmus. Es wird mit einem intern erzeugten Referenzsignal $x[k]$ gearbeitet. Das zuvor erstellte Sekundärstreckenmodell $\widehat{S}(z)$ wird für den verwendeten normalisierten FxLMS-Algorithmus herangezogen. Das Blockdiagramm zum Algorithmus ist in Abbildung 3.2 zu sehen. Wieder sind alle Vorgänge, welche auf dem DSP ablaufen mit einer gestrichelten Linie umschlossen. Zu beachten gilt es, dass, im Gegensatz zum in Kapitel 1.4.3 beschriebenen FxLMS-Algorithmus, sich das Fehlersignal $e[k]$ aus der Addition von unerwünschtem Primärsignal $d[k]$ und mit der Sekundärstrecke gefiltertem Filterausgangssignal $y'[k]$ ergibt. Das hat zur Folge, dass sich in den durchzuführenden Rechnungen einige Vorzeichenwechsel ergeben. Geschuldet ist dies der Tatsache, dass sich die Schalldrücke am Fehlermikrofon additiv überlagern. Somit ergeben sich folgende zyklisch durchzuführende Berechnungen:

$$y[k] = \boldsymbol{w}^{\mathsf{T}}[k]\,\boldsymbol{x}[k] \tag{3.5}$$

$$x'[k] = \widehat{\boldsymbol{s}}^{\,\mathsf{T}}\,\boldsymbol{x}[k] \tag{3.6}$$

$$\mu[k] = \frac{\beta}{\gamma + \boldsymbol{x'}^{\,\mathsf{T}}[k]\,\boldsymbol{x'}[k]} \tag{3.7}$$

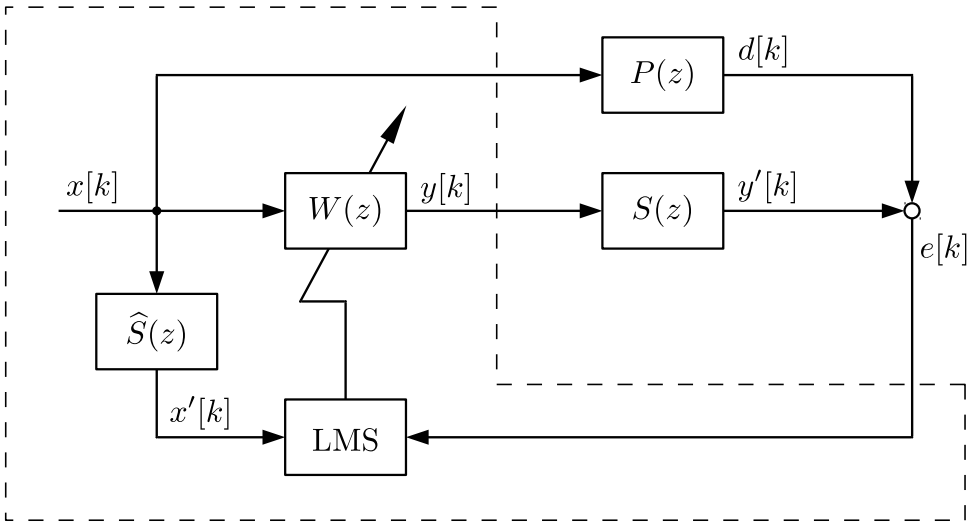

Abbildung 3.2: Blockdiagramm der einkanaligen adaptiven Steuerung

$$\boldsymbol{w}[k{+}1] = \boldsymbol{w}[k] - \mu[k]\, e[k]\, \boldsymbol{x}'[k] \tag{3.8}$$

Bei $e[k]$ handelt es sich um das mit dem Fehlermikrofon erfasste Signal, $y[k]$ ist das Signal, welches ausgegeben wird. Da mit interner Referenz gearbeitet wird, wird das Referenzsignal $x[k]$ über den zweiten Ausgang ausgegeben, um das Störsignal $d[k]$ zu erzeugen.

3.1.3 Einkanalige adaptive Regelung

Als zweiter einkanaliger Algorithmus wurde eine adaptive Regelung, ein 1×1×1-Feedback-Algorithmus, umgesetzt (Blockdiagramm siehe Abbildung 3.3). Der entscheidende Unterschied besteht hier darin, dass das Referenzsignal nicht direkt bekannt ist, sondern aus dem Fehlersignal $e[k]$ erzeugt wird. Daraus resultiert ein im Vergleich zur Steuerung etwas umfangreicherer Algorithmus:

$$y[k] = \boldsymbol{w}^{\mathsf{T}}[k]\, \widehat{\boldsymbol{d}}[k] \tag{3.9}$$

$$\widehat{y'}[k] = \widehat{\boldsymbol{s}}^{\mathsf{T}}\, \boldsymbol{y}[k] \tag{3.10}$$

$$\widehat{d}[k] = e[k] - \widehat{y'}[k] \tag{3.11}$$

$$x'[k] = \widehat{\boldsymbol{s}}^{\mathsf{T}}\, \widehat{\boldsymbol{d}}[k] \tag{3.12}$$

$$\mu[k] = \frac{\beta}{\gamma + \boldsymbol{x'}^{\mathsf{T}}[k]\, \boldsymbol{x}'[k]} \tag{3.13}$$

$$\boldsymbol{w}[k{+}1] = \boldsymbol{w}[k] - \mu[k]\, e[k]\, \boldsymbol{x}'[k] \tag{3.14}$$

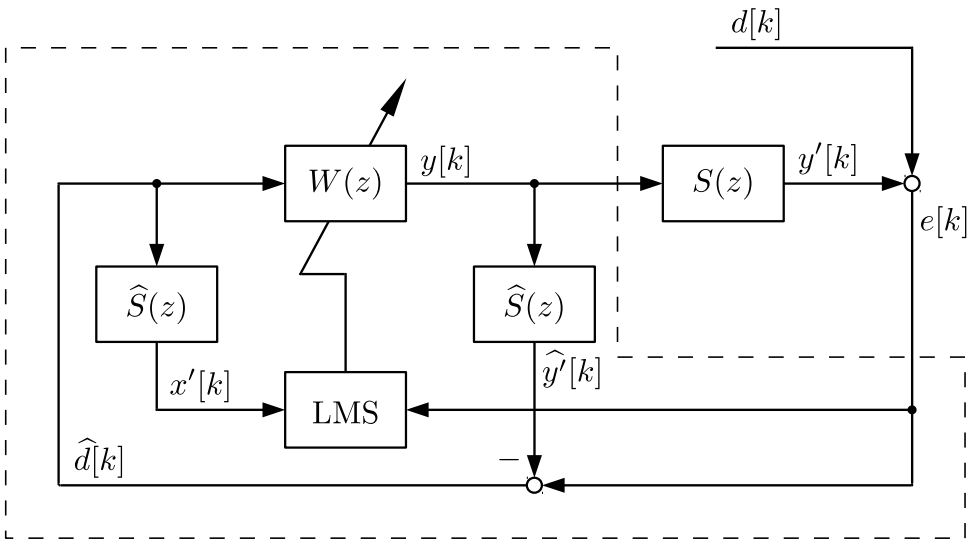

Abbildung 3.3: Blockdiagramm der einkanaligen adaptiven Regelung

Das Grundgerüst des Regelungsalgorithmus ist ähnlich dem der Steuerung. Die wesentlichen Unterschiede sind Gleichung (3.10) und (3.11), welche zur Erzeugung des Referenzsignals, das der rekonstruierten Störung $\widehat{d}[k]$ entspricht, dienen. Nicht abgebildet und aufgeführt ist der Übersicht halber das zur Sicherstellung der Stabilität zusätzlich eingesetzte Filter (vergleiche Kapitel 1.4.5), welches mit der Funktion des eigentlichen Algorithmus nichts zu tun hat.

3.2 Mehrkanalige adaptive Algorithmen

Neben den einkanaligen Algorithmen wurden weiter zwei mehrkanalige Algorithmen implementiert, deren Funktionsweise im Folgenden kurz beschrieben wird.

3.2.1 Sekundärstreckenidentifikation bei mehrkanaligen Systemen

Wie schon in Kapitel 1.5.2 angemerkt, gestaltet sich die Sekundärstreckenidentifikation bei mehrkanaligen Systemen etwas komplexer, da in der Regel alle Ausgangssignale auch auf alle Eingänge wirken.

Dies hat zur Folge, dass im Falle der adaptiven Steuerung mit einem Ausgangssignal und zwei Fehlersignalen zwei Sekundärstreckenmodelle, $S_1(z)$ und $S_2(z)$, erstellt werden müssen. Vorgegangen wird dabei genau wie bei einkanaligen Systemen,

indem ein Signal über den Ausgang ausgegeben wird und dabei beide Sekundärstreckenmodelle parallel identifiziert werden. Dass der Ausgang auf beide Eingänge wirkt und somit zwei Sekundärstrecken identifiziert werden müssen, wird auch aus Abbildung 3.4, welche das Blockdiagramm der adaptiven Steuerung zeigt, ersichtlich.

Bei der implementierten mehrkanaligen Regelung gestaltet sich die Modellerstellung noch ein wenig aufwendiger. Das System hat neben zwei Eingängen auch zwei Ausgänge, woraus eine Anzahl von vier Sekundärstrecken resultiert. Die Identifikation wird hier in zwei Schritten durchgeführt. Als Erstes wird ein Signal über den ersten Ausgang ausgegeben. Dabei werden die beiden Sekundärstrecken $S_{11}(z)$ und $S_{21}(z)$, welche das Übertragungsverhalten vom ersten Ausgang auf den ersten und den zweiten Eingang beschreiben, parallel identifiziert. Im zweiten Schritt werden dann die Modelle der Sekundärstrecken $S_{12}(z)$ und $S_{22}(z)$ ermittelt, indem ein Signal über den zweiten Ausgang ausgegeben wird. Der Zusammenhang der Ausgangssignale $y_1[k]$ und $y_2[k]$ mit den Fehlersignalen $e_1[k]$ und $e_2[k]$ ist in Abbildung 3.5 zu sehen. Die Identifikation einer einzelnen Sekundärstrecke läuft dabei wieder wie beim einkanaligen Fall ab.

Der Unterschied zwischen einer Druck-Druck und einer Druck-Druckgradient Steuerung/Regelung liegt bei der Sekundärstreckenidentifikation im erwünschten Signal $d_1[k]$ und $d_2[k]$. Handelt es sich um eine Druck-Druck Steuerung/Regelung gilt für diese:

$$d_1[k] = p_1[k] \qquad d_2[k] = p_2[k] \tag{3.15}$$

Bei einer Druck-Druckgradient Steuerung/Regelung gilt:

$$d_1[k] = p_1[k] + p_2[k] \qquad d_2[k] = p_2[k] - p_1[k] \tag{3.16}$$

$p_1[k]$ und $p_2[k]$ stehen für die von den beiden Mikrofonen erfassten Signale.

3.2.2 Mehrkanalige adaptive Steuerung

Die adaptive Steuerung mit zwei Eingängen, ein 1×1×2-Feedforward-System, ist im Folgenden beschrieben und das zugehörige Blockdiagramm in Abbildung 3.4 zu sehen. Wesentlicher Unterschied zur einkanaligen Steuerung ist Gleichung (3.22), die Updategleichung der Filterkoeffizienten. In diese gehen beide Fehlersignale $e_1[k]$ und $e_2[k]$ und das jeweils mit dem entsprechenden Sekundärstreckenmodell gefilterte Referenzsignal $x'_1[k]$ und $x'_2[k]$ ein. Hieraus ergibt sich die Notwenigkeit, dass das sowohl mit $S_1(Z)$ als auch mit $S_2(Z)$ gefilterte Referenzsignal, sowie die zwei entsprechenden Schrittweiten $\mu_1[k]$ und $\mu_2[k]$ berechnet werden müssen.

$$y[k] = \boldsymbol{w}^{\mathsf{T}}[k]\, \boldsymbol{x}[k] \tag{3.17}$$

$$x'_1[k] = \widehat{\boldsymbol{s}}_1^{\mathsf{T}}\, \boldsymbol{x}[k] \tag{3.18}$$

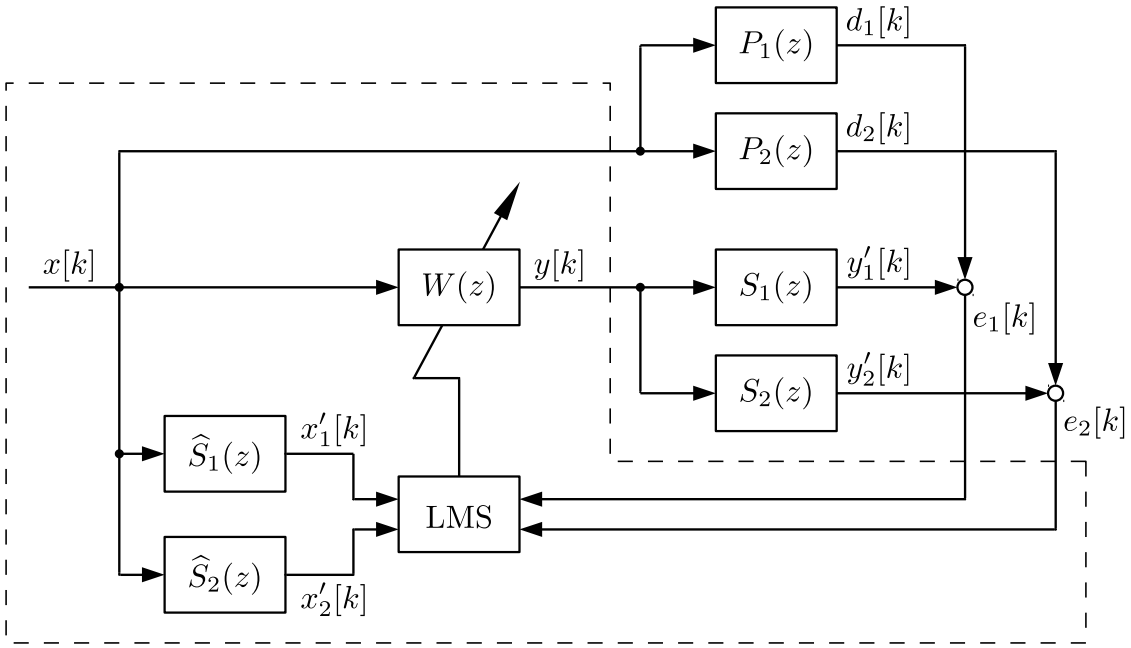

Abbildung 3.4: Blockdiagramm der adaptiven Steuerung mit zwei Eingängen

$$x_2'[k] = \widehat{\boldsymbol{s}}_2^{\mathsf{T}}\, \boldsymbol{x}[k] \tag{3.19}$$

$$\mu_1[k] = \frac{\beta}{\gamma + \boldsymbol{x}_1'^{\,\mathsf{T}}[k]\, \boldsymbol{x}_1'[k]} \tag{3.20}$$

$$\mu_2[k] = \frac{\beta}{\gamma + \boldsymbol{x}_2'^{\,\mathsf{T}}[k]\, \boldsymbol{x}_2'[k]} \tag{3.21}$$

$$\boldsymbol{w}[k{+}1] = \boldsymbol{w}[k] - \mu_1[k]\, e_1[k]\, \boldsymbol{x}_1'[k] - \mu_2[k]\, e_2[k]\, \boldsymbol{x}_2'[k] \tag{3.22}$$

Die Fehlersignale $e_1[k]$ und $e_2[k]$ setzten sich, abhängig von der Art der Steuerung, unterschiedlich zusammen. Bei einer Druck-Druck Steuerung gilt:

$$e_1[k] = p_1[k] \qquad e_2[k] = p_2[k] \tag{3.23}$$

Bei einer Druck-Druckgradient Steuerung gilt:

$$e_1[k] = p_1[k] + p_2[k] \qquad e_2[k] = p_2[k] - p_1[k] \tag{3.24}$$

3.2.3 Mehrkanalige adaptive Regelung

Den aufwendigsten implementierten Algorithmus stellt die adaptive Regelung mit zwei Ein- und Ausgängen, ein 1×2×2-Feedback-System, dar. Wie bereits zuvor besprochen, besitzt das System insgesamt vier Sekundärstrecken. Der Übersicht halber

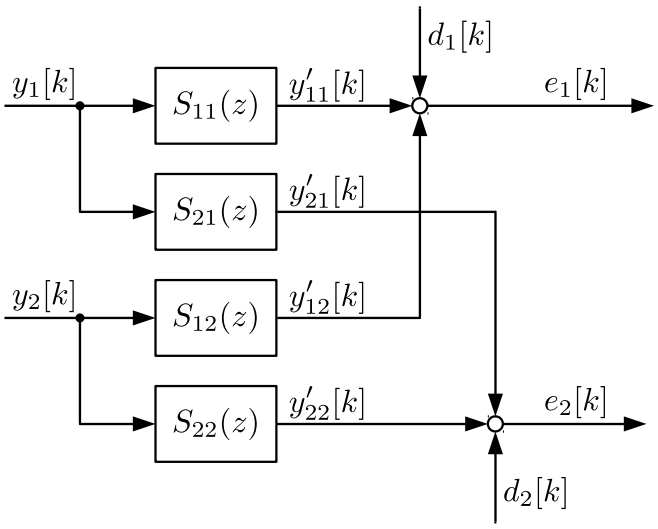

Abbildung 3.5: Blockdiagramm der Sekundärstrecken bei zwei Ein- und Ausgängen

ist das komplette System in zwei Blockdiagrammen dargestellt. Abbildung 3.5 zeigt den Zusammenhang der Ausgänge $y_1[k]$ und $y_2[k]$ mit den Eingängen $e_1[k]$ und $e_2[k]$ und den Einfluss der Störungen $d_1[k]$ und $d_2[k]$ auf selbige und beschreibt folglich die physikalischen Vorgänge im System. Abbildung 3.6 beinhaltet dagegen den kompletten Algorithmus, sprich alle Vorgänge, die auf dem DSP stattfinden.

Der Algorithmus nutzt, gegenüber einem auch möglichen 2×2×2-System, wie alle zuvor besprochenen Algorithmen lediglich ein Referenzsignal. In diesem Fall gibt es verschiedene Möglichkeiten, dieses zu bilden. Es besteht die Möglichkeit, dieses entweder aus einem der beiden Fehlersignale $e_1[k]$ beziehungsweise $e_2[k]$ oder aus einer Kombination aus beiden zu erzeugen. Der implementierte Algorithmus nutzt das aus dem Fehlersignal $e_1[k]$ rekonstruierte Störsignal $\widehat{d}_1[k]$ als Referenzsignal.

