Universität Erlangen-Nürnberg

Lehrstuhl für Multimediakommunikation und Signalverarbeitung

Prof. Dr.-Ing. Walter Kellermann

Bachelorarbeit

Implementierung und Evaluation eines Demonstrators zur adaptiven Filterung mittels LMS-artiger Algorithmen

von Markus Jonscher

September 2010

Betreuer: Prof. Dr.-Ing. Walter Kellermann
Dipl.-Ing. Marcus Zeller

Lehrstuhl für Multimediakommunikation und Signalverarbeitung
Prof. Dr.-Ing. W. Kellermann

Friedrich-Alexander-Universität
Erlangen-Nürnberg

Bachelorarbeit

für

Herrn cand. ing. Markus Jonscher

Implementierung und Evaluation eines Demonstrators zur adaptiven Filterung mittels LMS-artiger Algorithmen

In dieser Bachelorarbeit soll ein Demonstrator entworfen und implementiert werden, der das intuitive Verständnis adaptiver Filter fördern soll. Da ein Großteil der in der Praxis verwendeten Adaptionsalgorithmen aus Gründen der Komplexität und Robustheit auf dem LMS-Ansatz ('Least-Mean-Square') basieren, steht dieses Konzept im Fokus der Arbeit. Das Hauptaugenmerk liegt dabei auf einer Visualisierung der einzelnen Adaptionsschritte von Iteration zu Iteration. Die Wirkungsweise soll deshalb an Hand der zeitlichen Entwicklung des Restfehlers bzw. der Koeffizientenwerte dargestellt werden.

Als Beispiel soll die Identifikation linearer akustischer Systeme betrachtet werden. Dabei soll auch eine schritthaltende Evaluierung des Abstand zum wahren System implementiert werden, wobei die Impulsantworten sowohl von künstlichen Systemen, als auch aus real gemessene Daten ermittelt werden sollen. Als Eingangssignale des Filters sollen unterschiedliche Rausch- und Sprachsignale zur Verfügung stehen, um deren Auswirkungen auf die unterschiedlichen Konvergenzeigenschaften deutlich zu machen. Neben der Implementierung des ursprünglichen LMS-Algorithmus bieten sich auch Vergleiche mit Modifikationen wie NLMS („Normalized LMS") und PNLMS („Proportionate NLMS") an.

Der Demonstrator soll mit Hilfe des GUI-Toolkits von MATLAB entworfen werden. Auf eine ausführliche Dokumentation des Programmcodes und der Funktionalität des Demonstrators, sowie eine sorgfältige Ausarbeitung der grundlegenden Theorie adaptiver Filter wird besonderer Wert gelegt.

Beginn: 19.04.2010
Ende: 19.09.2010

(Prof. Dr.-Ing. W. Kellermann)

Erklärung

Ich versichere, dass ich die vorliegende Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Ort, Datum

Unterschrift

Inhaltsverzeichnis

Kurzfassung

In der vorliegenden Arbeit wird ein Demonstrator entworfen um das intuitive Verständnis adaptiver Filter zu fördern. Die Implementierung erfolgt mittels des GUI-Toolkits von MATLAB. Der Fokus liegt auf LMS-artigen Algorithmen, da diese in der Praxis aus Gründen der Komplexität und Robustheit häufig zum Einsatz kommen.

Neben einer Behandlung der Theorie und entsprechender Herleitung der unterschiedlichen Algorithmen wird die Implementierung und Bedienung der GUI beschrieben und zusätzlich ein Vergleich der einzelnen, hergeleiteten Algorithmen anhand bestimmter Kriterien durchgeführt.

Kapitel 1

Einleitung

1.1 Szenarien adaptiver Filterung

Die Anwendung von fest dimensionierten Filtern ist sinnvoll, falls keine Anpassungen an sich ändernde Umgebungsbedingungen während des Einsatzes notwendig sind. Jedoch sind in der Praxis die zu verarbeitenden Signale in vielen Fällen nichtstationär und/oder es handelt sich um ein zeitvariantes System, so dass kein optimaler Filterentwurf im Vorfeld möglich ist. Aus diesem Grund sind adaptive Filter notwendig, die sich selbstständig an veränderte Bedingungen anpassen.

Ein adaptives Filter in der digitalen Signalverarbeitung ist ein spezielles digitales Filter, das die Eigenschaft besitzt, seine Übertragungsfunktion im Betrieb selbstständig verändern zu können. Zu diesem Zweck wird im Regelfall ein FIR-Filter mit einem Einstellnetzwerk versehen, das nach bestimmten Regeln die Filterkoeffizienten des Transversalfilters verändern kann. Einige Einsatzgebiete adaptiver Filter sind unter anderem Echokompensation, Beamforming, seismologische Modellierung oder auch Radar und Zielverfolgung [11].

Diese unterschiedlichen Anwendungen lassen sich grob in vier Klassen einteilen.

- Systemidentifikation:

  Ziel der Systemidentifikation ist es, ein unbekanntes System mit Hilfe eines parallel geschalteten adaptiven Filters möglichst genau nachzubilden (Abb. 1.1). Dazu werden die Koeffizienten des adaptiven Filters fortlaufend an die Impulsantwort des unbekannten Systems angepasst. Durch Anwendung der Fourier-Transformation kann man somit die Übertragungsfunktion bzw. den Frequenzgang des unbekannten Systems messen.

  Typische Anwendungen sind z.B. seismologische Messungen oder auch die Echokompensation bei Freisprecheinrichtungen.

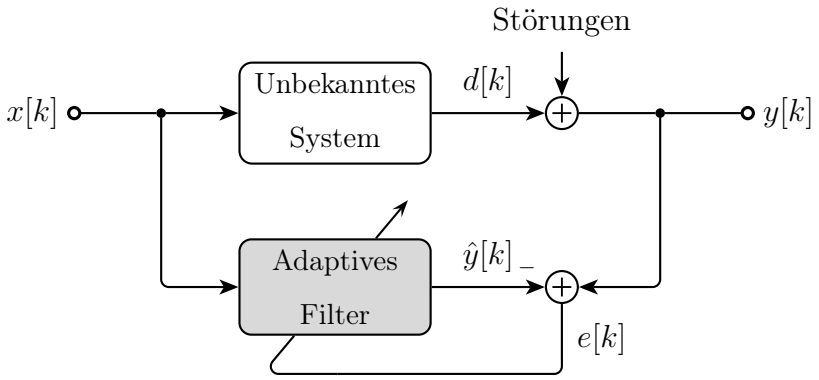

Abbildung 1.1: Systemidentifikation nach [2]

- Inverse Modellierung:

  Die inverse Modellierung findet beispielsweise Anwendung in der Übertragungstechnik zum Invertieren von Kanalverzerrungen und ist Bestandteil jedes Mobilfunksystems. Das unbekannte System besitzt normalerweise eine nicht minimalphasige Übertragungsfunktion $H(z)$, weshalb die inverse Übertragungsfunktion $1/H(z)$ im Allgemeinen nicht stabil ist. Falls man das Eingangssignal mittels eines Allpasses ausreichend verzögert, kann das (kausale) adaptive Filter eine verzögerte Version von $1/H(z)$ approximieren (Abb. 1.2).

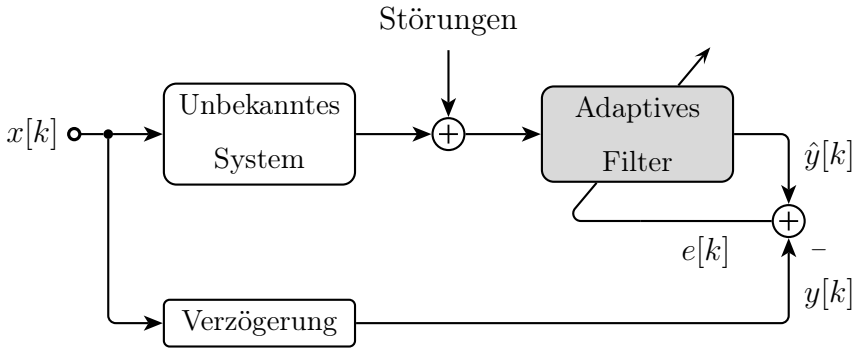

Abbildung 1.2: Inverse Modellierung nach [2]

- Prädiktion:

  Die Prädiktion (Abb. 1.3) zielt darauf ab, den aktuellen Wert eines unbekannten Signals $y[k]$ aus vergangenen Werten vorauszusagen. Das Restfehlersignal $e[k]$ stellt dabei den nicht vorhersagbaren, also den informationstragenden Teil dar. Bei Sprachcodern wäre dies das Ausgangssignal, welches anschließend quantisiert und zur Gegenseite übertragen wird. Anwendungsgebiete der Prädiktion sind z.B. LPC (*linear predictive coding*) oder ADPCM (*adaptive differential pulse code modulation*) bei der Sprach- und Audiocodierung.