Aus der Tatsache, dass das System zwei Ausgänge besitzt, ergibt sich, dass auch zwei adaptive Filter $W_1(z)$ und $W_2(z)$ vorhanden sein müssen, eines für jeden Ausgang. Für zwei adaptive Filter ergeben sich zwei Updategleichungen (Gleichungen (3.38) und (3.39)), in welche jeweils beide Fehlersignale $e_1[k]$ und $e_2[k]$ eingehen. Es sind vier Sekundärstreckenmodelle für die FxLMS-Algorithmen notwendig, wobei aufgrund der gewählten Methode zur Referenzsignalerzeugung lediglich zwei davon für diese herangezogen werden.

$$y_1[k] = \boldsymbol{w}_1^{\mathsf{T}}[k]\, \widehat{\boldsymbol{d}}_1[k] \tag{3.25}$$

$$y_2[k] = \boldsymbol{w}_2^{\mathsf{T}}[k]\, \widehat{\boldsymbol{d}}_1[k] \tag{3.26}$$

$$\widehat{y'}_{11}[k] = \widehat{\boldsymbol{s}}_{11}^{\mathsf{T}}\, \boldsymbol{y}_1[k] \tag{3.27}$$

$$\widehat{y'}_{12}[k] = \widehat{\boldsymbol{s}}_{12}^{\mathsf{T}}\, \boldsymbol{y}_2[k] \tag{3.28}$$

$$\widehat{d}_1[k] = e_1[k] - \widehat{y'}_{11} - \widehat{y'}_{12} \tag{3.29}$$

$$x'_{11}[k] = \widehat{\boldsymbol{s}}_{11}^{\mathsf{T}}\, \widehat{\boldsymbol{d}}_1[k] \tag{3.30}$$

$$x'_{12}[k] = \widehat{\boldsymbol{s}}_{21}^{\mathsf{T}}\, \widehat{\boldsymbol{d}}_1[k] \tag{3.31}$$

$$x'_{21}[k] = \widehat{\boldsymbol{s}}_{12}^{\mathsf{T}}\, \widehat{\boldsymbol{d}}_1[k] \tag{3.32}$$

$$x'_{22}[k] = \widehat{\boldsymbol{s}}_{22}^{\mathsf{T}}\, \widehat{\boldsymbol{d}}_1[k] \tag{3.33}$$

$$\mu_{11}[k] = \frac{\beta}{\gamma + \boldsymbol{x}'_{11}{}^{\mathsf{T}}[k]\, \boldsymbol{x}'_{11}[k]} \tag{3.34}$$

$$\mu_{12}[k] = \frac{\beta}{\gamma + \boldsymbol{x}'_{12}{}^{\mathsf{T}}[k]\, \boldsymbol{x}'_{12}[k]} \tag{3.35}$$

$$\mu_{21}[k] = \frac{\beta}{\gamma + \boldsymbol{x}'_{21}{}^{\mathsf{T}}[k]\, \boldsymbol{x}'_{21}[k]} \tag{3.36}$$

$$\mu_{22}[k] = \frac{\beta}{\gamma + \boldsymbol{x}'_{22}{}^{\mathsf{T}}[k]\, \boldsymbol{x}'_{22}[k]} \tag{3.37}$$

$$\boldsymbol{w}_1[k{+}1] = \boldsymbol{w}_1[k] - \mu_{11}[k]\, e_1[k]\, \boldsymbol{x}'_{11}[k] - \mu_{12}[k]\, e_2[k]\, \boldsymbol{x}'_{12}[k] \tag{3.38}$$

$$\boldsymbol{w}_1[k{+}1] = \boldsymbol{w}_1[k] - \mu_{21}[k]\, e_1[k]\, \boldsymbol{x}'_{21}[k] - \mu_{22}[k]\, e_2[k]\, \boldsymbol{x}'_{22}[k] \tag{3.39}$$

Auch hier gilt wieder, dass sich das Fehlersignal ja nach Art der Regelung ergibt. Für eine Druck-Druck Regelung gilt (3.23), für eine Druck-Druckgradient Regelung (3.24).

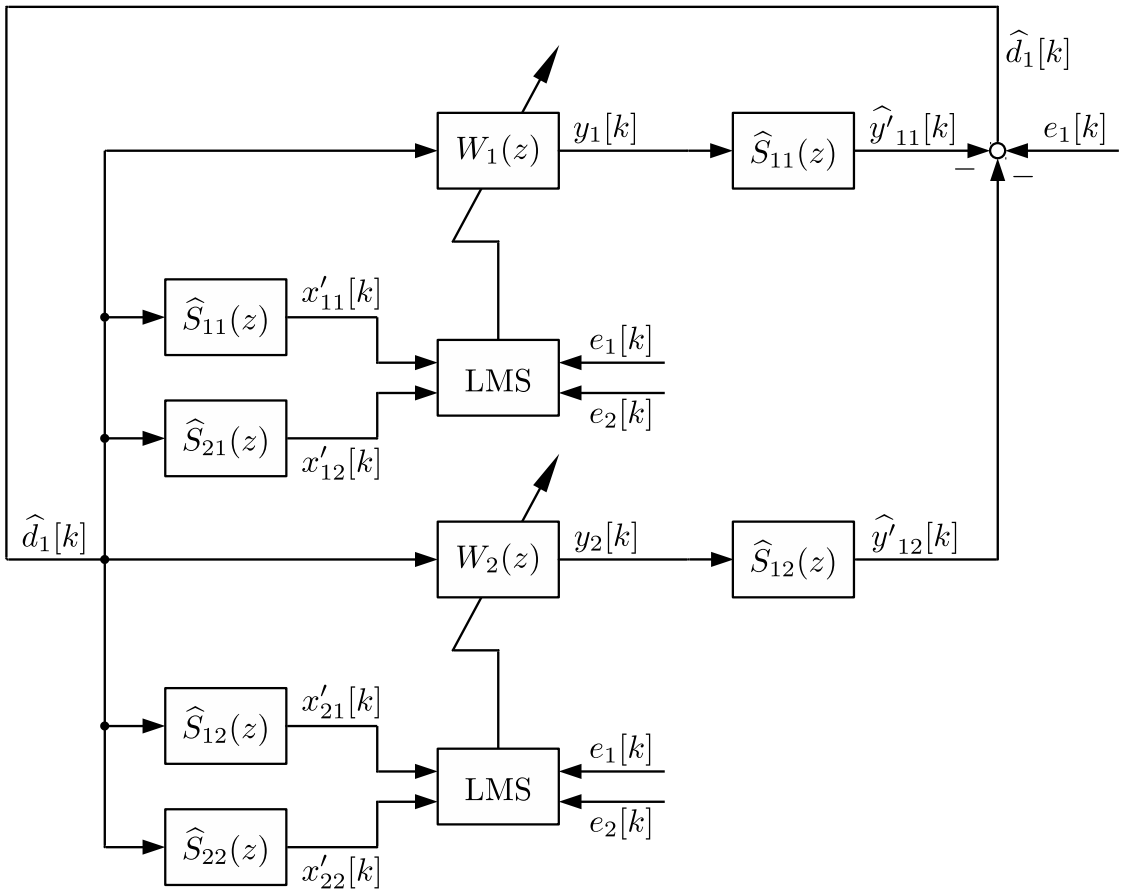

Abbildung 3.6: Blockdiagramm der adaptiven Regelung mit zwei Ein- und Ausgängen

4 Versuchsdurchführung

4.1 Vergleich zwischen einkanaliger Steuerung und Regelung

Im ersten Versuch werden eine einkanalige Steuerung und Regelung gegenüber gestellt. In Abbildung 4.1 ist die Versuchsanordnung gezeigt. Die Störquelle P befindet sich in einem Abstand von 275 mm zum Mittelpunkt $\boldsymbol{x}_0 = [\, x_0 \quad y_0 \,]^\mathsf{T}$ bei 0 ° des Messfeldes, die Gegenschallquelle S ebenfalls in einem Abstand von 275 mm , aber um einen Winkel von 135 ° versetzt. Das Fehlermikrofon M befindet sich im Mittelpunkt des Messfeldes. Durch den gestrichelt dargestellten Kreis ist das Messfeld angedeutet.

In Kapitel 3.1 sind die beiden verwendeten Algorithmen (einkanalige Steuerung und Regelung) beschrieben.

In Abbildung 4.2 und 4.3 ist die jeweilige Änderung des Schallfeldes durch Steuerung beziehungsweise Regelung des Schalldrucks, sprich die Differenz der Schalldruckpegel ΔL_p im Schallfeld ohne und mit ANC, und in 4.4 die um 10 dB beruhigten Zonen dargestellt. Es werden durch Steuerung und Regelung in etwa gleiche Ergebnisse erzielt. Zu erkennen ist, dass sich der beruhigte Bereich unter einem Winkel von etwa 67,5 ° ausbildet. Dies entspricht genau der Hälfte der Winkeldifferenz zwischen Primär- und Sekundärquelle.

Die ursprünglich erfassten Schallfelder, aus denen diese Ergebnisse gewonnen wurden, sind in Anhang A.1 gezeigt.

Um die Qualität der Steuerung/Regelung noch besser bewerten zu können, sind in Anhang B.1 zusätzlich Ausschnitte der Zeitverläufe und die Spektren des am Fehlermikrofon eingelesenen Signals dargestellt.

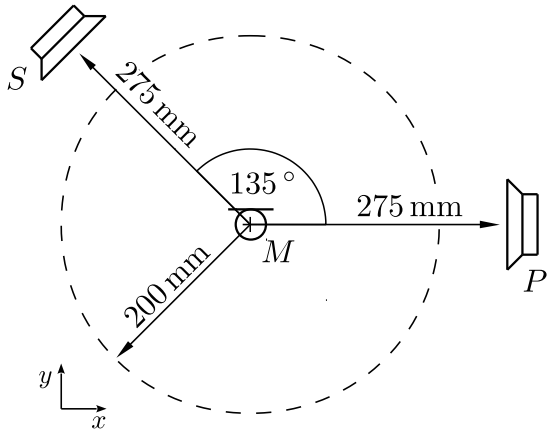

Abbildung 4.1: Versuchsanordnung beim Vergleich zwischen einkanaliger Steuerung und Regelung

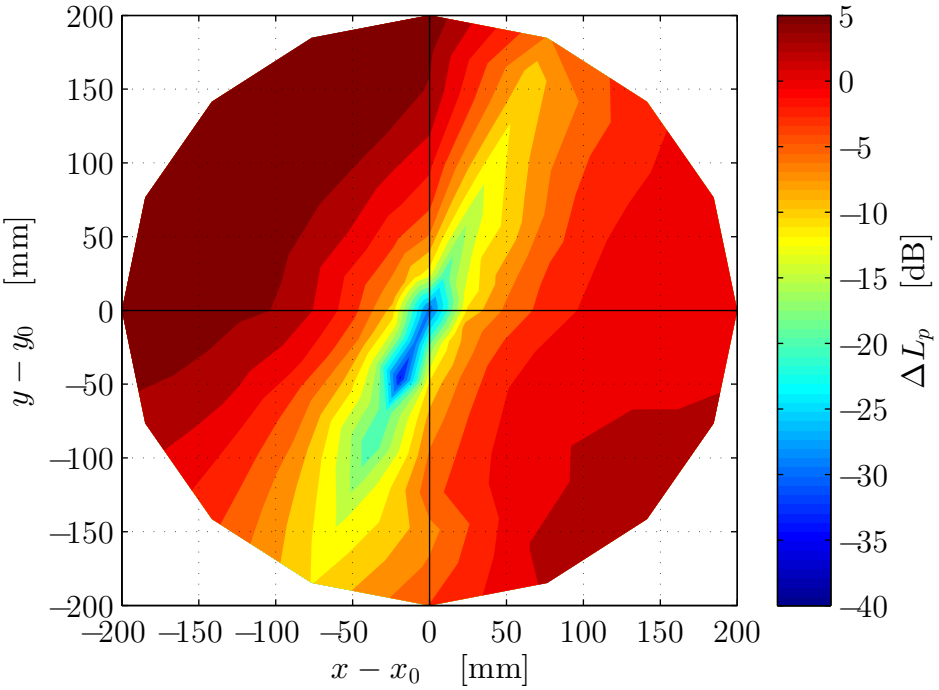

Abbildung 4.2: Änderung des Schalldruckpegels bei der einkanaliger Steuerung

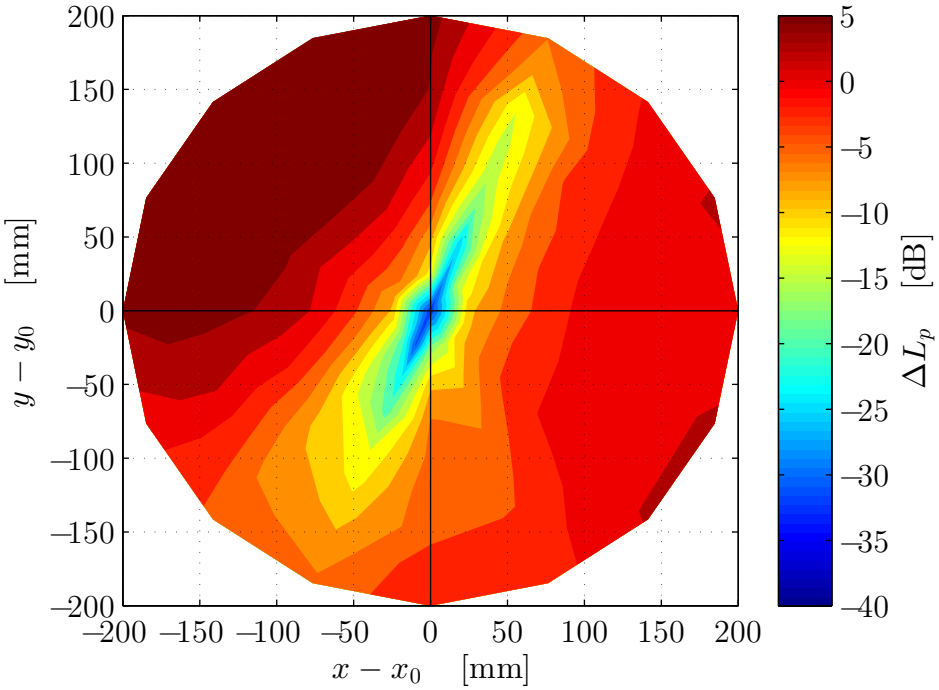

Abbildung 4.3: Änderung des Schalldruckpegels bei der einkanaliger Regelung

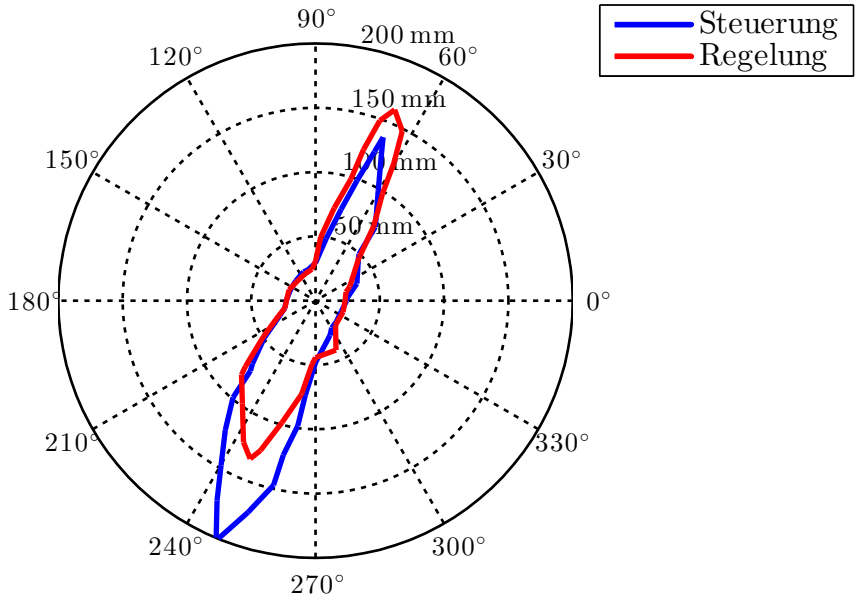

Abbildung 4.4: Vergleich der um 10 dB beruhigten Zonen zwischen einkanaliger Steuerung und Regelung

4.2 Vergleich zwischen Druck-Druck und Druck-Druckgradient Steuerung

Beim zweiten Versuch sind zwei unterschiedliche Steuerungen gegenübergestellt. Beide arbeiten mit einer Sekundärquelle S, welche sich in einem Abstand von 275 mm zum Messfeldmittelpunkt $\boldsymbol{x}_0$ befindet (siehe Abbildung 4.5). Die Störquelle P hat den selben Abstand zum Mittelpunkt. Beide Quellen sind um einen Winkel von 22,5 ° versetzt. Zur Signalerfassung dienen die beiden Mikrofone $M1$ und $M2$, welche einem Abstand von 50 mm aufweisen. Die Achse, welche durch beide Mikrofone geht, ist parallel zur x-Achse angeordnet. Verglichen wird die Druck-Druck Steuerung, deren Ziel es ist, die Schalldrücke an beiden Mikrofonen zu minimieren, mit der Druck-Druckgradient Steuerung. Bei dieser wird, wie bereits erläutert, der angenäherte Schalldruck und Schalldruckgradient in einer Raumrichtung mittig der beiden Mikrofone, also im Mittelpunkt $\boldsymbol{x}_0$, minimiert. Der Schalldruckgradient wird, vorgegeben durch die Anordnung der Mikrofone, in Richtung der x-Achse angenähert.

Der verwendete Algorithmus ist in Kapitel 3.2.2 beschrieben.

Zu erwähnen gilt es, dass es sich hier um ein überbestimmtes System handelt, da die Steuerung zwei allgemein unabhängige Eingangssignale aufweist, aber lediglich ein Ausgang zur Einflussnahme auf diese zur Verfügung steht. In der Praxis können deshalb im Allgemeinen nicht beide Forderungen nach Minimierung erfüllt werden und es stellt sich eine Lösung ein, welche die am besten zu erreichende Lösung, sprich einen Kompromiss darstellt.

In den Abbildungen 4.6 bis 4.8 sind die Änderungen der Schalldruckpegel ΔL_p im Schallfeld und die um 10 dB beruhigten Zonen dargestellt. Die Steuerungserfolge fallen ziemlich ähnlich aus. Was auffällt ist, dass sich bei der Druck-Druck-Steuerung keine so starke Reduktion im Mittelpunkt $\boldsymbol{x}_0$ wie bei der Druck-Druckgradient Steuerung einstellt. Dafür fällt der beruhigte Bereich etwas größer aus.

In Anhang A.2 sind die zugehörigen erfassten Schallfelder, in B.2 Ausschnitte der Zeitsignale und Spektren der gesteuerten Signale dargestellt.