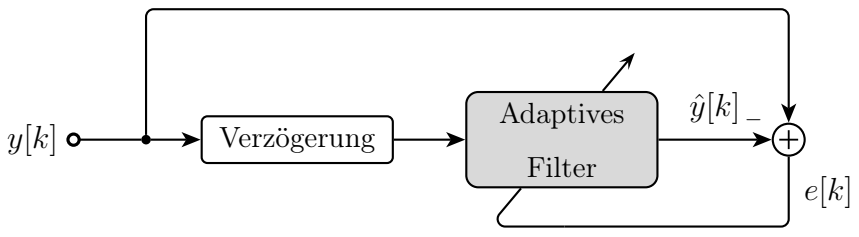

Abbildung 1.3: Prädiktion nach [2]

- Interferenz-Unterdrückung:

  Durch Interferenz-Unterdrückung kann ein im Eingangssignal enthaltener Störanteil eliminiert werden. Dazu wird ein Referenzsignal $x_{ref}[k]$ benö-

tigt, welches mit dem in $y[k]$ enthaltenen Störanteil auf lineare, aber sonst unbekannte Weise zusammenhängt, jedoch zum Nutzanteil in $y[k]$ unkorreliert ist. Da der Ausgang des adaptiven Filters $\hat{y}[k]$ nur mit $x[k]$ korrelierte Anteile enthalten kann, wird der Nutzanteil nicht gelöscht und bleibt im Resfehlersignal $e[k]$ enthalten. Anwendungen sind z.B. Störbefreiung von EKG-Messungen oder Interferenzunterdrückung bei Mikrofon-Arrays [2].

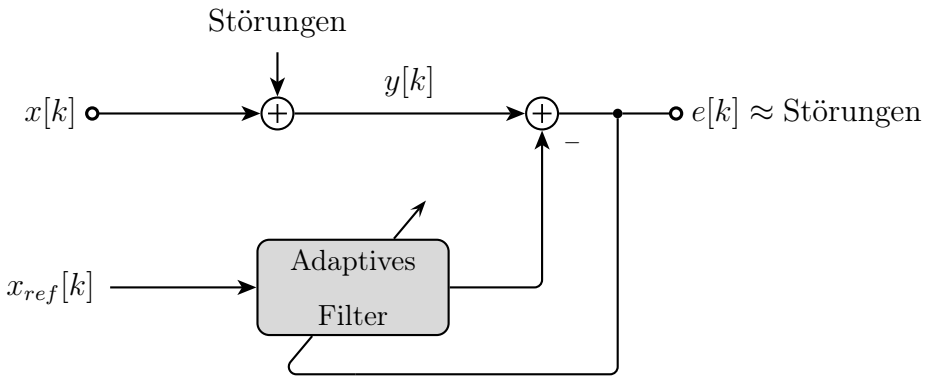

Abbildung 1.4: Interferenz-Unterdrückung nach [2]

1.2 Akustische Echokompensation

In den folgenden Kapiteln wird, aufgrund der Einfachheit und der im Gegensatz zu IIR-Filtern gesicherten Stabilität, ausschließlich der Fall linearer FIR-Filter und das spezielle Anwendungsgebiet der akustischen Echokompensation (Abb. 1.5) betrachtet.

Eine Echokompensation ist z.B. bei Freisprecheinrichtungen wünschenswert, da sich hier der ferne Sprecher $x[k]$ aufgrund der akustischen Kopplung zwischen Lautsprecher und Mikrofon auf der Gegenseite mit einer Verzögerung selbst hört. Falls keine Maßnahmen gegen diese Echos getroffen werden, kann eine sinnvolle Kommunikation zwischen den Gesprächspartnern unmöglich werden. Das Mikrofonsignal $y[k]$ setzt sich neben dem Echosignal $d[k]$ noch aus dem Sprachsignal

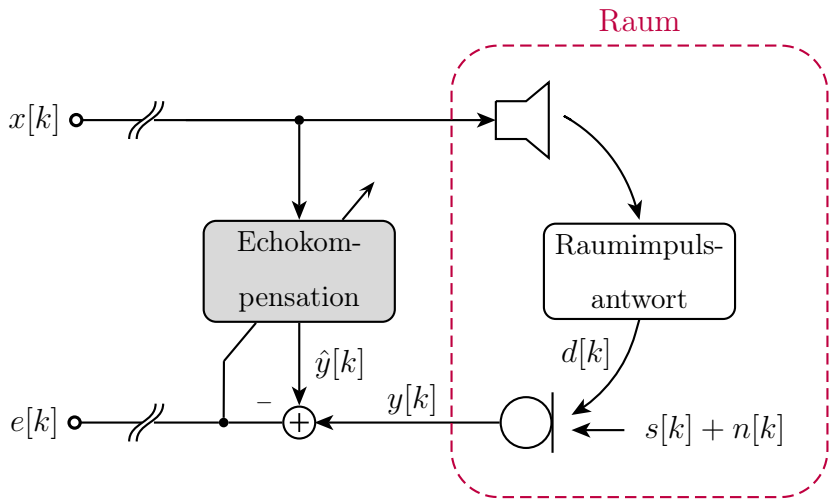

Abbildung 1.5: Die akustische Echokompensation

des lokalen Sprechers $s[k]$, sowie additiver Störungen $n[k]$ zusammen. Somit gilt:

$$y[k] = d[k] + s[k] + n[k], \tag{1.1}$$

wobei $s[k]$ und $n[k]$ das lokale Nutzsignal bilden. Für das Echosignal $d[k]$ gilt außerdem:

$$d[k] = x[k] * h[k], \tag{1.2}$$

wobei das Signal des fernen Sprechers $x[k]$ mit der Raumimpulsantwort $h[k]$ gefaltet wird. Die Raumimpulsantwort beschreibt die akustische Übertragungsstrecke zwischen Lautsprecher und Mikrofon.

Ziel der Echokompensation ist es nun, mit Hilfe eines adaptiven FIR-Filters die Raumimpulsantwort nachzubilden und das im Mikrofonsignal enthaltene Echosignal zu schätzen. Um die Echokomponenten aus dem Mikrofonsignal zu entfernen, wird dann einfach das geschätzte Echosignal vom Mikrofonsignal subtrahiert.

Kapitel 2

LMS-artige Algorithmen

Das Prinzip der adaptiven Filterung ist in allen gezeigten Fällen (Abb. 1.1 bis 1.4) aus der Einleitung gleich, jedoch wird die Herleitung der verschiedenen LMS-artigen Algorithmen aus Gründen der Übersichtlichkeit für das Problem der Systemidentifikation bzw. dem speziellen Anwendungsgebiet der AEC (*acoustic echo cancelation*, also der akustischen Echokompensation) durchgeführt.

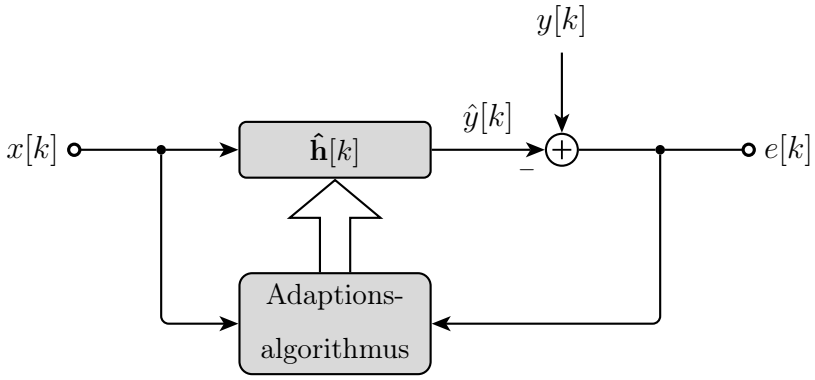

Abbildung 2.1: Grundlegende Struktur eines adaptiven Filters nach [2]