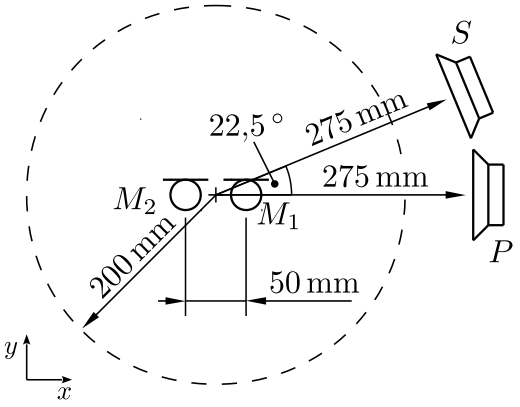

Abbildung 4.5: Versuchsanordnung beim Vergleich zwischen Druck-Druck und Druck-Druckgradient Steuerung

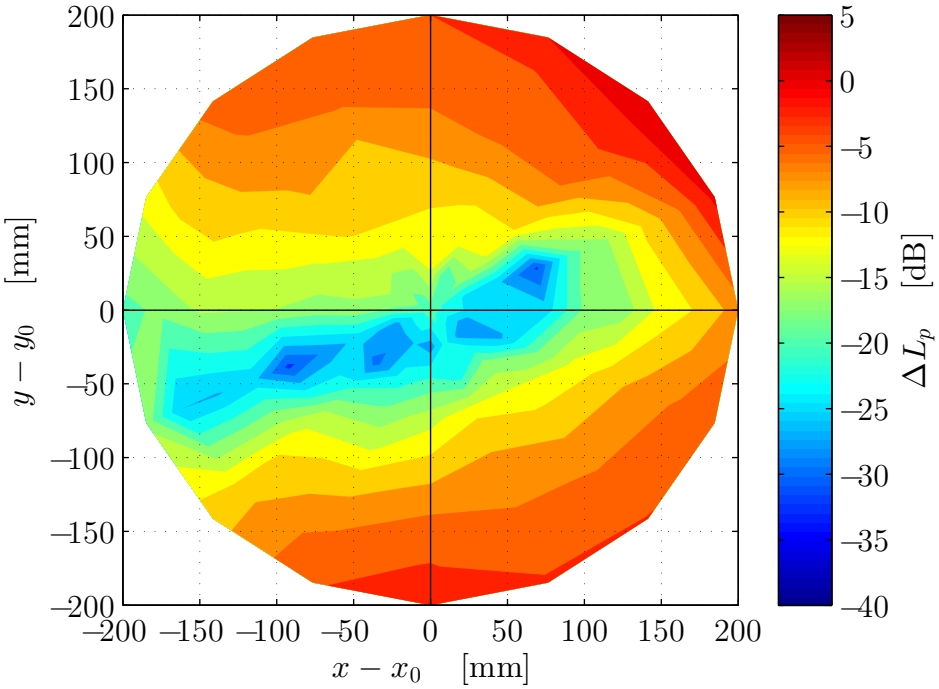

Abbildung 4.6: Änderung des Schalldruckpegels bei der Druck-Druck Steuerung

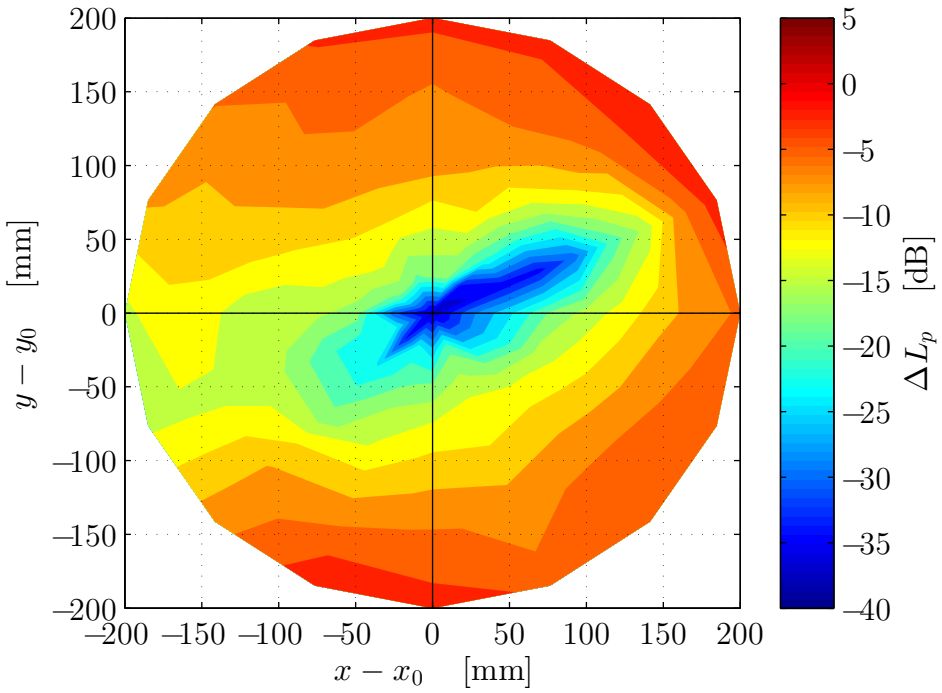

Abbildung 4.7: Änderung des Schalldruckpegels bei der Druck-Druckgradient Steuerung

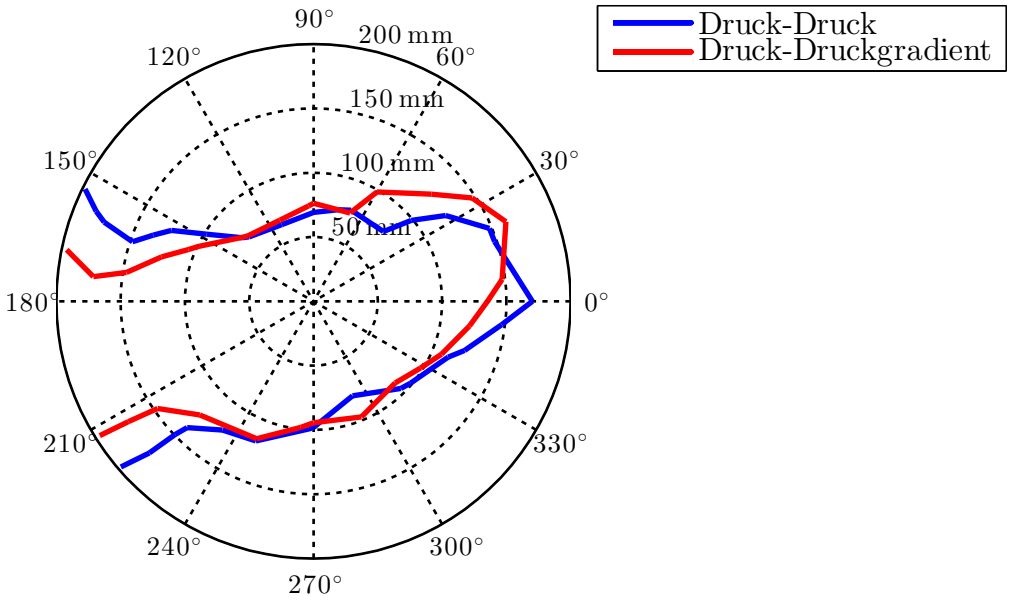

Abbildung 4.8: Vergleich der um 10 dB beruhigten Zonen bei der mehrkanaligen Steuerung

4.3 Vergleich zwischen Druck-Druck und Druck-Druckgradient Regelung

Im dritten Versuch sind zwei Regelungen mit je zwei Eingangssignalen gegenübergestellt. Die Störquelle P steht in einer Entfernung von 275 mm unter einem Winkel von 0 °(siehe Abbildung 4.9). Die Anordnung der Mikrofone ist gleich der bei der Druck-Druck beziehungsweise Druck-Druckgradient Steuerung. Im Unterschied zu dieser kommen bei der Regelung zwei Gegenschallquellen, S_1 und S_2 , zum Einsatz. Diese sind unter einem Winkel von 22,5 ° und 45 ° in einem Abstand von je 200 mm zum Messfeldmittelpunkt $\boldsymbol{x}_0$ angeordnet.

Der mehrkanalige Regelungsalgorithmus ist in Kapitel 3.2.3 beschrieben.

Vergleicht man die erhaltenen Ergebnisse der Änderung des Schalldruckpegels im Schallfeld, der durch die Regelung verursacht wird, und die um mindestens 10 dB beruhigten Bereiche (Abbildung 4.10 bis 4.12), so erkennt man, dass die Druck-Druck Regelung tendenziell einen größeren Bereich beruhigt. Auch zu sehen ist, dass sich der beruhigte Bereich bei der Druck-Druckgradient Regelung entlang der x-Achse etwas weiter erstreckt. Dies ist interessant, da entlang dieser Achse der angenäherte Druckgradient aktiv beeinflusst wird. Dies deckt sich mit der theoretischen Überlegung, den Schalldruck und gleichzeitig die Änderung des Schalldruckes entlang dieser Achse zu minimieren und so einen größeren beruhigten Bereich in dieser Richtung zu erhalten. Dies geht allerdings mit einer nicht so starken Absenkung des Schalldruckpegels um den Mittelpunkt $\boldsymbol{x}_0$ einher.

In Anhang A.3 sind die gemessenen Schallfelder ohne und mit ANC-Regelung abgebildet. Anhang B.3 zeigt einen Ausschnitt des Zeitbereichs der jeweils geregelten Signale und deren Frequenzspektren. Hier ist noch ein Nachteil der Regelung zu erkennen. Zwar werden die Anteile der geregelten Frequenz deutlich in der Amplitude gesenkt, dafür ist eine Erhöhung, vor allem bei höherharmonischen der geregelten Frequenz, zu erkennen.

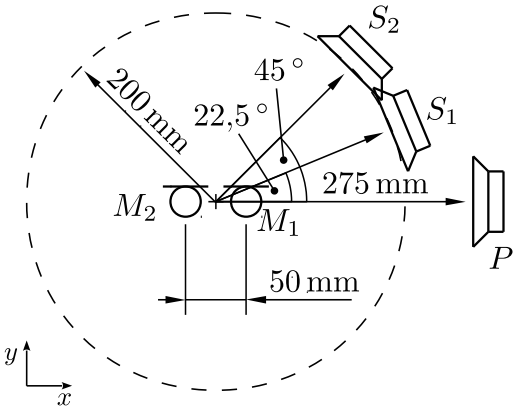

Abbildung 4.9: Versuchsanordnung beim Vergleich zwischen Druck-Druck und Druck-Druckgradient Regelung

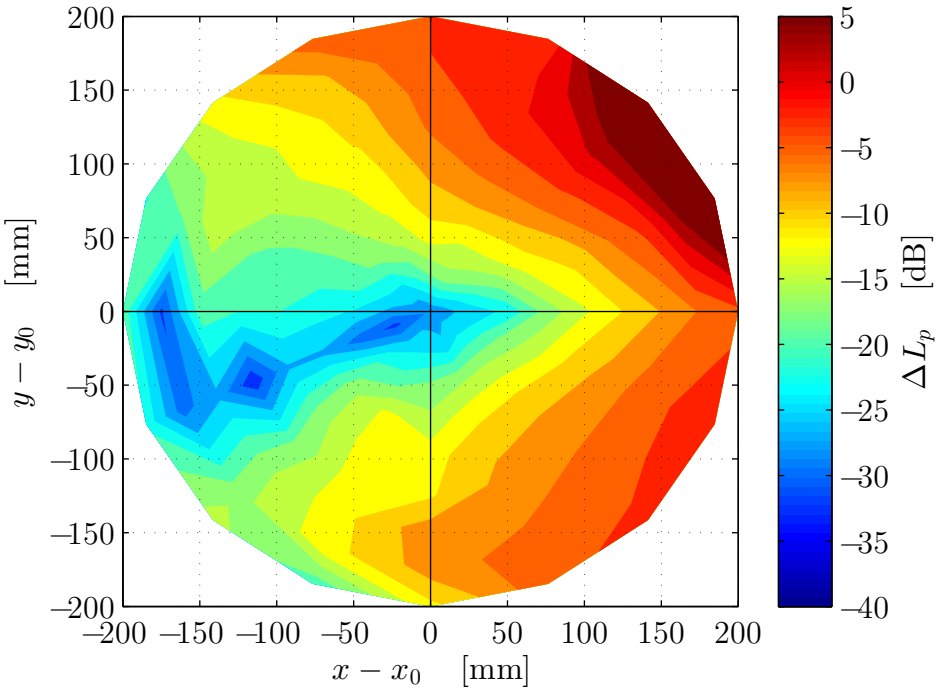

Abbildung 4.10: Änderung des Schalldruckpegels bei der Druck-Druck Regelung

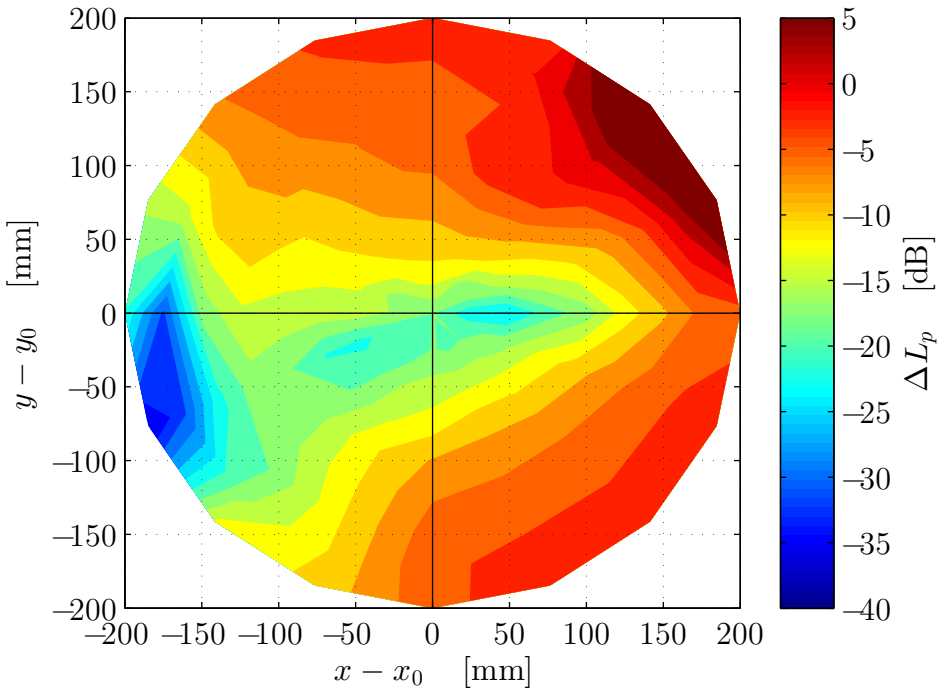

Abbildung 4.11: Änderung des Schalldruckpegels bei der Druck-Druckgradient Regelung

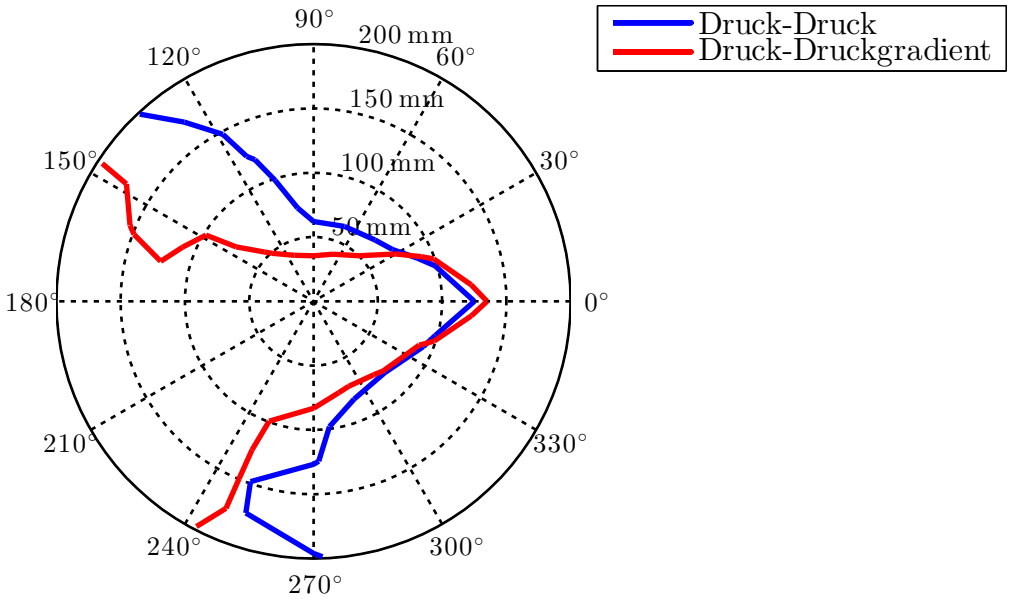

Abbildung 4.12: Vergleich der um 10 dB beruhigten Zonen bei der mehrkanaligen Regelung

5 Zusammenfassung

Im Zuge dieser Arbeit wurden alle für diese relevanten Grundlagen des ANC zusammengefasst angeführt. Es wurden insgesamt vier Algorithmen implementiert und eine Umgebung geschaffen, um diese zu testen und zu bewerten. Dabei wurde die einkanalige Steuerung der einkanaligen Regelung, die Druck-Druck der Druck-Druckgradient Steuerung mit einer Gegenschallquelle und die Druck-Druck der Druck-Druckgradient Regelung mit je zwei Gegenschallquellen gegenübergestellt.

Insbesondere bei den mehrkanaligen Systemen sind Unterschiede zu erkennen. Es wurde gezeigt, dass es prinzipiell möglich ist, neben dem Druck auch den Druckgradienten aktiv zu beeinflussen, um einen im Schalldruckpegel gesenkten Bereich zu erzeugen. Um einen umfangreicheren Überblick über die Wirksamkeit einer Druck-Druckgradient ANC Steuerung beziehungsweise Regelung zu erhalten, könnten die Steuerungs-/Regelungserfolge für zum Beispiel verschiedene Anordnungen der Primär-, Sekundärquellen und Fehlermikrofone untersucht werden, was aufbauend auf den Ergebnissen dieser Arbeit vorstellbar ist. Weiter wäre die direkte Erfassung der Schallschnelle mit Hilfe eines Hitzedrahtmikrofons denkbar, wodurch eventuell noch genauer auf den Schalldruckgradienten geschlossen werden könnte.

In der Arbeit wurde lediglich die aktive Beeinflussung des Schalldruckgradienten in einer Raumrichtung betrachtet. Durch die Verwendung von drei beziehungsweise vier Mikrofonen, ist es denkbar, den Schalldruckgradienten in zwei beziehungsweise allen drei Raumrichtungen anzunähern und zu beeinflussen, so dass sich theoretisch eine weitere Vergrößerung des beruhigten Bereichs erzielen ließe.

Da die nötigen Algorithmen schnell sehr umfangreich werden, ist es durchaus angebracht, Überlegungen in Hinblick auf eine Vergrößerung der Rechenleistung anzustellen. Eine Möglichkeit, welche man in Betracht ziehen sollte, ist der Einsatz von FPGAs (Field Programmable Gate Array). Hier werden zunehmend Modelle entwickelt, welche speziell für den Einsatz in der Signalverarbeitung gedacht sind und entsprechend optimierte Hardware-Blöcke zur Verfügung stellen. Der Vorteil von FPGAs liegt darin, dass sie die parallele Implementation verschiedener Operationen erlauben, was insbesondere im Hinblick auf mehrkanalige Systeme interessant ist. Von Nachteil bei der Umsetzung solcher Algorithmen auf einem FPGA ist, dass diese im Vergleich zu einem DSP deutlich anspruchsvoller ist.