Den prinzipiellen Aufbau eines adaptiven Filters sieht man in Abb. 2.1. Dabei wird ein FIR-Filter der Länge L mit dem Koeffizientenvektor

$$\hat{\mathbf{h}}[k] = [h_0[k], h_1[k], \ldots, h_{L-1}[k]]^T \tag{2.1}$$

von einem Adaptionsalgorithmus eingestellt wird. Die Aufgabe des adaptiven

Filters ist hierbei, das Referenzsignal $x[k]$ so zu filtern, dass sein Ausgangssignal

$$\hat{y}[k] = \sum_{i=0}^{L-1} \hat{h}_i[k]x[k-i] = \mathbf{x}^T[k]\hat{\mathbf{h}}[k] \tag{2.2}$$

idealerweise dem gewünschten Signal $y[k]$ entspricht. Damit sich die Faltung in Vektorschreibweise nach Gl. 2.2 kompakter formulieren lässt, werden L Abtastwerte des Eingangssignals $x[k]$ zusammengefasst:

$$\mathbf{x}[k] = [x[k], x[k-1], \ldots, x[k-L+1]]^T. \tag{2.3}$$

Daraus folgt, dass es die Aufgabe des Adaptionsalgorithmus sein muss, das Restfehlersignal

$$e[k] = y[k] - \hat{y}[k] = y[k] - \mathbf{x}^T[k]\hat{\mathbf{h}}[k] \tag{2.4}$$

bezüglich eines geeigneten Maßes zu minimieren.

Die meisten bekannten Adaptionsalgorithmen versuchen den MSE (*mean squared error*), also das mittlere Fehlerquadrat $E\{e^2[k]\}$ zu minimieren. Dadurch erhält man im Idealfall die Wiener-Lösung $\hat{\mathbf{h}}_{opt}[k]$. Ein direktes Minimieren des MSE führt laut [2] auf folgendes Gleichungssystem:

$$\hat{\mathbf{h}}_{opt} = {\mathbf{R}_{\mathbf{xx}}}^{-1} \cdot \mathbf{r}_{y\mathbf{x}}, \tag{2.5}$$

wobei Gl. 2.5 als sogenannte Wiener-Hopf Gleichung bezeichnet wird. Dabei entspricht

$$\mathbf{R_{xx}} = \begin{bmatrix} R_{xx}[0] & R_{xx}[1] & \ldots & R_{xx}[L-1] \\ R_{xx}[-1] & R_{xx}[0] & \ldots & R_{xx}[L-2] \\ \vdots & \vdots & \ddots & \vdots \\ R_{xx}[-L+1] & R_{xx}[-L+2] & \ldots & R_{xx}[0] \end{bmatrix}, \tag{2.6}$$

der Autokorrelationsmatrix von $\mathbf{x}[k]$ und

$$\mathbf{r}_{y\mathbf{x}} = [R_{yx}[0], R_{yx}[1], \ldots, R_{yx}[L-1]]^T \tag{2.7}$$

dem Kreuzkorrelationsvektor. Eine anschauliche Herleitung findet man in [9].

Stellt man nun den MSE in Abhängigkeit der Filterkoeffizienten dar, so erhält man für eine Erregung des Systems mit weißem Rauschen die in Abb. 2.2 gezeigte typische Fehleroberfläche. Bei FIR-Filtern hat man immer eine parabolische Form und dementsprechend nur ein Minumum, was bei den nachfolgenden Algorithmen von Bedeutung ist, da dieses Minimum die optimale, d.h. die Wiener-Lösung beschreibt. [2]