Literaturverzeichnis

[1] Chassaing, Rulph: *Digital Signal Processing and Applications with the C6713 and C6416 DSK*. Hoboken, NJ, USA : John Wiley & Sons, 2005. – ISBN 978–0–471–70406–5

[2] Elliott, S. J. ; Garcia-Bonito, J.: Active cancellation of pressure and pressure gradient in a diffuse sound field. In: *Journal of Sound and Vibration* 186 (1995), Nr. 4, S. 696–704

[3] Frey, Thomas ; Bossert, Martin: *Signal- und Systemtheorie*. 2. Aufl. Wiesbaden, DE : Vieweg+Teubner, 2008. – ISBN 978–3–8351–0249–1

[4] Hansen, H. C.: *Understanding Active Noise Cancellation*. New York, NY, USA : Spon Press, 2001. – ISBN 978–0–203–46733–7

[5] Henn, Hermann ; Sinambari, Gholam R. ; Fallen, Manfred: *Ingenieurakustik : Physikalische Grundlagen und Anwendungsbeispiele*. 4. Aufl. Berlin, DE : Springer, 2008. – ISBN 978–3–8348–0255–2

[6] Kuo, Sen M. ; Morgan, Dennis R.: *Active Noise Control Systems : Algorithms and DSP Implementations*. New York, NY, USA : John Wiley & Sons, 1996. – ISBN 978–0–471–13424–4

[7] Lerch, Reinhard ; Sessler, Gerhard M. ; Wolf, Dietrich: *Technische Akustik : Grundlagen und Anwendungen*. Berlin, DE : Springer, 2009. – ISBN 978–3–540–23430–2

[8] Moschytz, George S. ; Hofbauer, Markus: *Adaptive Filter : Eine Einführung in die Theorie mit Aufgaben und MATLAB-Simulationen auf CD-ROM*. Berlin, DE : Springer, 2000. – ISBN 978–3–540–67651–1

[9] Möser, Michael: *Technische Akustik*. 9. Aufl. Berlin, DE : Springer, 2012. – ISBN 978–3–642–30933–5

[10] Spectrum Digital, Inc.: *TMS320C6713 DSK Technical Reference*, 2003

[11] Wang, Fenglin ; Mechefske, Chris K.: Active Feedback Control Using Adaptive Filtering. In: *Journal of Vibration and Control* 10 (2004), Nr. 1, S. 25–38

Anhang

A Gemessene Schallfelder ohne und mit ANC

A.1 Schallfelder beim Vergleich zwischen einkanaliger Steuerung und Regelung

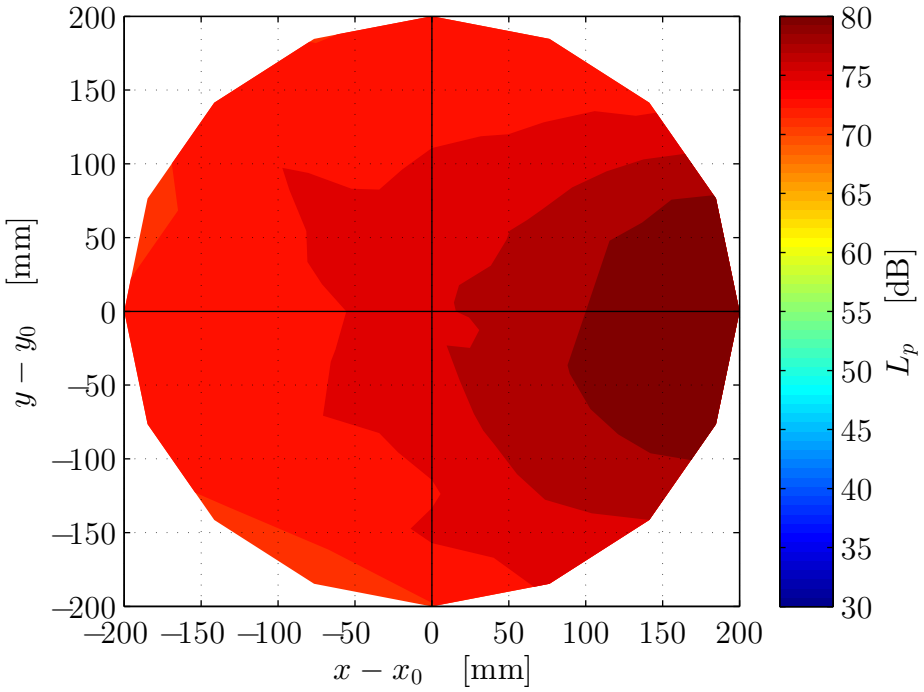

Abbildung A.1: Gemessenes Schallfeld ohne ANC

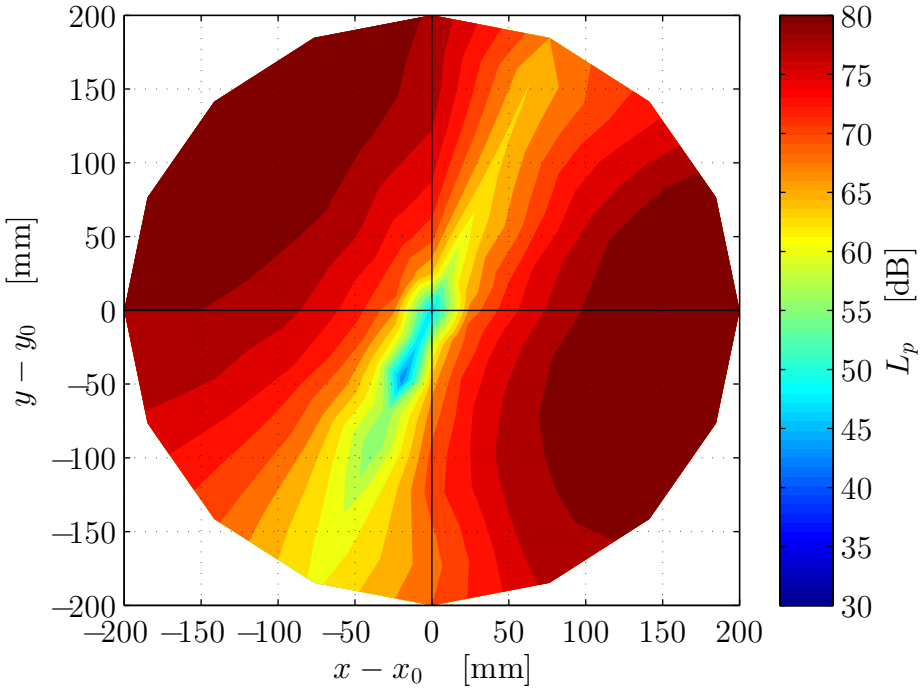

Abbildung A.2: Gemessenes Schallfeld bei einkanaliger Steuerung

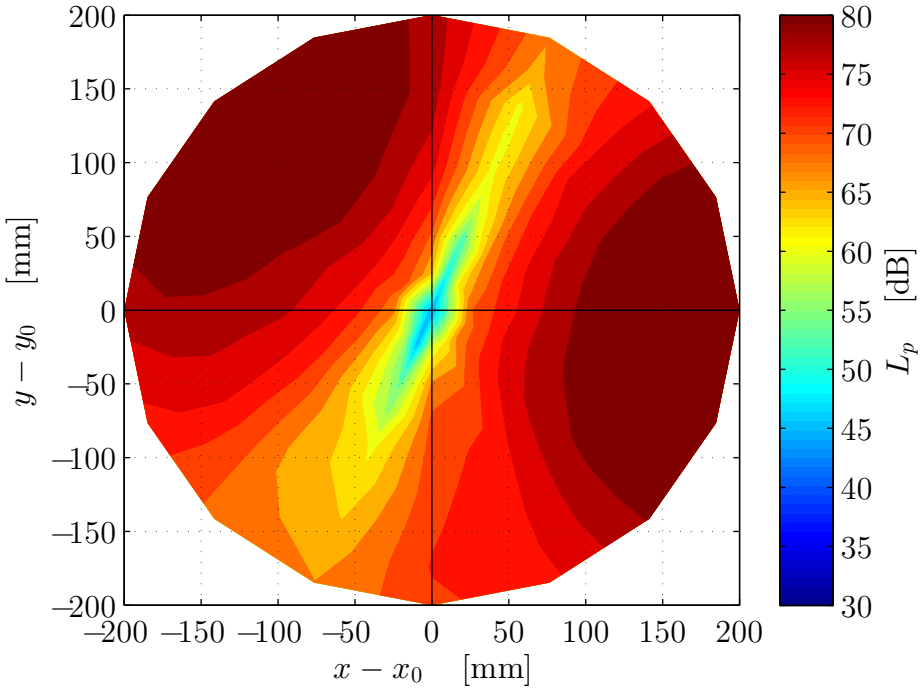

Abbildung A.3: Gemessenes Schallfeld bei einkanaliger Regelung

A.2 Schallfelder beim Vergleich zwischen Druck-Druck und Druck-Druckgradient Steuerung

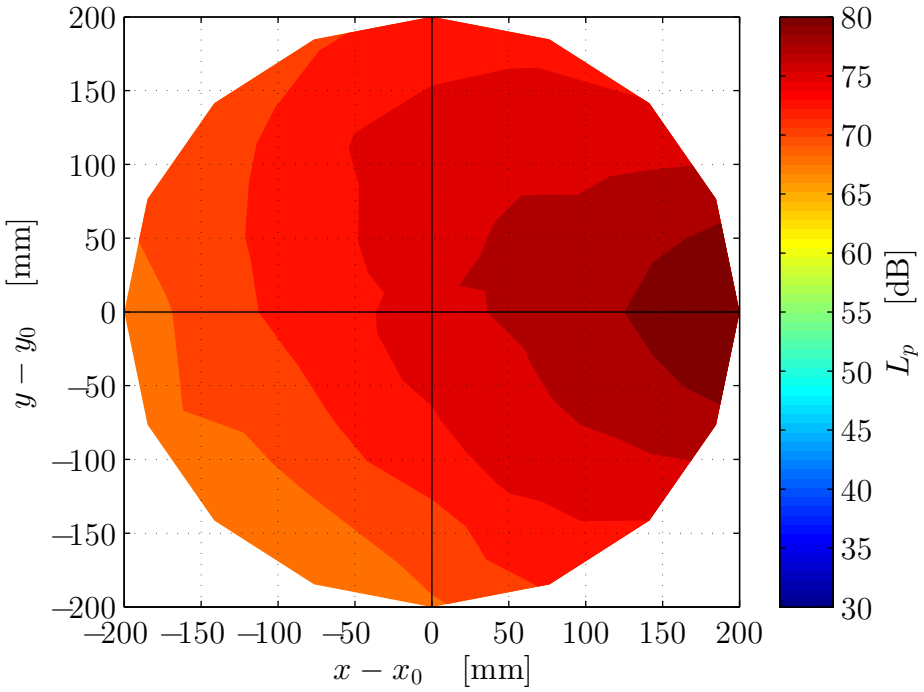

Abbildung A.4: Gemessenes Schallfeld ohne ANC

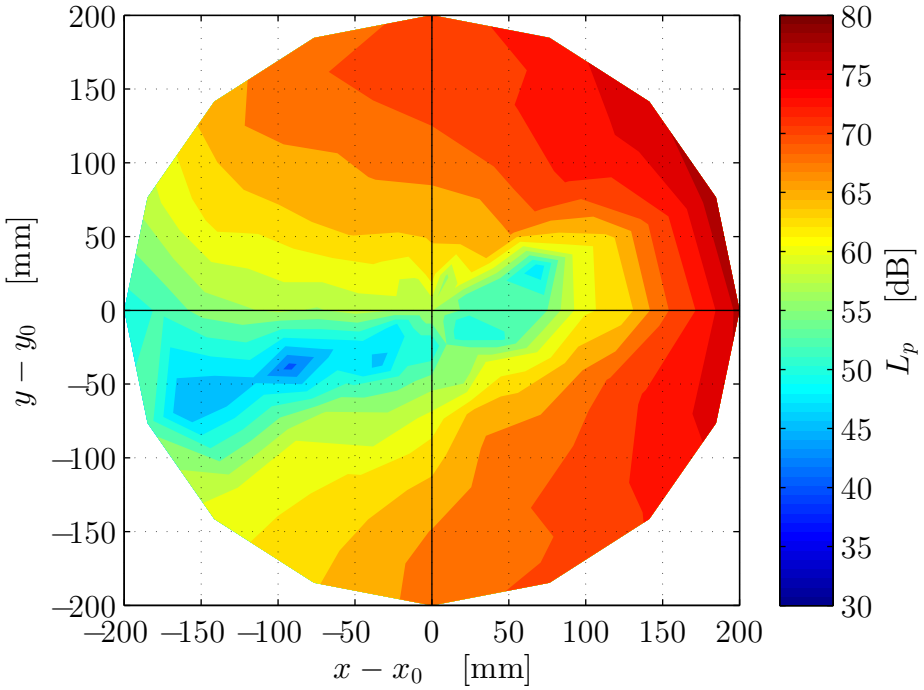

Abbildung A.5: Gemessenes Schallfeld bei Druck-Druck Steuerung

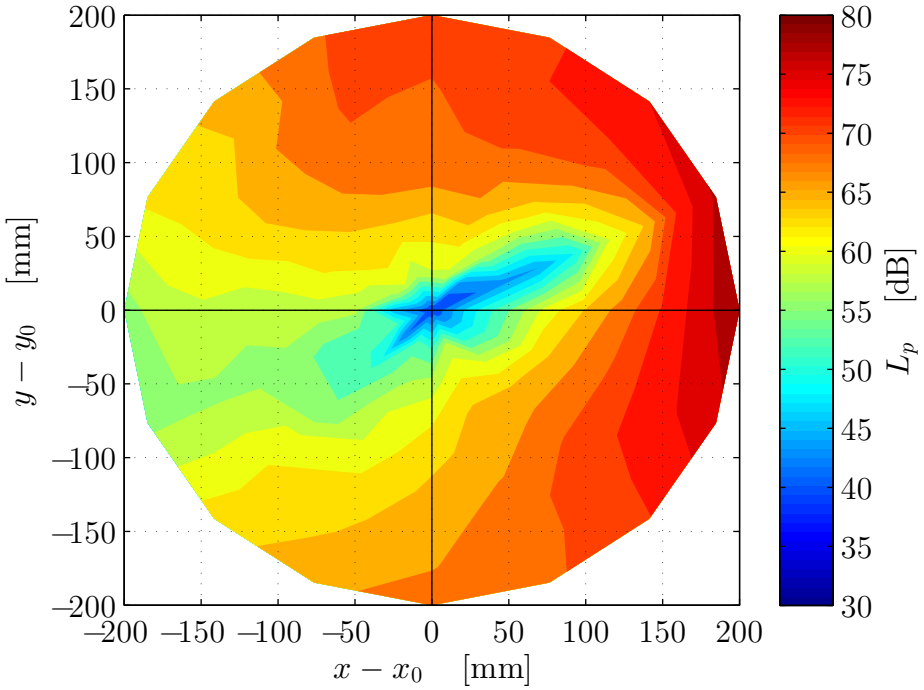

Abbildung A.6: Gemessenes Schallfeld bei Druck-Druckgradient Steuerung

A.3 Schallfelder beim Vergleich zwischen Druck-Druck und Druck-Druckgradient Regelung

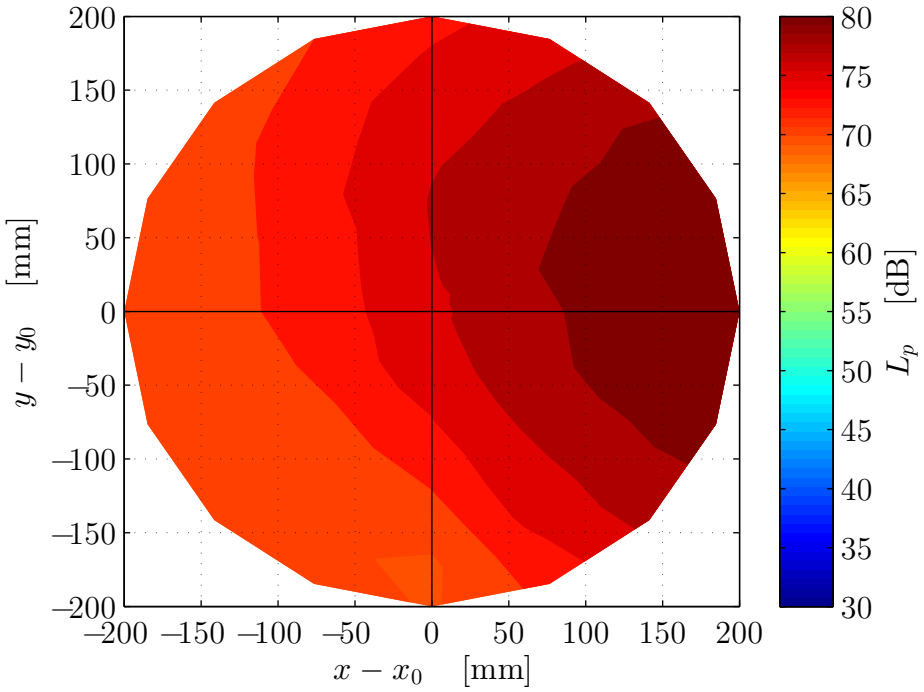

Abbildung A.7: Gemessenes Schallfeld ohne ANC

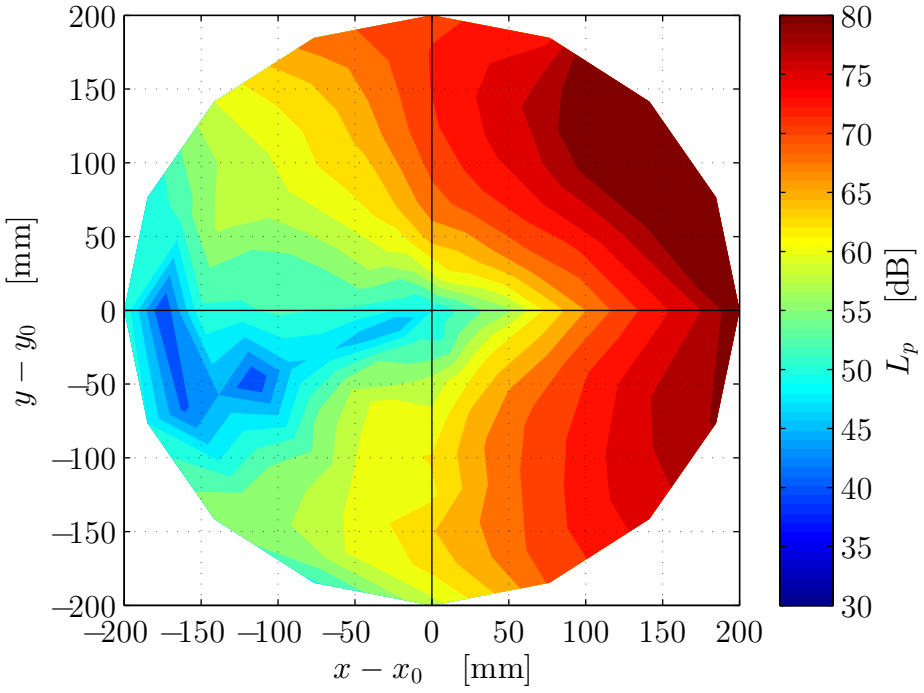

Abbildung A.8: Gemessenes Schallfeld bei Druck-Druck Regelung

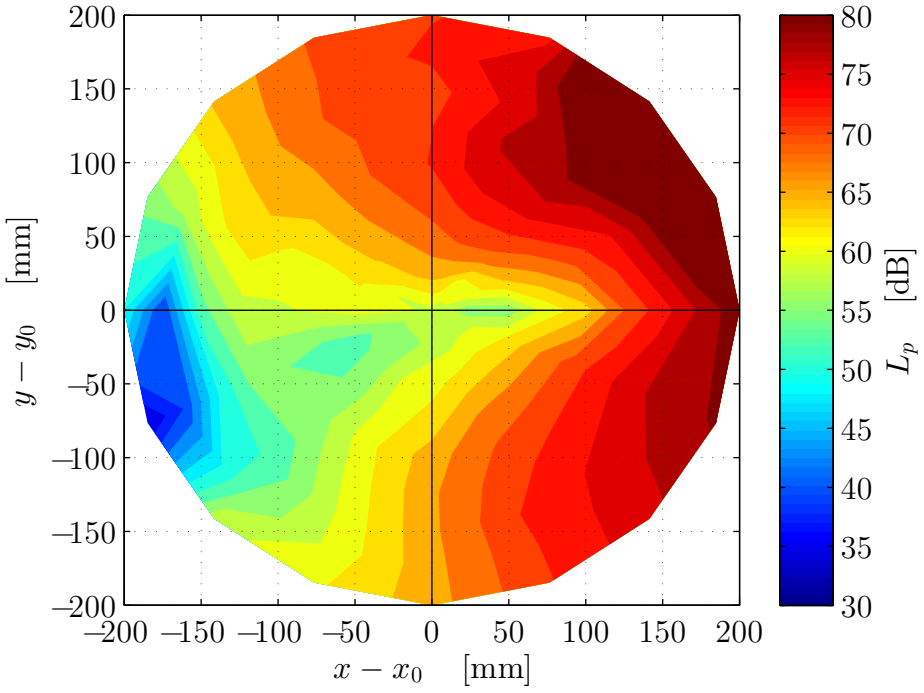

Abbildung A.9: Gemessenes Schallfeld bei Druck-Druckgradient Regelung

B Zeitverläufe und Frequenzspektren ohne und mit ANC

B.1 Zeitverläufe und Frequenzspektren beim Vergleich zwischen einkanaliger Steuerung und Regelung

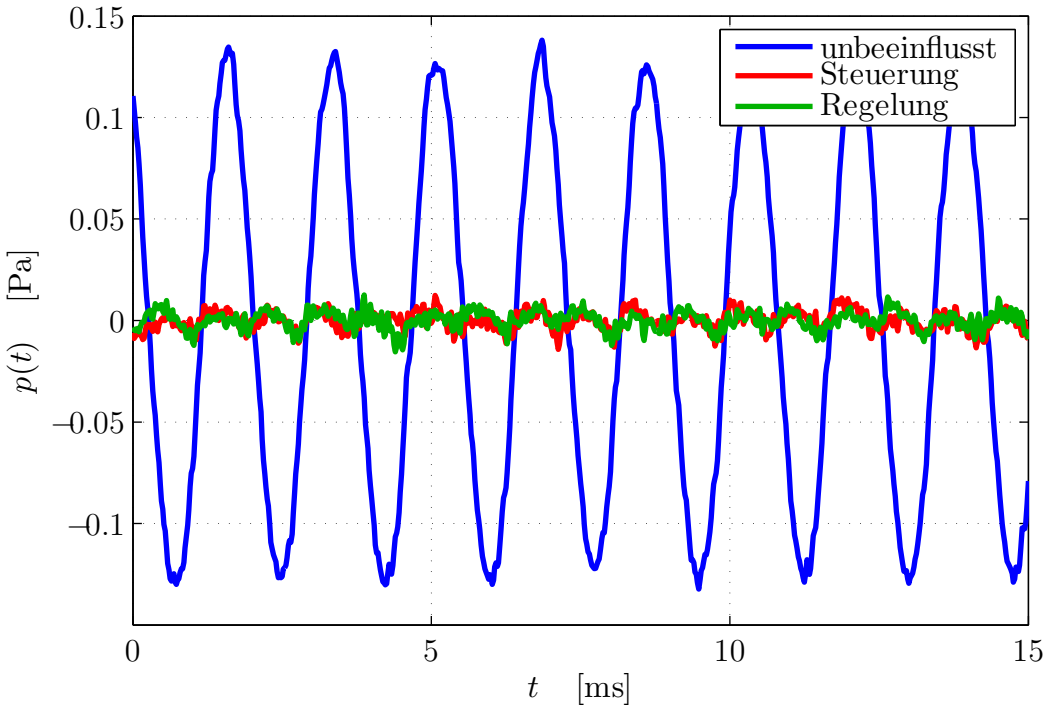

Abbildung B.1: Ausschnitt des Zeitverlaufs des Schalldrucks am Fehlermikrofon

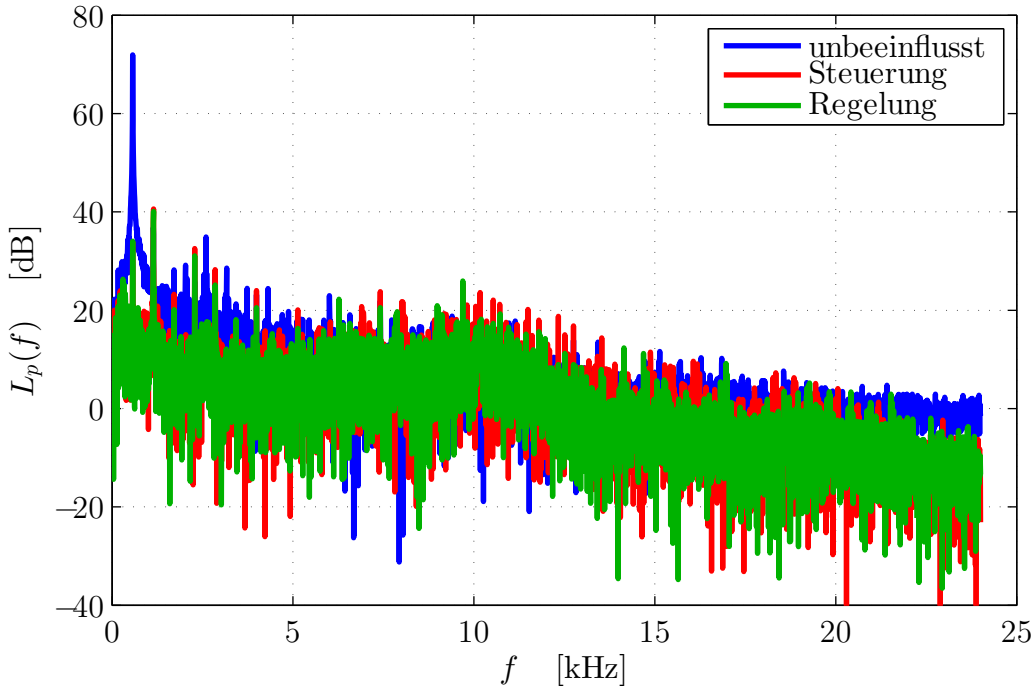

Abbildung B.2: Frequenzspektrum des Schalldruckpegels am Fehlermikrofon

B.2 Zeitverläufe und Frequenzspektren beim Vergleich zwischen Druck-Druck und Druck-Druckgradient Steuerung

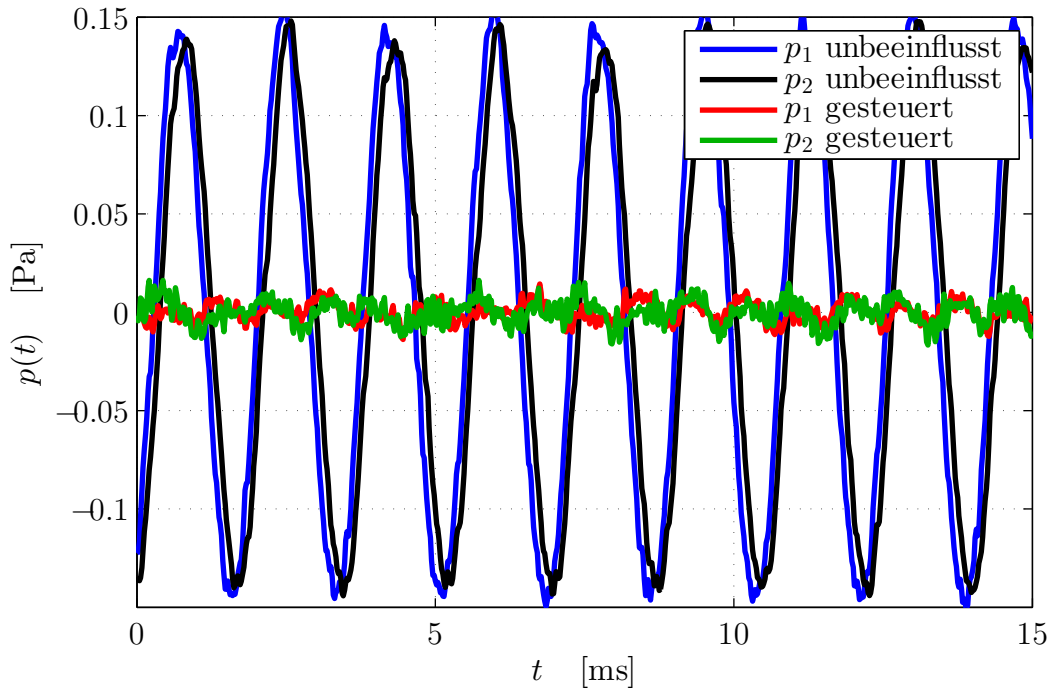

Abbildung B.3: Ausschnitt des Zeitverlaufs der Schalldrücke am Fehlermikrofon bei der Druck-Druck Steuerung

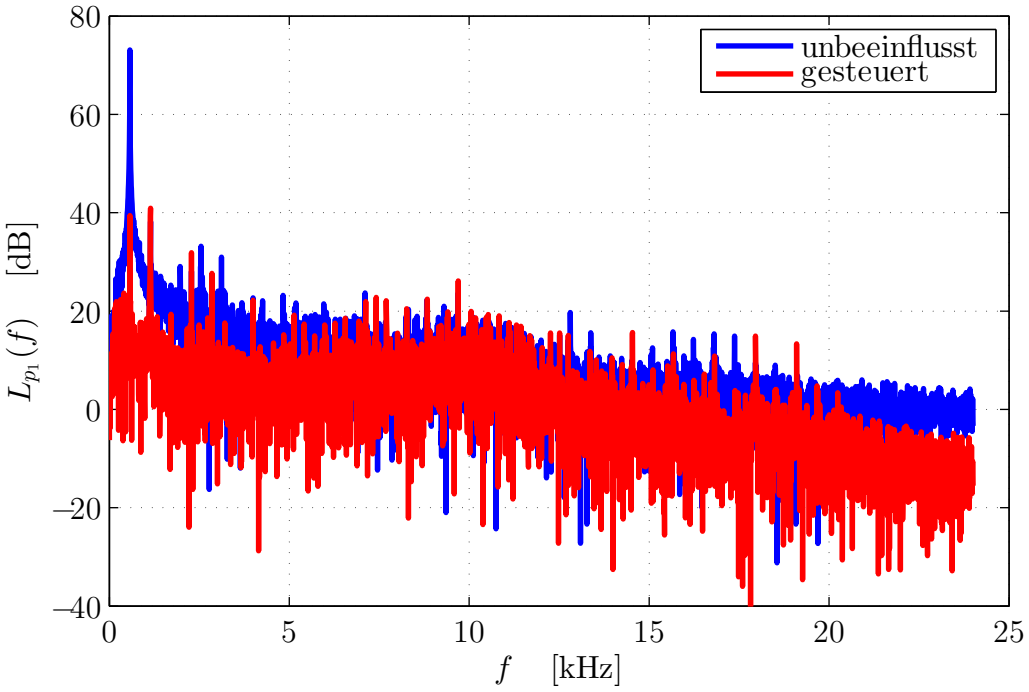

Abbildung B.4: Frequenzspektrum des Schalldruckpegels am ersten Fehlermikrofon bei der Druck-Druck Steuerung

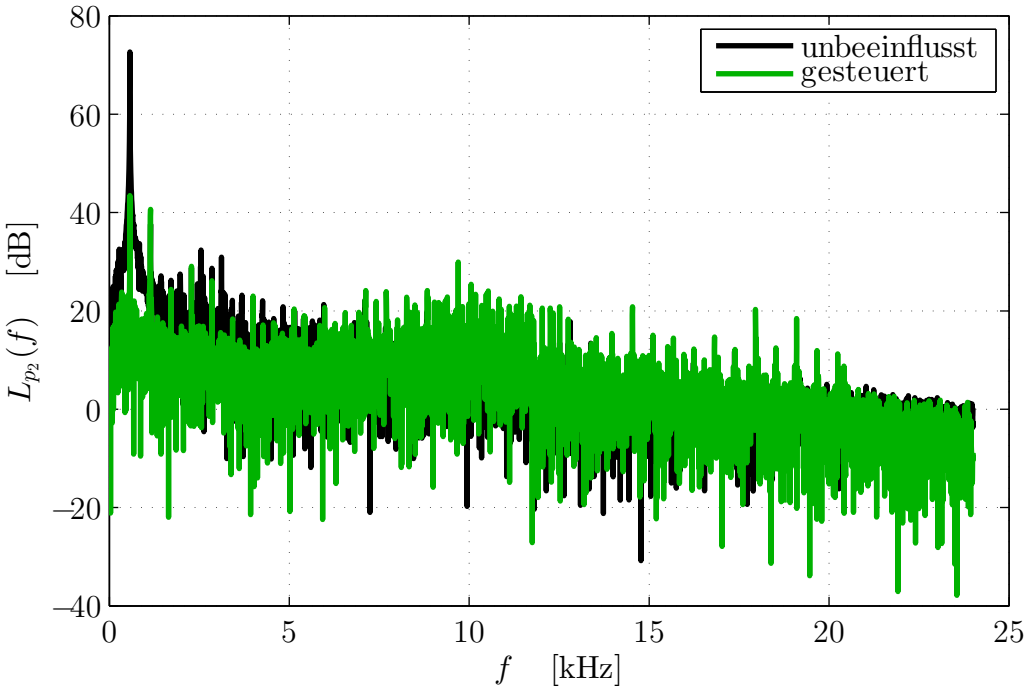

Abbildung B.5: Frequenzspektrum des Schalldruckpegels am zweiten Fehlermikrofon bei der Druck-Druck Steuerung

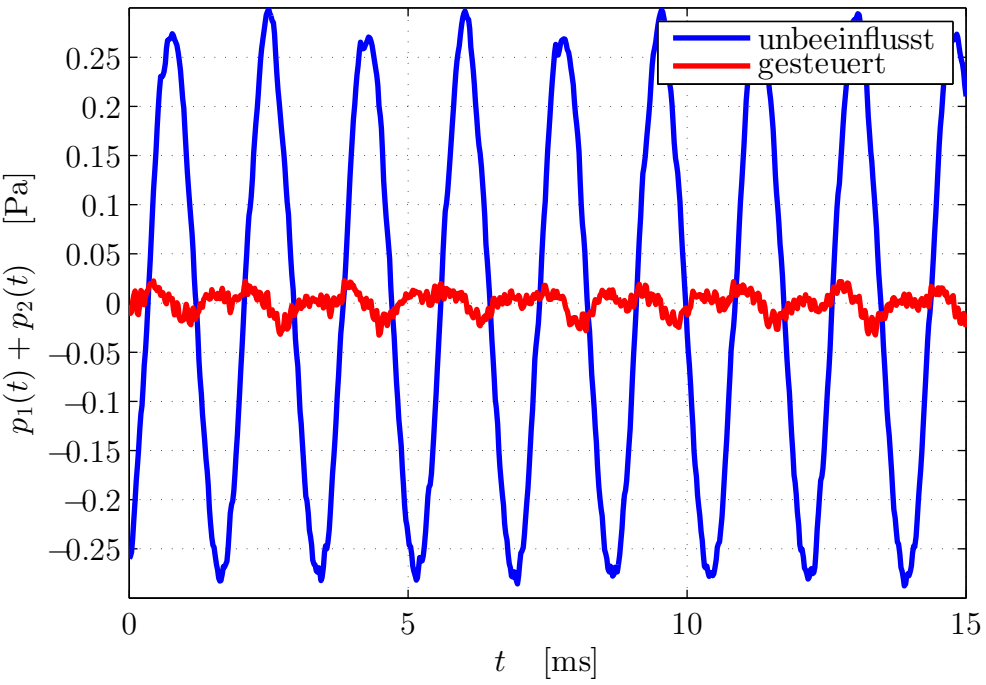

Abbildung B.6: Ausschnitt des Zeitverlaufs der addierten Schalldrücke bei der Druck-Druckgradient Steuerung

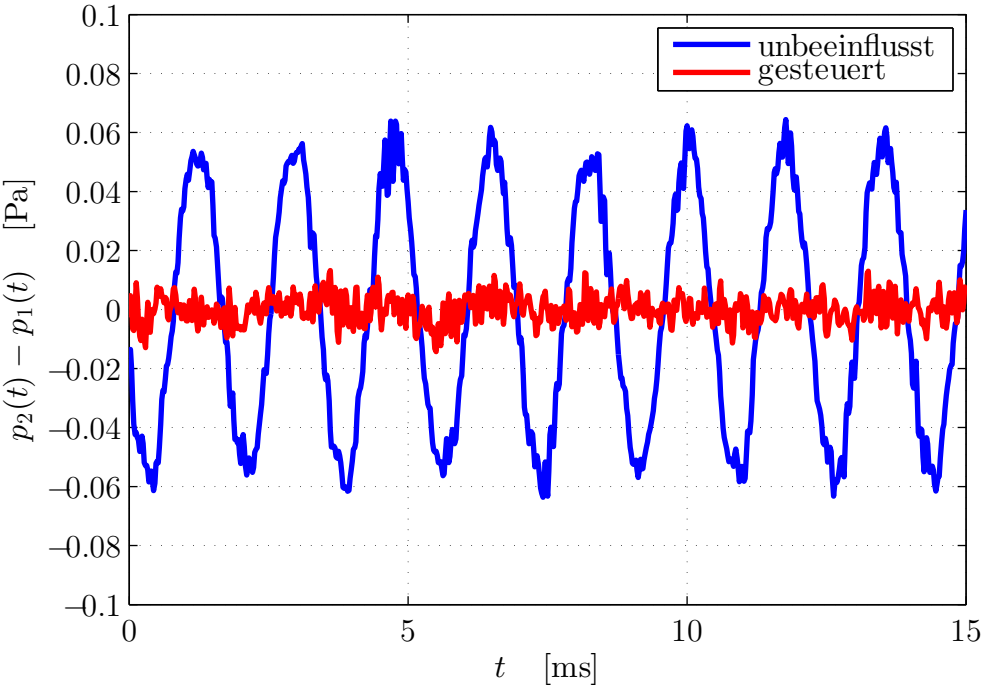

Abbildung B.7: Ausschnitt des Zeitverlaufs der subtrahierten Schalldrücke bei der Druck-Druckgradient Steuerung

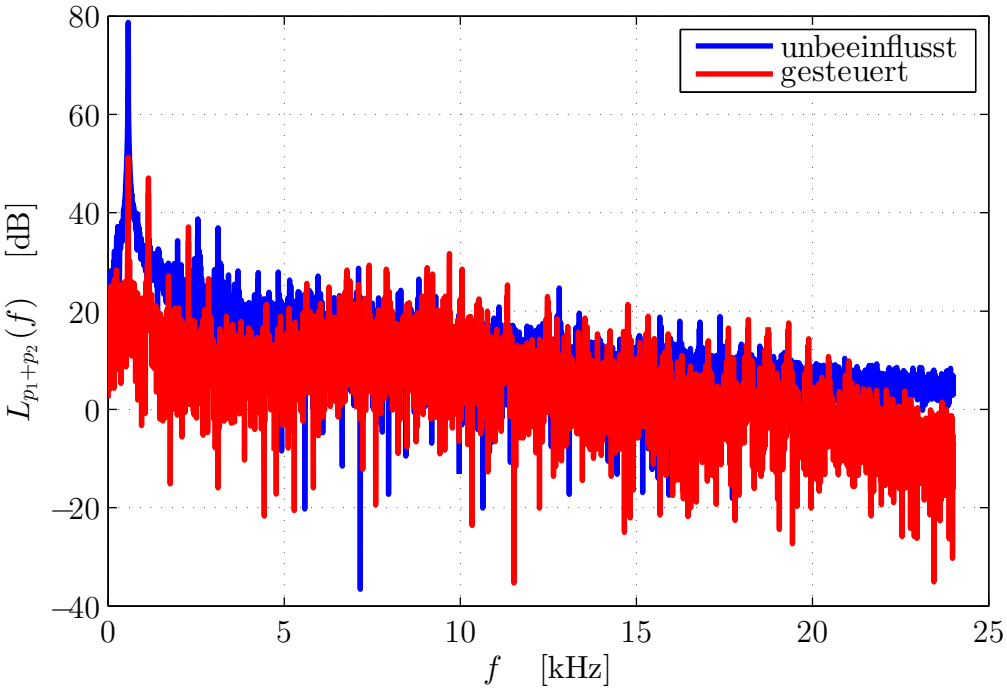

Abbildung B.8: Frequenzspektrum des Schalldruckpegels der addierten Schalldrücke bei der Druck-Druckgradient Steuerung

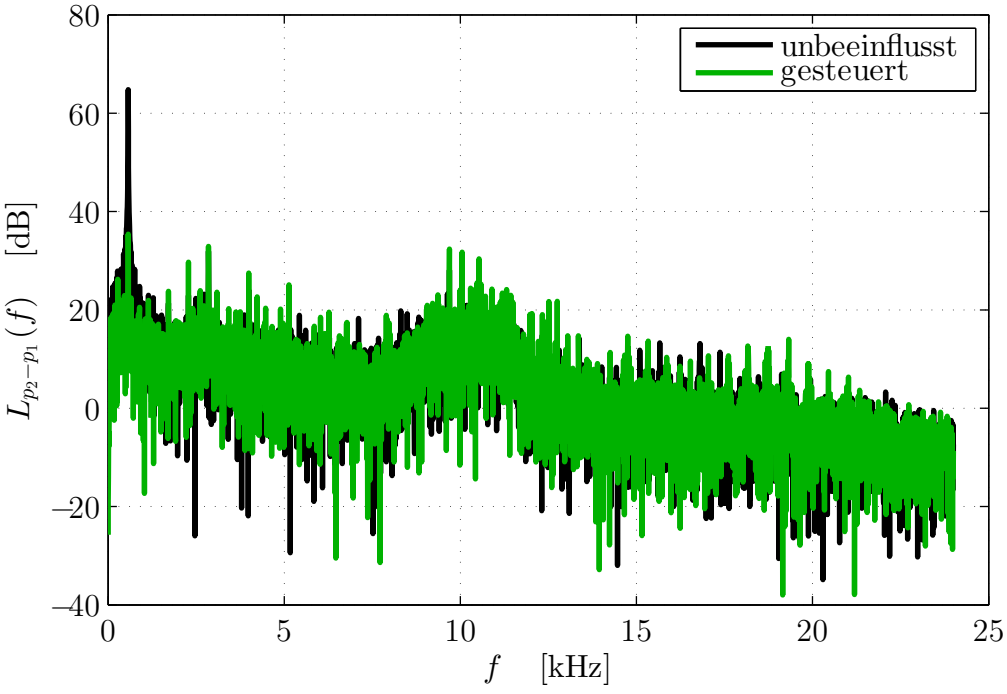

Abbildung B.9: Frequenzspektrum des Schalldruckpegels der subtrahierten Schalldrücke bei der Druck-Druckgradient Steuerung