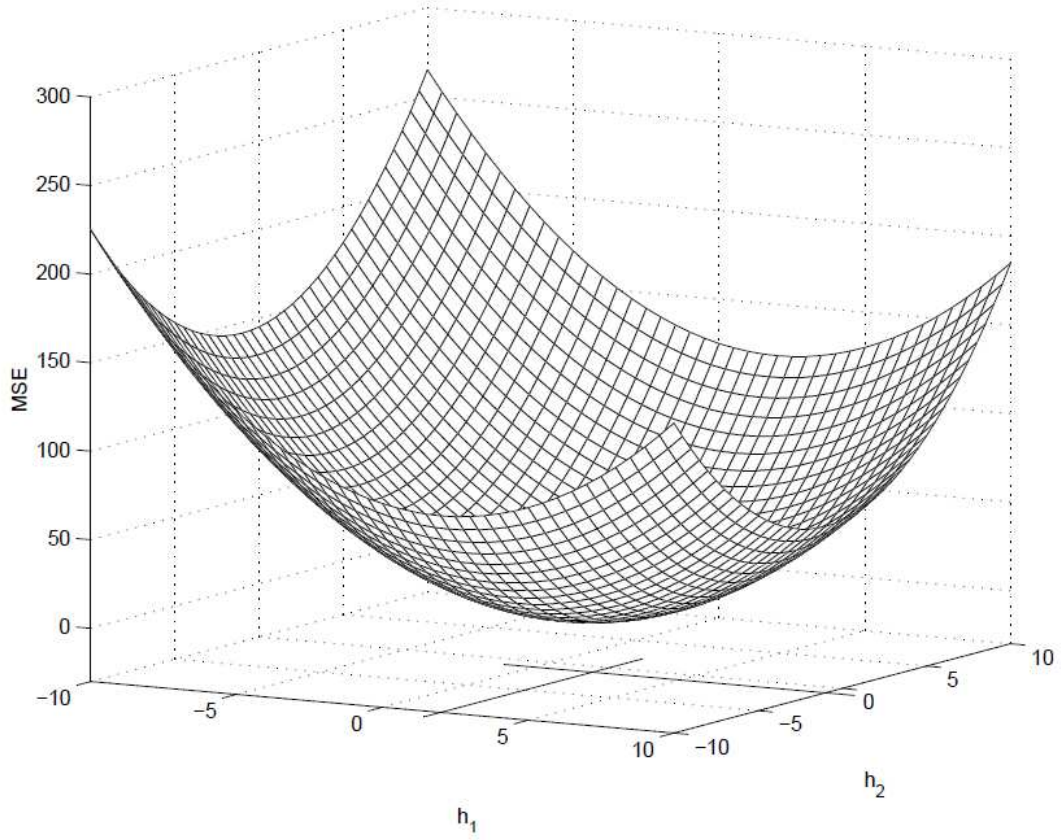

Abbildung 2.2: Fehleroberfläche für ein adaptives FIR-Filter mit zwei Koeffizienten aus [2]

Eine Lösung der Wiener-Hopf Gleichung ist mit dem Gradientenverfahren (*Me-*

thode des steilsten Abstiegs) möglich, das auf der Idee beruht, die Koeffizienten $\hat{\mathbf{h}}[k]$ in jedem Abtasttakt entgegen der Richtung des Gradienten

$$\nabla E\{e^2[k]\} = \frac{\partial E\{e^2[k]\}}{\partial \hat{\mathbf{h}}[k]} \tag{2.8}$$

ein Stück zu verändern. Diese Änderung erfolgt gemäß

$$\hat{\mathbf{h}}[k+1] = \hat{\mathbf{h}}[k] - \frac{\mu}{2}\nabla E\{e^2[k]\}, \tag{2.9}$$

mit der Schrittweite μ. Diese beeinflusst direkt, wie schnell das adaptive Filter konvergiert. Bei einer kleinen Schrittweite ändern sich die Koeffizienten bei jedem k nur geringfügig, während das Filter bei größerer Schrittweite schneller konvergiert. Wenn die Schrittweite jedoch zu groß gewählt wurde, ändern sich die Filterkoeffizienten zu schnell und das Filter divergiert. Laut [7] konvergiert der MSE bei Erregung mit stationärem weißen Rauschen und verschiedener Unabhängigkeitsannahmen gegen einen bestimmten Wert, wenn für die Schrittweite

$$0 < \mu < \frac{2}{\|\mathbf{x}[k]\|^2} \tag{2.10}$$

gilt. $\|.\|^2$ beschreibt die L_2-Norm und ist definiert als $\|\mathbf{x}[k]\|^2 = \mathbf{x}^T[k]\mathbf{x}^[k]$. Solange $y[k]$ keine Störungen enthält erreicht man die schnellste Konvergenz für $\mu = 1/\|\mathbf{x}[k]\|^2$.

Beim Gradientenverfahren schreitet man demnach auf der Fehleroberfläche von einem gewissen Näherungswert aus in Richtung des negativen Gradienten, also in Richtung des steilsten Abstiegs, bis man das Minimum erreicht hat bzw. keine numerische Verbesserung mehr erzielt wird.