B.3 Zeitverläufe und Frequenzspektren beim Vergleich zwischen Druck-Druck und Druck-Druckgradient Regelung

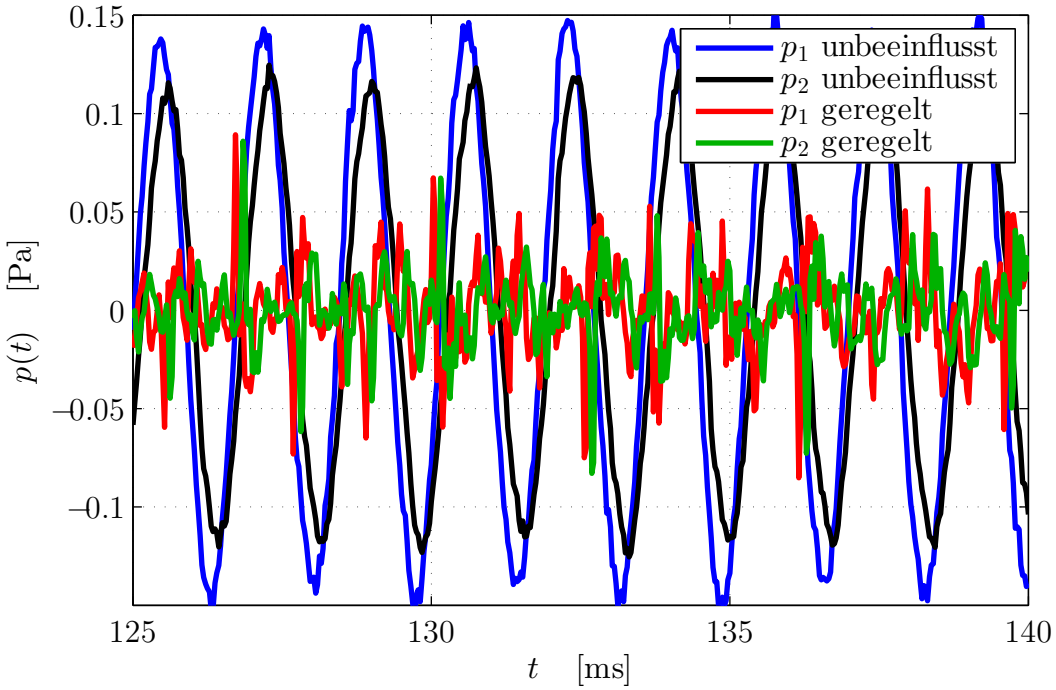

Abbildung B.10: Ausschnitt des Zeitverlaufs der Schalldrücke am Fehlermikrofon bei der Druck-Druck Regelung

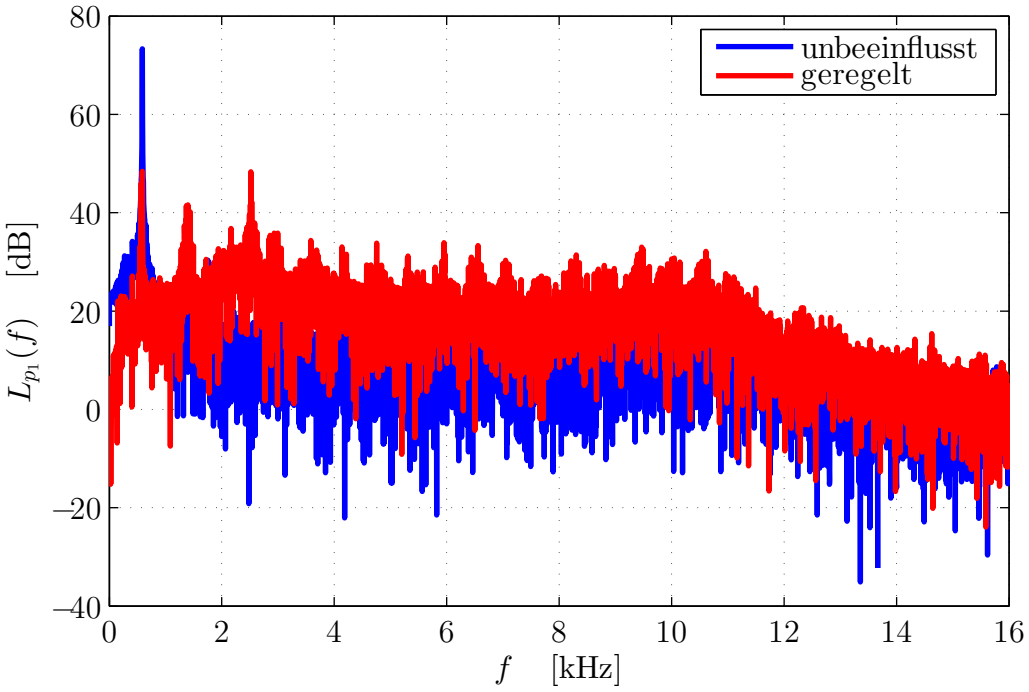

Abbildung B.11: Frequenzspektrum des Schalldruckpegels am ersten Fehlermikrofon bei der Druck-Druck Regelung

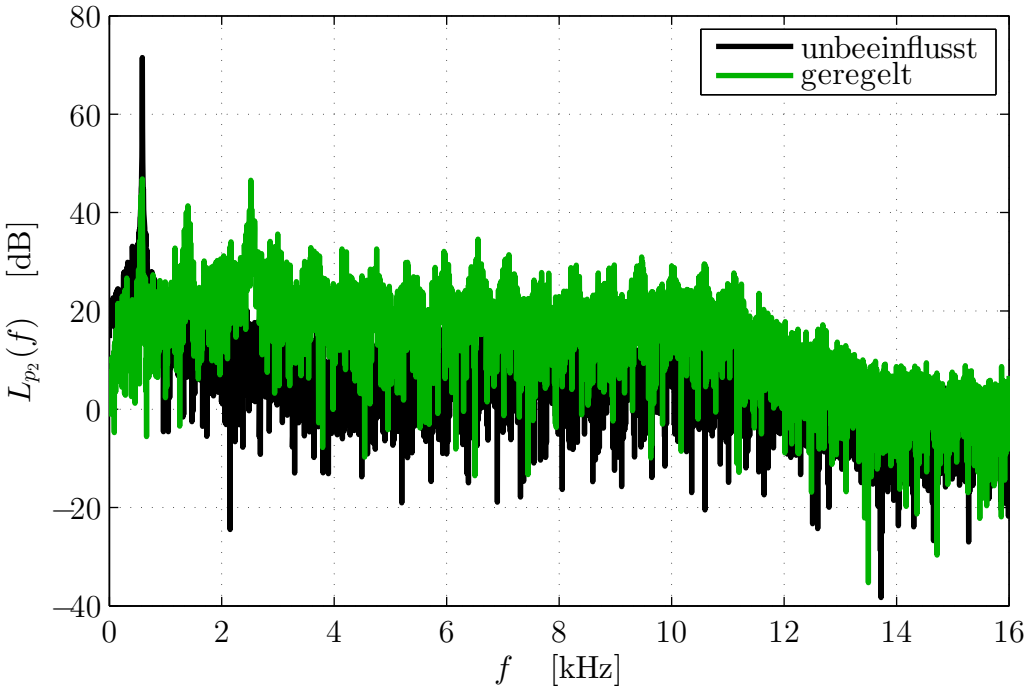

Abbildung B.12: Frequenzspektrum des Schalldruckpegels am zweiten Fehlermikrofon bei der Druck-Druck Regelung

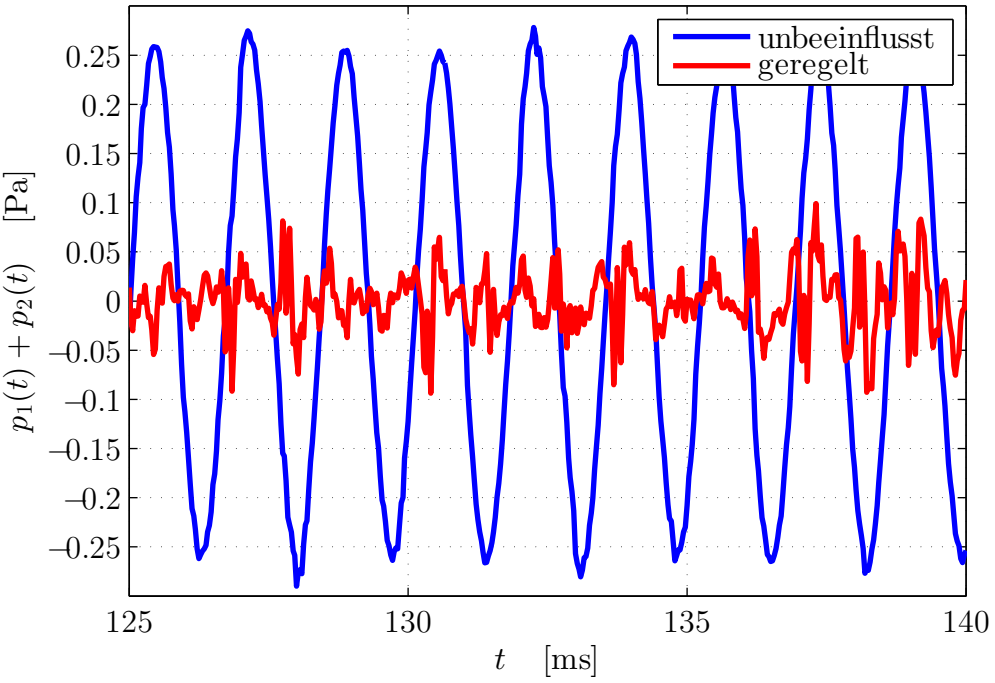

Abbildung B.13: Ausschnitt des Zeitverlaufs der addierten Schalldrücke bei der Druck-Druckgradient Regelung

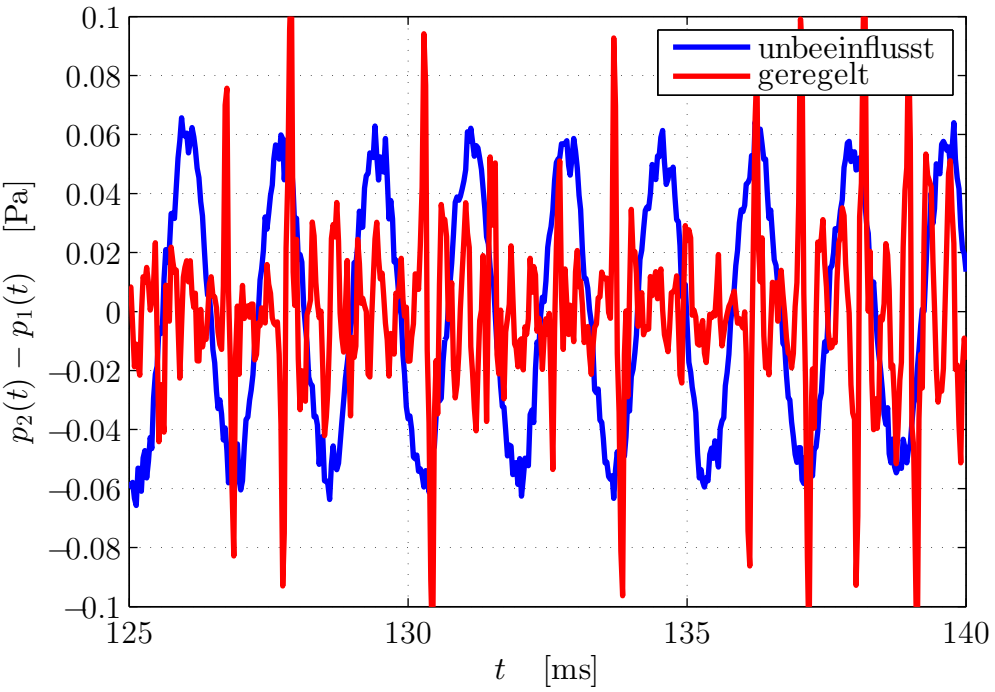

Abbildung B.14: Ausschnitt des Zeitverlaufs der subtrahierten Schalldrücke bei der Druck-Druckgradient Regelung

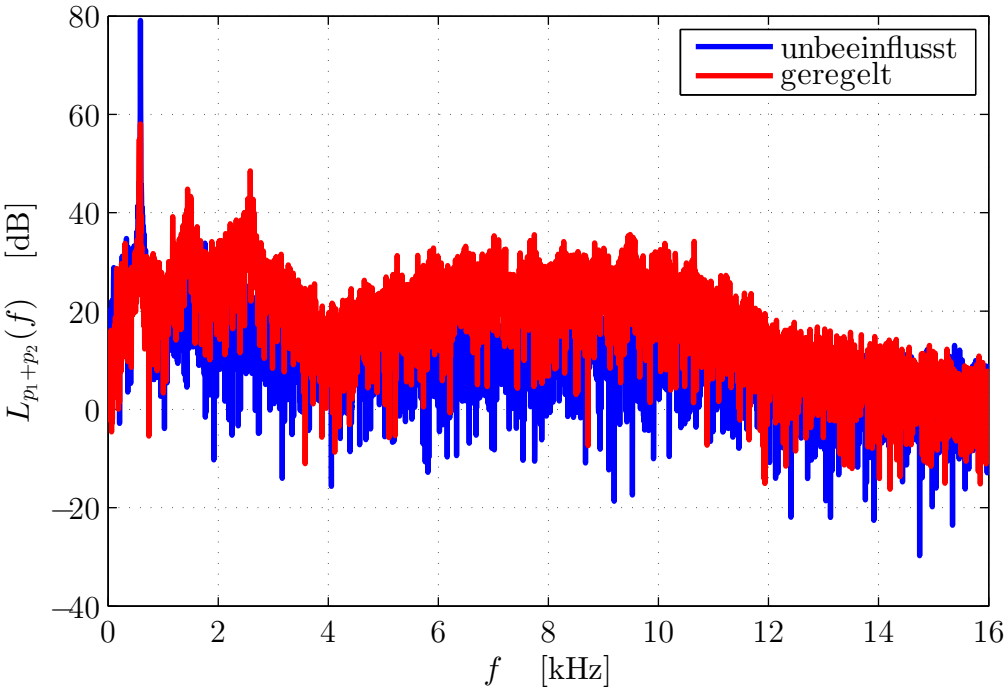

Abbildung B.15: Frequenzspektrum des Schalldruckpegels der addierten Schalldrücke bei der Druck-Druckgradient Regelung

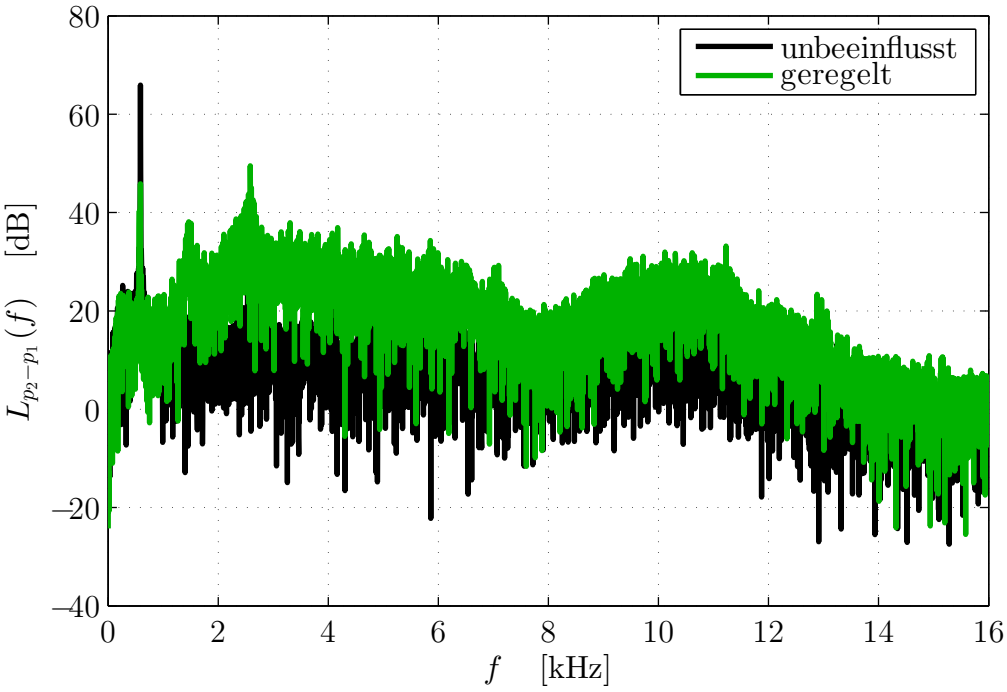

Abbildung B.16: Frequenzspektrum des Schalldruckpegels der subtrahierten Schalldrücke bei der Druck-Druckgradient Regelung

C C-Code implementierter Algorithmen

C.1 C-Code einkanalige adaptive Steuerung

```
1  //
2  // File:    ff_1x1x1.c
3  //
4  // Project: FxLMS 1x1x1 feedforward
5  //
6  // Author:  Dennis Crantz
7  //
8
9  #include <math.h>
10 #include "dsk6713_aic23.h"  // codec-DSK support file
11
12 Uint32 fs = DSK6713_AIC23_FREQ_48KHZ;  // set
      sampling rate
13 Uint16 inputsource = 0x0011;  // select line_in as
      input
14 #define LEFT 0
15 #define RIGHT 1
16 union {Uint32 combo;
17        short channel[2];
18 } io_data;
19
20 // 2. porder IIR filter - high pass
21 // w0 = 2*250Hz/fs -> w0 = 200/24000
22 float a[2] = {-1.953727949140776, 0.954774559921040};
        // {a1 a2}
23 float b[3] = {0.977125627265454, -1.954251254530908,
      0.977125627265454};  // {b0 b1 b2}
24 float inR_x[3] = {0.0, 0.0, 0.0};  // {x[k] x[k-1] x[
      k-2]}
25 float inR_y[3] = {0.0, 0.0, 0.0};  // {y[k] y[k-1] y[
```

```
    k-2]}
26
27  #define PIx2 6.2831853071795864  // define 2 times pi
        (2*3.14159....)
28
29  #define N_SIN 1600  // f_sin = fs*D_SIN/N_SIN ;
    cancel the fraction ...
30  #define D_SIN 19  // ... N_SIN/D_SIN for smallest
    memory usage
31  int sin_loop = 0;
32  short sin_table[N_SIN];
33
34  int safety = 0;  // savety
35  #define MAX_AMP 700  // max amplitude befor safety
    shutdown
36
37  #define N_s 2  // number of taps for secondary path
    modell
38  #define N_w 2  // number of taps for adaptive filter
39  #if N_w > N_s  // maximum number of tabs of filters
40      #define N_m N_w
41  #else
42      #define N_m N_s
43  #endif
44
45  #define N_secMod 50000  // number of itterations for
    secondary path modelling
46
47  int secPathMod_count = N_secMod;  // counter variable
        for sec path modelling
48  #define MOD_AMP 300  // amplitude for secondary path
    modelling
49
50  int step = 0;  // step (0 = sec path modelling; 1 =
    wait; 2 = adaptive filtering)
51  #define WAIT 50000  // cycles to wait for adaption
    algorithm
52  int wait_count = WAIT;  // counter wait for adaptive
    filter after sec path mod
53  int secPathTrue = 0;  // logical var for sec path
    modelling after wait
54
55  float mu = 0;                                    //
    LMS step width
```

```
56  float NP = 0;  // signal power times N for step width
        calc (NLMS)
57
58  float s[N_s];  // secondary path modell
59  float w[N_w];  // filter
60
61  float e = 0;  // error signal
62
63  float xs = 0;  // vectorproduct result x*s
64  float d = 0;  // input signal buffer
65
66  float x[N_m];  // reference signal
67  float x_p[N_w];  // filtered primary signal buffer
68  float y = 0;  // output signal buffer
69
70  void init_sine_table() {
71      int i;
72
73      for(i = 0; i < N_SIN; i++)
74          sin_table[i] = MOD_AMP*sin(PIx2*D_SIN/N_SIN*i
               );
75  }
76
77  void init_vec() {  // initialize vectors
78      int i;
79
80      for(i = 0; i < N_s; i++) {
81          s[i] = 0.0;
82      }
83      for(i = 0; i < N_w; i++) {
84          w[i]   = 0.0;
85          x_p[i] = 0.0;
86      }
87      for(i = 0; i < N_m; i++) {
88          x[i] = 0.0;
89      }
90  }
91
92  interrupt void c_int11() {  // interrupt service
        routine
93      int i;
94
95      io_data.combo = input_sample();  // input 32-bit
            sample
```

```
96
97      // biquad filter
98      inR_x[2] = inR_x[1]; inR_x[1] = inR_x[0];  //
           shift IIR input registers
99      inR_y[2] = inR_y[1]; inR_y[1] = inR_y[0];  //
           shift IIR input registers
100     inR_x[0] = io_data.channel[RIGHT];  // get new
           input value
101     inR_y[0] = b[0]*inR_x[0] + b[1]*inR_x[1] + b[2]*
           inR_x[2] - a[0]*inR_y[1] - a[1]*inR_y[2];  //
           calc IIR output
102     // wait a moment
103     if(step == 0) {
104         if(wait_count-- == 0) {
105             step++;
106             wait_count = WAIT;
107         }
108         io_data.channel[RIGHT] = sin_table[sin_loop];
109         io_data.channel[LEFT] = 0;
110         if(++sin_loop == N_SIN)
111             sin_loop = 0;
112     }
113     // secondary path modelling
114     else if(step == 1) {
115         d = inR_y[0];  // update desired signal
116         NP = x[0]*x[0];
117         for(i = N_s-1; i > 0; i--)
118             NP += x[i]*x[i];  // calc power of x
119         mu = 1E-3/(1 + NP);  // calc step width
120         xs = x[0]*s[0];
121         for(i = N_s-1; i > 0; i--)
122             xs += x[i]*s[i];
123         e = d - xs;  // calc error
124         for(i = N_s-1; i >= 0; i--)
125             s[i] += mu*e*x[i];  // calc sec path
                filter coeffs
126         for(i = N_s-2; i >= 0; i--)
127             x[i+1] = x[i];  // shift ref buffer
128         x[0] = sin_table[sin_loop];  // update ref
               signal
129         if(++sin_loop == N_SIN)
130             sin_loop = 0;
131         io_data.channel[RIGHT] = x[0];  // set output
               data
```

```
132             io_data.channel[LEFT] = 0;
133             if(secPathMod_count-- == 0)
134                 step++;  // count down sec path modelling
                        itterations
135             }
136         }
137         // wait a moment
138         else if(step == 2) {
139             if(wait_count-- == 0) {
140                 step++;
141                 for(i = 0; i < N_m; i++) {
142                     x[i] = 0.0;
143                 }
144             }
145             io_data.channel[RIGHT] = 0;
146             io_data.channel[LEFT] = sin_table[sin_loop];
                    // update sin wave
147             if(++sin_loop == N_SIN)
148                 sin_loop = 0;
149         }
150         // adaptive filter
151         else {
152             e = inR_y[0];  // update error signal
153             for(i = N_w-2; i >= 0; i--)
154                 x_p[i+1] = x_p[i];  // shift x_p buffer
155             x_p[0] = s[0]*x[0];
156             for(i = N_s-1; i > 0; i--)
157                 x_p[0] += s[i]*x[i];  // calc new d_h_s
158             NP = x_p[0]*x_p[0];
159             for(i = N_w-1; i > 0; i--)
160                 NP += x_p[i]*x_p[i];  // calc N times
                        power of x_p
161             mu = 5E-5/(1 + NP);  // calc step width
162             for(i = N_w-1; i >= 0; i--)
163                 w[i] -= mu*e*x_p[i];  // calc adaptive
                        filter coeffs
164             y = x[0]*w[0];
165             for(i = N_w-1; i>0; i--)
166                 y += x[i]*w[i];  // calc new filter
                        output
167             for(i = N_m-2; i >= 0; i--)
168                 x[i+1] = x[i];  // shift ref buffer
169             x[0] = sin_table[sin_loop];  // update ref
                    signal
```

```
170             if(++sin_loop == N_SIN)
171                 sin_loop = 0;
172             if(y > MAX_AMP)
173                 safety = 1;
174             if(safety == 0) {
175                 io_data.channel[RIGHT] = y;  // set
                      output data
176                 io_data.channel[LEFT] = x[0];  // output
                      ref signal
177             }
178             else {
179                 io_data.combo = 0;  // for safety -> no
                      output!
180             }
181         }
182
183         output_sample(io_data.combo);  // output to both
              channels
184         return;
185     }
186
187     void main() {  // main function
188         init_vec();  // initialize vectors
189         init_sine_table();  // initialize sin table
190         comm_intr();  // init DSK, codec, McBSP
191         while(1);  // infinite loop, wait for interrupt
192     }
```

C.2 C-Code einkanalige adaptive Regelung

```
1  //
2  // File:    fb_1x1x1.c
3  //
4  // Project: FxLMS 1x1x1 feedback with stability
      filter
5  //
6  // Author:  Dennis Crantz
7  //
8
9  #include <math.h>
10 #include "dsk6713_aic23.h"  // codec-DSK support file
11
12 Uint32 fs = DSK6713_AIC23_FREQ_48KHZ;  // set
      sampling rate
13 Uint16 inputsource = 0x0011;  // select line_in as
      input
14 #define LEFT 0
15 #define RIGHT 1
16 union {Uint32 combo;
17        short channel[2];
18 } io_data;
19
20 // 2. order IIR filter - high pass
21 // w0 = 2*f0/fs -> w0 = 250/24000
22 float a[2] = {-1.953727949140776, 0.954774559921040};
        // {a1 a2}
23 float b[3] = {0.977125627265454, -1.954251254530908,
      0.977125627265454};  // {b0 b1 b2}
24 float inR_x[3] = {0.0, 0.0, 0.0};  // {x[k] x[k-1] x[
      k-2]}
25 float inR_y[3] = {0.0, 0.0, 0.0};  // {y[k] y[k-1] y[
      k-2]}
26
27 // 2. order IIR filter - peak filter
28 // w0 = 2*f0/fs -> w0 = 570/24000;   bw = 100/24000
29 float h_a[2] = {-1.991828209961762,
      0.997385427096602};  // {a1 a2}
30 float h_b[3] = {0.001307286451699,
      -0.001307286451699};  // {b0 b2}
31 float h_x[3] = {0.0, 0.0, 0.0};  // {x[k] x[k-1] x[k
      -2]}
```

```
32 float h_y[3] = {0.0, 0.0, 0.0};  // {y[k] y[k-1] y[k
      -2]}
33
34 #define PIx2 6.2831853071795864  // define 2 times pi
       (2*3.14159....)
35
36 #define N_SIN 1600  // f_sin = fs*D_SIN/N_SIN ;
      cancel the fraction ...
37 #define D_SIN 19  // ... N_SIN/D_SIN for smallest
      memory usage
38 int sin_loop = 0;
39 short sin_table[N_SIN];
40
41 int safety = 0;  // savety
42 #define MAX_AMP 800  // max amplitude befor safety
      shutdown
43
44 #define N_s 2  // number of taps for secondary path
      modell
45 #define N_w 2  // number of taps for adaptive filter
46 #if N_w > N_s  // maximum number of tabs of filters
47     #define N_m N_w
48 #else
49     #define N_m N_s
50 #endif
51
52 #define N_secMod 50000  // number of itterations for
      secondary path modelling
53
54 int secPathMod_count = N_secMod;  // counter variable
       for sec path modelling
55 #define MOD_AMP 300  // amplitude for secondary path
      modelling
56
57 int step = 0;  // step (0 = sec path modelling; 1 =
      wait; 2 = adaptive filtering)
58 #define WAIT 50000
59 int wait_count = WAIT;  // counter wait for adaptive
      filter after sec path mod
60
61 float mu = 0;  // LMS step width
62 float NP = 0;  // signal power times N for step width
       calc (NLMS)
63
```

```
64  float s[N_s];  // secondary path modell
65  float w[N_w];  // filter
66
67  float e = 0;  // error signal
68
69  float xs = 0;  // vectorproduct result x*s
70  float d = 0;  // input signal buffer
71
72  float x[N_m];  // reference signal buffer sec path
       modelling
73
74  float x_p[N_w];  // filtered primary signal buffer
75  float y[N_s];  // output signal buffer
76  float y_h_p = 0;  // filtered output signal
77
78  void init_sine_table() {
79      int i;
80
81      for(i = 0; i < N_SIN; i++)
82          sin_table[i] = MOD_AMP*sin(PIx2*D_SIN/N_SIN*i
               );
83  }
84
85  void init_vec() {  // initialize vectors
86      int i;
87
88      for(i = 0; i < N_s; i++) {
89          s[i] = 0.0;
90          y[i] = 0.0;
91      }
92      for(i = 0; i < N_w; i++) {
93          w[i]   = 0.0;
94          x_p[i] = 0.0;
95      }
96      for(i = 0; i < N_m; i++) {
97          x[i] = 0.0;
98      }
99  }
100
101 interrupt void c_int11() {  // interrupt service
       routine
102     int i;
103
104     io_data.combo = input_sample();  // input 32-bit
```

```
            sample
105
106         // biquad filter
107         inR_x[2] = inR_x[1]; inR_x[1] = inR_x[0];  //
               shift IIR input registers
108         inR_y[2] = inR_y[1]; inR_y[1] = inR_y[0];  //
               shift IIR input registers
109         inR_x[0] = io_data.channel[RIGHT];  // get new
               input value
110         inR_y[0] = b[0]*inR_x[0] + b[1]*inR_x[1] + b[2]*
               inR_x[2] - a[0]*inR_y[1] - a[1]*inR_y[2];  //
               calc IIR output
111         // wait a moment
112         if(step == 0) {
113             if(wait_count-- == 0) {
114                 step++;
115                 wait_count = WAIT;
116             }
117             io_data.channel[RIGHT] = sin_table[sin_loop];
118             io_data.channel[LEFT] = 0;
119             if(++sin_loop == N_SIN)
120                 sin_loop = 0;
121         }
122         // secondary path modelling
123         else if(step == 1) {
124             d = inR_y[0];  // update desired signal
125             NP = x[0]*x[0];
126             for(i = N_s-1; i > 0; i--)
127                 NP += x[i]*x[i];  // calc power of x
128             mu = 1E-3/(1 + NP);  // calc step width
129             xs = x[0]*s[0];
130             for(i = N_s-1; i > 0; i--)
131                 xs += x[i]*s[i];
132             e = d - xs;  // calc error
133             for(i = N_s-1; i >= 0; i--)
134                 s[i] += mu*e*x[i];  // calc sec path
                       filter coeffs
135             for(i = N_s-2; i >= 0; i--)
136                 x[i+1] = x[i];  // shift ref buffer
137             x[0] = sin_table[sin_loop];  // update ref
                   signal
138             if(++sin_loop == N_SIN)
139                 sin_loop = 0;
140             io_data.channel[RIGHT] = x[0];  // set output
```

```
                data
141             io_data.channel[LEFT] = 0;
142             if(secPathMod_count-- == 0)
143                 step++;  // count down sec path modelling
                        itterations
144         }
145         // wait a moment
146         else if(step == 2) {
147             if(wait_count-- == 0) {
148                 step++;
149                 for(i = 0; i < N_m; i++) {
150                     x[i] = 0.0;
151                 }
152             }
153             io_data.channel[RIGHT] = 0;
154             io_data.channel[LEFT] = sin_table[sin_loop];
                    // update sin wave
155             if(++sin_loop == N_SIN)
156                 sin_loop = 0;
157         }
158         // adaptive filter
159         else {
160             e = inR_y[0];  // update error signal
161             // filter H
162             h_x[2] = h_x[1]; h_x[1] = h_x[0];  // shift
                    IIR input registers
163             h_y[2] = h_y[1]; h_y[1] = h_y[0];  // shift
                    IIR output registers
164             h_x[0] = e - y_h_p;  // calc new input value
165             h_y[0] = h_b[0]*h_x[0] + h_b[1]*h_x[2] - h_a
                    [0]*h_y[1] - h_a[1]*h_y[2];  // calc IIR
                    output
166             for(i = N_m-2; i >= 0; i--)
167                 x[i+1] = x[i];  // shift x buffer
168             x[0] = h_y[0];  // set new x
169             for(i = N_w-2; i >= 0; i--)
170                 x_p[i+1] = x_p[i];  // shift x_p buffer
171             x_p[0] = s[0]*x[0];
172             for(i = N_s-1; i > 0; i--)
173                 x_p[0] += s[i]*x[i];                //
                        calc new x_p
174             NP = x_p[0]*x_p[0];
175             for(i = N_w-1; i > 0; i--)
176                 NP += x_p[i]*x_p[i];                //
```

```
                calc power of x_p
177         mu = 1E-5/(1 + NP);                         //
               calc step width
178         for(i = N_w-1; i >= 0; i--)
179             w[i] -= mu*e*x_p[i];                    //
                   calc adaptive filter coeffs
180         for(i = N_s-2; i >= 0; i--)
181             y[i+1] = y[i];                          //
                   shift filter output buffer
182         y[0] = x[0]*w[0];
183         for(i = N_w-1; i>0; i--)
184             y[0] += x[i]*w[i];                      //
                   calc new filter output
185         y_h_p = y[0]*s[0];
186         for(i = N_s-1; i>0; i--)
187             y_h_p += y[i]*s[i];                     //
                   calc new y_h_p
188         if(y[0] > MAX_AMP)
189             safety = 1;
190         if(safety == 0) {
191             io_data.channel[RIGHT] = y[0];          //
                   set output data
192             io_data.channel[LEFT]  = sin_table[
                   sin_loop];
193         }
194         else {
195             io_data.combo = 0;                      //
                   for safety -> no output!
196         }
197                                                     //
                                                           update

                                                           sin

                                                           wave

198         if(++sin_loop == N_SIN)
199             sin_loop = 0;
200     }
201
202     output_sample(io_data.combo);                   //
           output to both channels
203     return;
204 }
```

```
205
206 void main() {                                          //
       main function
207   init_vec();                                         //
        initialize vectors
208   init_sine_table();                                  //
        initialize sin table
209   comm_intr();                                        //
        init DSK, codec, McBSP
210   while(1);                                           //
        infinite loop, wait for interrupt
211 }
```

C.3 C-Code mehrkanalige adaptive Steuerung

```
1  //
2  // File:    ff_1x1x2.c
3  //
4  // Project: FxLMS 1x1x2 feedforward
5  //
6  // Author:  Dennis Crantz
7  //
8
9  #include <math.h>
10 #include "dsk6713_aic23.h"  // codec-DSK support file
11
12 Uint32 fs = DSK6713_AIC23_FREQ_48KHZ;  // set
      sampling rate
13 Uint16 inputsource = 0x0011;  // select line_in as
      input
14 #define LEFT 0
15 #define RIGHT 1
16 union {Uint32 combo;
17         short channel[2];
18 } io_data;
19
20 // 2. porder IIR filter - high pass
21 // w0 = 2*f0/fs -> w0 = 250/24000
22 float a[2] = {-1.953727949140776, 0.954774559921040};
       // {a1 a2}
23 float b[3] = {0.977125627265454, -1.954251254530908,
      0.977125627265454};  // {b0 b1 b2}
24 float inR_x[3] = {0.0, 0.0, 0.0};  // {x[k] x[k-1] x[
      k-2]}
25 float inL_x[3] = {0.0, 0.0, 0.0};
26 float inR_y[3] = {0.0, 0.0, 0.0};  // {y[k] y[k-1] y[
      k-2]}
27 float inL_y[3] = {0.0, 0.0, 0.0};
28
29 #define PIx2 6.2831853071795864  // define 2 times pi
       (2*3.14159....)
30
31 #define N_SIN 1600  // f_sin = fs*D_SIN/N_SIN ;
      cancel the fraction ...
32 #define D_SIN 19  // ... N_SIN/D_SIN for smallest
      memory usage
```

```
33 int sin_loop = 0;
34 short sin_table[N_SIN];
35
36 int safety = 0;  // savety
37 #define MAX_AMP 600  // max amplitude befor safety
      shutdown
38
39 #define N_s 2  // number of taps for secondary path
      modell
40 #define N_w 2  // number of taps for adaptive filter
41 #if N_w > N_s  // maximum number of tabs of filters
42     #define N_m N_w
43 #else
44     #define N_m N_s
45 #endif
46
47 #define N_secMod 50000  // number of itterations for
      secondary path modelling
48
49 int secPathMod_count = N_secMod;  // counter variable
       for sec path modelling
50 #define MOD_AMP 300  // amplitude for secondary path
      modelling
51
52 int step = 0;  // step (0 = sec path modelling; 1 =
      wait; 2 = adaptive filtering)
53 #define WAIT 50000  // cycles to wait for adaption
      algorithm
54 int wait_count = WAIT;  // counter wait for adaptive
      filter after sec path mod
55 int secPathTrue = 0;  // logical var for sec path
      modelling after wait
56
57 float NP = 0;  // signal power times N for step width
       calc (NLMS)
58 float NP_1, NP_2, mu_1, mu_2;
59
60 float s_1[N_s];  // secondary path modells
61 float s_2[N_s];
62 float w[N_w];  // filter
63
64 float e_1 = 0;  // error signals
65 float e_2 = 0;
66
```

```
67  float xs_1 = 0;  // vectorproduct results x*s
68  float xs_2 = 0;
69  float d_1 = 0;  // input signals
70  float d_2 = 0;
71
72  float x[N_m];  // reference signal
73  float x_p_1[N_w];  // filtered primary signal buffer
74  float x_p_2[N_w];
75  float y = 0;  // output signal
76
77  void init_sine_table() {
78      int i;
79
80      for(i = 0; i < N_SIN; i++)
81          sin_table[i] = MOD_AMP*sin(PIx2*D_SIN/N_SIN*i
               );
82  }
83
84  void init_vec() {  // initialize vectors
85      int i;
86
87      for(i = 0; i < N_s; i++) {
88          s_1[i] = 0.0;
89          s_2[i] = 0.0;
90      }
91      for(i = 0; i < N_w; i++) {
92          w[i]     = 0.0;
93          x_p_1[i] = 0.0;
94          x_p_2[i] = 0.0;
95      }
96      for(i = 0; i < N_m; i++) {
97          x[i] = 0.0;
98      }
99  }
100
101 interrupt void c_int11() {  // interrupt service
       routine
102     int i;
103
104     io_data.combo = input_sample();  // input 32-bit
          sample
105
106     // biquad filter
107     inR_x[2] = inR_x[1]; inR_x[1] = inR_x[0];  //
```

```
            shift IIR input registers
108         inL_x[2] = inL_x[1]; inL_x[1] = inL_x[0];
109         inR_y[2] = inR_y[1]; inR_y[1] = inR_y[0];  //
            shift IIR input registers
110         inL_y[2] = inL_y[1]; inL_y[1] = inL_y[0];
111         inR_x[0] = io_data.channel[RIGHT];  // get new
            input value
112         inL_x[0] = io_data.channel[LEFT];
113         inR_y[0] = b[0]*inR_x[0] + b[1]*inR_x[1] + b[2]*
            inR_x[2] - a[0]*inR_y[1] - a[1]*inR_y[2];
114         inL_y[0] = b[0]*inL_x[0] + b[1]*inL_x[1] + b[2]*
            inL_x[2] - a[0]*inL_y[1] - a[1]*inL_y[2];  //
            calc IIR output
115         // wait a moment
116         if(step == 0) {
117             if(wait_count-- == 0) {
118                 step++;
119                 wait_count = WAIT;
120             }
121             io_data.channel[RIGHT] = sin_table[sin_loop];
122             io_data.channel[LEFT] = 0;
123             if(++sin_loop == N_SIN)
124                 sin_loop = 0;
125         }
126         // secondary path modelling
127         else if(step == 1) {
128             d_1 = inR_y[0];  // update desired signals
129             d_2 = inL_y[0];
130             //d_1 = inR_y[0] + inL_y[0];
131             //d_2 = inR_y[0] - inL_y[0];
132             NP = x[0]*x[0];
133             for(i = N_s-1; i > 0; i--)
134                 NP += x[i]*x[i];  // calc power of x
135             mu_1 = 5E-3/(1 + NP);  // calc step width
136             mu_2 = 5E-3/(1 + NP);
137             xs_1 = x[0]*s_1[0];
138             xs_2 = x[0]*s_2[0];
139             for(i = N_s-1; i > 0; i--) {
140                 xs_1 += x[i]*s_1[i];
141                 xs_2 += x[i]*s_2[i];
142             }
143             e_1 = d_1 - xs_1;  // calc error
144             e_2 = d_2 - xs_2;
145             for(i = N_s-1; i >= 0; i--) {
```

```
146             s_1[i] += mu_1*e_1*x[i];  // calc sec
                    path filter coeffs
147             s_2[i] += mu_2*e_2*x[i];
148         }
149         for(i = N_s-2; i >= 0; i--)
150             x[i+1] = x[i];  // shift ref buffer
151         x[0] = sin_table[sin_loop];  // update ref
                signal
152         if(++sin_loop == N_SIN)
153             sin_loop = 0;
154         io_data.channel[RIGHT] = x[0];  // set output
                data
155         io_data.channel[LEFT] = 0;
156         if(secPathMod_count-- == 0)
157             step++;  // count down sec path modelling
                    itterations
158     }
159     // wait a moment
160     else if(step == 2) {
161         if(wait_count-- == 0) {
162             step++;
163             for(i = 0; i < N_m; i++) {
164                 x[i] = 0.0;
165             }
166         }
167         io_data.channel[RIGHT] = 0;
168         io_data.channel[LEFT] = sin_table[sin_loop];
                // update sin wave
169         if(++sin_loop == N_SIN)
170             sin_loop = 0;
171 
172     }
173     // adaptive filter
174     else {
175         e_1 = inR_y[0];  // update error signals
176         e_2 = inL_y[0];
177         //e_1 = inR_y[0] + inL_y[0];
178         //e_2 = inR_y[0] - inL_y[0];
179         for(i = N_w-2; i >= 0; i--) {
180             x_p_1[i+1] = x_p_1[i];  // shift x_s
                    buffers
181             x_p_2[i+1] = x_p_2[i];
182         }
183         x_p_1[0] = s_1[0]*x[0];
```

```
184             x_p_2[0] = s_2[0]*x[0];
185             for(i = N_s-1; i > 0; i--) {
186                 x_p_1[0] += s_1[i]*x[i];  // calc new
                        d_h_s
187                 x_p_2[0] += s_2[i]*x[i];
188             }
189             NP_1 = x_p_1[0]*x_p_1[0];
190             NP_2 = x_p_2[0]*x_p_2[0];
191             for(i = N_w-1; i > 0; i--) {
192                 NP_1 += x_p_1[i]*x_p_1[i];  // calc N
                        times power of x_s
193                 NP_2 += x_p_2[i]*x_p_2[i];
194             }
195             mu_1 = 1E-5/(1 + NP_1);  // calc step width
196             mu_2 = 1E-5/(1 + NP_2);
197             for(i = N_w-1; i >= 0; i--)
198                 w[i] = w[i] - mu_1*e_1*x_p_1[i] - mu_2*
                        e_2*x_p_2[i];  // calc adaptive filter
                         coeffs
199             y = x[0]*w[0];
200             for(i = N_w-1; i>0; i--)
201                 y += x[i]*w[i];  // calc new filter
                        output
202             for(i = N_m-2; i >= 0; i--)
203                 x[i+1] = x[i];  // shift ref buffer
204             x[0] = sin_table[sin_loop];  // update ref
                    signal
205             if(++sin_loop == N_SIN)
206                 sin_loop = 0;
207             if(y > MAX_AMP)
208                 safety = 1;
209             if(safety == 0) {
210                 io_data.channel[RIGHT] = y;  // set
                        output data
211                 io_data.channel[LEFT] = x[0];  // output
                        ref signal
212             }
213             else {
214                 io_data.combo = 0;  // for safety -> no
                        output!
215             }
216         }
217
218         output_sample(io_data.combo);  // output to both
```

```
            channels
219         return;
220     }
221
222     void main() {  // main function
223         init_vec();  // initialize vectors
224         init_sine_table();  // initialize sin table
225         comm_intr();  // init DSK, codec, McBSP
226         while(1);  // infinite loop, wait for interrupt
227     }
```