Die Vorteile gegenüber eine direkten Berechnung nach Gl. 2.5 sind ein geringerer Rechenaufwand und die Möglichkeit der Zeitvarianz eines unbekannten Systems zu folgen, da bei jedem k erneut die optimale Impulsantwort geschätzt wird.

2.1 LMS-Algorithmus

Wie aus Gl. 2.9 zu erkennen ist, basiert die Methode des steilsten Abstiegs allerdings auf den echten Erwartungswerten $E\{e^2[k]\}$ und daher auch $\mathbf{R_{xx}}[k]$ und $\mathbf{r}_{y\mathbf{x}}[k]$, die in der Praxis jedoch nicht bekannt sind. Eine Schätzung dieser beiden Größen ist sehr aufwändig, weshalb zur Vereinfachung der Erwartungswert $E\{e^2[k]\}$ durch $e^2[k]$, also den aktuell vorliegenden quadratischen Restfehler, ersetzt wird. Nach [2] erreicht man die benötigte Mittelung mit einer entsprechend kleinen Schrittweite μ und erhält als Gradient

$$\nabla e^2[k] = \frac{\partial e^2[k]}{\partial \hat{\mathbf{h}}[k]} = 2e[k]\frac{\partial e[k]}{\partial \hat{\mathbf{h}}[k]} = 2e[k]\frac{\partial (y[k] - \hat{\mathbf{h}}^T[k]\mathbf{x}[k])}{\partial \hat{\mathbf{h}}[k]} = -2e[k]\mathbf{x}[k]. \quad (2.11)$$

In Gl. 2.9 eingesetzt erhält man somit den Adaptionsschritt des LMS-Algorithmus (*least mean square*):

$$\hat{\mathbf{h}}[k+1] = \hat{\mathbf{h}}[k] + \mu\mathbf{x}[k]e[k]. \quad (2.12)$$

Der LMS-Algorithmus geht auf Widrow und Hoff [13] zurück und bildet wegen seiner numerischen Stabilität, seiner niedrigen Komplexität und seiner Robustheit die Grundlage der Klasse der LMS-artigen Algorithmen, die in der Praxis am häufigsten eingesetzt werden. Die Adaption eines FIR-Filters mittels des LMS Algorithmus ist in Tab. 2.1 zusammengefasst.

für jedes k:	
1. FIR-Filterung:	$\hat{y}[k] = \hat{\mathbf{h}}^T[k]\mathbf{x}[k]$
2. Fehlersignal berechnen:	$e[k] = y[k] - \hat{y}[k]$
3. Adaption:	$\hat{\mathbf{h}}[k+1] = \hat{\mathbf{h}}[k] + \mu\mathbf{x}[k]e[k]$

Tabelle 2.1: Adaption mittels des LMS-Algorithmus nach [2]

In Quellcode A.1 in Anhang A sieht man eine Möglichkeit, wie man den LMS-Algorithmus in Matlab implementieren kann. Die Funktion `lms.m` gibt das Fehlersignal $e[k]$, eine Matrix mit den geschätzen Impulsantworten $\hat{h}[k]$ des unbekannten

Systems und das geschätze Ausgangssignal $\hat{y}[k]$ zurück. Aufgerufen wird der Algorithmus durch Übergabe des Eingangssignals $x[k]$, des gewünschten Signals (in diesem Fall $y[k]$), der Schrittweite μ und der Filterlänge L.

Der erste Block im Code ab Zeile 4 dient der Initialisierung und eventueller Rücksetzung vorhandener Variablen. Danach folgt der Start des Algorithmus. Dabei wird eine for-Schleife der Länge des Eingangssignals durchlaufen und für jedes k wird in Zeile 13 bis 14 durch `X = [x(i);X(1:end-1)];` und `ys(i) = hs' * X` zuerst eine FIR-Filterung vorgenommen und anschließend das Fehlersignal $e[k]$ (Zeile 15) berechnet. Die abschließende Adaption erfolg in Zeile 16 durch `hs = hs + mu * e(i)* X` und endet in der nachfolgenden Zeile mit der Speicherung der aktuellen geschätzen Impulsantwort $\hat{h}[k]$ in eine Matrix. Dadurch hat man anschließend Zugriff auf alle geschätzten Impulsantworten und kann somit z.B. den Systemabstand berechnen (vgl. Kap. 3.1.1).