C.4 C-Code mehrkanalige adaptive Regelung

```
1  //
2  // File:    fb_1x2x2.c
3  //
4  // Project: FxLMS 1x2x2 feedback with stability
      filter
5  //
6  // Author:  Dennis Crantz
7  //
8
9  #include <math.h>
10 #include "dsk6713_aic23.h"  // codec-DSK support file
11
12 Uint32 fs = DSK6713_AIC23_FREQ_32KHZ;  // set
      sampling rate
13 Uint16 inputsource = 0x0011;  // select line_in as
      input
14 #define LEFT 0
15 #define RIGHT 1
16 union {Uint32 combo;
17        short channel[2];
18 } io_data;
19
20 // 2. order IIR filter - high pass
21 // w0 = 2*f0/fs -> w0 = 250/16000
22 float a[2] = {-1.930606427219668, 0.932934731756612};
        // {a1 a2}
23 float b[3] = {0.965885289744070, -1.931770579488140,
      0.965885289744070};  // {b0 b1 b2}
24 float inR_x[3] = {0.0, 0.0, 0.0};  // {x[k] x[k-1] x[
      k-2]}
25 float inL_x[3] = {0.0, 0.0, 0.0};  // {x[k] x[k-1] x[
      k-2]}
26 float inR_y[3] = {0.0, 0.0, 0.0};  // {y[k] y[k-1] y[
      k-2]}
27 float inL_y[3] = {0.0, 0.0, 0.0};  // {y[k] y[k-1] y[
      k-2]}
28
29 // 2. order IIR filter - peak filter
30 // w0 = 2*f0/fs -> w0 = 570/16000;   bw = 15/16000
31 float h_a[2] = {-1.984564638745086,
      0.997059085615275};  // {a1 a2}
```

```
32  float h_b[3] = {0.001470457192363,
       -0.001470457192363};  // {b0 b2}
33  float h_x[3] = {0.0, 0.0, 0.0};  // {x[k] x[k-1] x[k
       -2]}
34  float h_y[3] = {0.0, 0.0, 0.0};  // {y[k] y[k-1] y[k
       -2]}
35
36  #define PIx2 6.2831853071795864  // define 2 times pi
       (2*3.14159....)
37
38  #define N_SIN 3200  // f_sin = fs*D_SIN/N_SIN ;
39  #define D_SIN 57  // cancel the fraction N_SIN/D_SIN
       for smallest memory usage
40  int sin_loop = 0;
41  short sin_table[N_SIN];
42
43  int safety = 0;  // savety
44  #define MAX_AMP 800  // max amplitude befor safety
       shutdown
45
46  #define N_s 2  // number of taps for secondary path
       modell
47  #define N_w 2  // number of taps for adaptive filter
48  #if N_w > N_s  // maximum number of tabs of filters
49      #define N_m N_w
50  #else
51      #define N_m N_s
52  #endif
53
54  #define N_secMod 50000  // number of itterations for
       secondary path modelling
55
56  int secPathMod_count = N_secMod;  // counter variable
       for sec path modelling
57  #define MOD_AMP 300  // amplitude for secondary path
       modelling
58
59  int step = 0;  // step (0-3 = sec path modelling; 4 =
       wait; 5 = adaptive filtering)
60  #define WAIT 50000
61  int wait_count = WAIT;  // counter wait for adaptive
       filter after sec path mod
62
63  float mu_11, mu_21, mu_12, mu_22;  // LMS step width
```

```
64  float NP_11, NP_21, NP_12, NP_22;  // signal power
       times N for step width calc (NLMS)
65
66  float s_11[N_s];  // secondary path modells
67  float s_21[N_s];
68  float s_12[N_s];
69  float s_22[N_s];
70  float w_1[N_w];  // filter
71  float w_2[N_w];
72
73  float e_1, e_2;  // error signal
74
75  float xs_1, xs_2;  // vectorproduct result x*s
76  float d_1, d_2;  // input signal
77
78  float x[N_m];  // reference signal buffer
79
80  float x_p_11[N_w];  // filtered primary signal buffer
81  float x_p_21[N_w];
82  float x_p_12[N_w];
83  float x_p_22[N_w];
84  float y_1[N_s], y_2[N_s];  // output signal buffer
85  float y_h_p_11 = 0;  // filtered output signal
86  float y_h_p_12 = 0;
87
88  void init_sine_table() {
89      int i;
90
91      for(i = 0; i < N_SIN; i++)
92          sin_table[i] = MOD_AMP*sin(PIx2*D_SIN/N_SIN*i
               );
93  }
94
95  void init_vec() {  // initialize vectors
96      int i;
97
98      for(i = 0; i < N_s; i++) {
99          s_11[i] = 0.0;
100         s_21[i] = 0.0;
101         s_12[i] = 0.0;
102         s_22[i] = 0.0;
103         y_1[i]  = 0.0;
104         y_2[i]  = 0.0;
105     }
```

```
106       for(i = 0; i < N_w; i++) {
107           w_1[i]    = 0.0;
108           w_2[i]    = 0.0;
109           x_p_11[i] = 0.0;
110           x_p_21[i] = 0.0;
111           x_p_12[i] = 0.0;
112           x_p_22[i] = 0.0;
113       }
114       for(i = 0; i < N_m; i++) {
115           x[i] = 0.0;
116       }
117   }
118
119   interrupt void c_int11() {  // interrupt service
          routine
120       int i;
121
122       io_data.combo = input_sample();  // input 32-bit
            sample
123
124       // biquad filter
125       inR_x[2] = inR_x[1]; inR_x[1] = inR_x[0];  //
            shift IIR input registers
126       inL_x[2] = inL_x[1]; inL_x[1] = inL_x[0];
127       inR_y[2] = inR_y[1]; inR_y[1] = inR_y[0];  //
            shift IIR input registers
128       inL_y[2] = inL_y[1]; inL_y[1] = inL_y[0];
129       inR_x[0] = io_data.channel[RIGHT];  // get new
            IIR input values
130       inL_x[0] = io_data.channel[LEFT];
131       inR_y[0] = b[0]*inR_x[0] + b[1]*inR_x[1] + b[2]*
            inR_x[2] - a[0]*inR_y[1] - a[1]*inR_y[2];
132       inL_y[0] = b[0]*inL_x[0] + b[1]*inL_x[1] + b[2]*
            inL_x[2] - a[0]*inL_y[1] - a[1]*inL_y[2];  //
            calc IIR output
133       // wait a moment
134       if(step == 0 || step == 2) {
135           if(wait_count-- == 0) {
136               step++;
137               wait_count = WAIT;
138           }
139           if(step == 0) {
140               io_data.channel[RIGHT] = sin_table[
                    sin_loop];
```

```
141                 io_data.channel[LEFT]  = 0;
142             } else {
143                 io_data.channel[RIGHT] = 0;
144                 io_data.channel[LEFT]  = sin_table[
                        sin_loop];
145             }
146             if(++sin_loop == N_SIN)
147                 sin_loop = 0;
148         }
149         // secondary path modelling
150         else if(step == 1 || step == 3) {
151             d_1 = inR_y[0];  // update desired signals
152             d_2 = inL_y[0];
153             //d_1 = inR_y[0] + inL_y[0];
154             //d_2 = inR_y[0] - inL_y[0];
155             NP_11 = x[0]*x[0];
156             for(i = N_s-1; i > 0; i--)
157                 NP_11 += x[i]*x[i];  // calc power of x
158             mu_11 = 1E-3/(1 + NP_11);  // calc step width
159             if(step == 1) {
160                 xs_1 = x[0]*s_11[0];
161                 xs_2 = x[0]*s_21[0];
162                 for(i = N_s-1; i > 0; i--) {
163                     xs_1 += x[i]*s_11[i];
164                     xs_2 += x[i]*s_21[i];
165                 }
166                 e_1 = d_1 - xs_1;  // calc errors
167                 e_2 = d_2 - xs_2;
168                 for(i = N_s-1; i >= 0; i--) {
169                     s_11[i] += mu_11*e_1*x[i];  // calc
                            sec path filter coeffs
170                     s_21[i] += mu_11*e_2*x[i];
171                 }
172             } else {
173                 xs_1 = x[0]*s_12[0];
174                 xs_2 = x[0]*s_22[0];
175                 for(i = N_s-1; i > 0; i--) {
176                     xs_1 += x[i]*s_12[i];
177                     xs_2 += x[i]*s_22[i];
178                 }
179                 e_1 = d_1 - xs_1;  // calc errors
180                 e_2 = d_2 - xs_2;
181                 for(i = N_s-1; i >= 0; i--) {
182                     s_12[i] += mu_11*e_1*x[i];  // calc
```

```
                    sec path filter coeffs
183                 s_22[i] += mu_11*e_2*x[i];
184             }
185         }
186         for(i = N_s-2; i >= 0; i--)
187             x[i+1] = x[i];  // shift ref buffer
188         x[0] = sin_table[sin_loop];             //
              update ref signal
189         if(++sin_loop == N_SIN)
190             sin_loop = 0;
191         if(step == 1) {
192             io_data.channel[RIGHT] = x[0];  // set
                  output data
193             io_data.channel[LEFT]  = 0;
194         } else {
195             io_data.channel[RIGHT] = 0;
196             io_data.channel[LEFT]  = x[0];
197         }
198         if(secPathMod_count-- == 0) {
199             step++;  // count down sec path modelling
                  itterations
200             secPathMod_count = N_secMod;
201         }
202     }
203     // wait a moment
204     else if(step == 4) {
205         if(wait_count-- == 0) {
206             step++;
207             for(i = 0; i < N_m; i++) {
208                 x[i] = 0.0;
209             }
210         }
211         io_data.combo = 0;
212     }
213     // adaptive filter
214     else {
215         e_1 = inR_y[0];  // update error signals
216         e_2 = inL_y[0];
217         //e_1 = inR_y[0] + inL_y[0];
218         //e_2 = inR_y[0] - inL_y[0];
219         // filter H
220         h_x[2] = h_x[1]; h_x[1] = h_x[0];  // shift
              IIR input registers
221         h_y[2] = h_y[1]; h_y[1] = h_y[0];  // shift
```

```
            IIR output registers
222         h_x[0] = e_1 - y_h_p_11 - y_h_p_12;  // calc
            new input value
223         h_y[0] = h_b[0]*h_x[0] + h_b[1]*h_x[2] - h_a
            [0]*h_y[1] - h_a[1]*h_y[2];  // calc IIR
            output
224         for(i = N_m-2; i >= 0; i--)
225             x[i+1] = x[i];  // shift x buffer
226         x[0] = h_y[0];  // set new x
227         for(i = N_w-2; i >= 0; i--) {  // shift x_s
            buffers
228             x_p_11[i+1] = x_p_11[i];
229             x_p_12[i+1] = x_p_12[i];
230             x_p_21[i+1] = x_p_21[i];
231             x_p_22[i+1] = x_p_22[i];
232         }
233         x_p_11[0] = s_11[0]*x[0];
234         x_p_12[0] = s_21[0]*x[0];
235         x_p_21[0] = s_12[0]*x[0];
236         x_p_22[0] = s_22[0]*x[0];
237         for(i = N_s-1; i > 0; i--) {
238             x_p_11[0] += s_11[i]*x[i];  // calc new
                x_s
239             x_p_12[0] += s_21[i]*x[i];
240             x_p_21[0] += s_12[i]*x[i];
241             x_p_22[0] += s_22[i]*x[i];
242         }
243         NP_11 = x_p_11[0]*x_p_11[0];
244         NP_12 = x_p_12[0]*x_p_12[0];
245         NP_21 = x_p_21[0]*x_p_21[0];
246         NP_22 = x_p_22[0]*x_p_22[0];
247         for(i = N_w-1; i > 0; i--) {
248             NP_11 += x_p_11[i]*x_p_11[i];  // calc
                power of x_s
249             NP_12 += x_p_12[i]*x_p_12[i];
250             NP_21 += x_p_21[i]*x_p_21[i];
251             NP_22 += x_p_22[i]*x_p_22[i];
252         }
253         mu_11 = 5E-6/(1 + NP_11);  // calc step width
254         mu_12 = 5E-6/(1 + NP_12);
255         mu_21 = 5E-6/(1 + NP_21);
256         mu_22 = 5E-6/(1 + NP_22);
257         for(i = N_w-1; i >= 0; i--) {  // calc
            adaptive filter coeffs
```

```
258             w_1[i] = w_1[i] - mu_11*e_1*x_p_11[i] -
                   mu_12*e_2*x_p_12[i];
259             w_2[i] = w_2[i] - mu_21*e_1*x_p_21[i] -
                   mu_22*e_2*x_p_22[i];
260         }
261         for(i = N_s-2; i >= 0; i--) {  // shift
               filter output buffer
262             y_1[i+1] = y_1[i];
263             y_2[i+1] = y_2[i];
264         }
265         y_1[0] = x[0]*w_1[0];
266         y_2[0] = x[0]*w_2[0];
267         for(i = N_w-1; i>0; i--) {  // calc new
               filter output
268             y_1[0] += x[i]*w_1[i];
269             y_2[0] += x[i]*w_2[i];
270         }
271         y_h_p_11 = y_1[0]*s_11[0];
272         y_h_p_12 = y_2[0]*s_12[0];
273         for(i = N_s-1; i>0; i--) {  // calc new y_h_s
274             y_h_p_11 += y_1[i]*s_11[i];
275             y_h_p_12 += y_2[i]*s_12[i];
276         }
277         if(y_1[0] > MAX_AMP || y_2[0] > MAX_AMP)
278             safety = 1;
279         if(safety == 0) {
280             io_data.channel[RIGHT] = y_1[0];  // set
                   output data
281             io_data.channel[LEFT]  = y_2[0];
282         }
283         else {
284             io_data.combo = 0;  // for safety -> no
                   output!
285         }
286     }
287
288     output_sample(io_data.combo);  // output to both
           channels
289     return;
290 }
291
292 void main() {  // main function
293     init_vec();  // initialize vectors
294     init_sine_table();  // initialize sin table
```

```
295         comm_intr();  // init DSK, codec, McBSP
296         while(1);  // infinite loop, wait for interrupt
297     }
```

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „– bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] – ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Dieses Blatt, mit der folgenden Erklärung, ist nach Fertigstellung der Abschlussarbeit durch den Studierenden auszufüllen und jeweils mit Originalunterschrift als letztes Blatt in das Prüfungsexemplar der Abschlussarbeit einzubinden.
Eine unrichtig abgegebene Erklärung kann -auch nachträglich- zur Ungültigkeit des Studienabschlusses führen.

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: Crantz

Vorname: Dennis

dass ich die vorliegende Bachelorarbeit bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Aktive Regelung von Schalldruck und –schnelle mit einem digitalen Signalprozessor

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

- die folgende Aussage ist bei Gruppenarbeiten auszufüllen und entfällt bei Einzelarbeiten -

Die Kennzeichnung der von mir erstellten und verantworteten Teile der Bachelorarbeit ist erfolgt durch:

Hamburg	07.10.2014	
Ort	Datum	Unterschrift im Original