2.2 Normalized LMS-Algorithmus (NLMS)

Der NLMS-Algorithmus ist eine modifizierte Version des LMS-Algorithmus. Das Konvergenzkriterium aus Gl. 2.10 legt nahe, die Schrittweite μ auf $1/\mathbf{x}^T[k]\mathbf{x}[k]$ zu normieren. Diese Schrittweite hilft v.a. für eine einfache Wahl stabiler Schrittweiten. Dabei ändert sich die Update-Gleichung des LMS zu

$$\hat{\mathbf{h}}[k+1] = \hat{\mathbf{h}}[k] + \frac{\mu_n e[k]\mathbf{x}[k]}{\mathbf{x}^T[k]\mathbf{x}[k] + \delta_{NLMS}}, \tag{2.13}$$

dem Adaptionsschritt des NLMS-Algorithmus mit der normierten Schrittweite $\mu_n = \mu\mathbf{x}^T[k]\mathbf{x}[k]$. Die Konstante δ_{NLMS} ist eine sehr kleine Zahl, die dazu dient, eine Division durch Null zu verhindern, falls $\mathbf{x}^T[k]\mathbf{x}[k] \approx 0$, was z.B. in Sprachpausen bei Sprachsignalen der Fall ist. Bei der Implementierung des NLMS Algorithmus in Matlab ändert sich im Gegensatz zu dem LMS, wie schon angedeutet,

nur der dritte Schritt aus Tab. 2.1. Im Quellcode A.2 sieht man ein Implementierungsbeispiel und in Tab. 2.2 sind die einzelnen Schritte wieder zusammengefasst.

für jedes k:	
1. FIR-Filterung:	$\hat{y}[k] = \hat{\mathbf{h}}^T[k]\mathbf{x}[k]$
2. Fehlersignal berechnen:	$e[k] = y[k] - \hat{y}[k]$
3. Adaption:	$\hat{\mathbf{h}}[k+1] = \hat{\mathbf{h}}[k] + \frac{\mu_n e[k]\mathbf{x}[k]}{\mathbf{x}^{\mathrm{T}}[k]\mathbf{x}[k] + \delta_{NLMS}}$

Tabelle 2.2: Adaption mittels des NLMS-Algorithmus nach [2]

2.3 Proportionate NLMS-Algorithmus (PNLMS)

Eine Variante des normierten LMS-Algorithmus ist der PNLMS-Algorithmus, welcher auf Duttweiler [3] zurückgeht. Die Idee hinter diesem Algorithmus ist es, jedem Filterkoeffizienten eine individuelle Schrittweite zuzuweisen, da in vielen Anwendungen nur wenige signifikante Koeffizienten vorhanden sind, die dann aber schneller konvergieren können, da die zur Verfügung stehende „Adaptionsenergie“ nur auf diese verteilt wird. Diese Schrittweiten werden aus der letzten Schätzung der Filterkoeffizienten auf solche Weise berechnet, dass größere Koeffizienten eine größere Schrittweite erhalten und so die Konvergenz dieses Koeffizienten erhöht wird. Das hat den Effekt, dass „aktive“ Koeffizienten schneller adaptiert werden, als „nichtaktive“, also sehr kleine Koeffizienten.

Wie später durch Simulationen gezeigt wird (Kap. 3), konvergiert der PNLMS für deltaimpulsähnliche Impulsantworten, also Impulsantworten, bei denen nur wenige deutliche Regionen an Koeffizienten signifikant sind, viel schneller als der NLMS. Der PNLMS Algorithmus besitzt die Update-Gleichung

$$\hat{\mathbf{h}}[k+1] = \hat{\mathbf{h}}[k] + \frac{\mu_n \mathbf{G}[k]\mathbf{x}[k]e[k]}{\mathbf{x}^{\mathrm{T}}[k]\mathbf{G}[k]\mathbf{x}[k] + \delta_{PNLMS}}, \tag{2.14}$$

wobei

$$\mathbf{G}[k] = \mathrm{diag}(g_0[k], \ldots, g_{L-1}[k]) \tag{2.15}$$

eine Diagonalmatrix darstellt, die die Schrittweiten der individuellen Koeffizienten des Filters einstellt. μ_n ist weiterhin die normierte „globale“ Schrittweite und δ_{PNLMS} der Regularisierungsfaktor des PNLMS. Dieser wird gewöhnlich zu

$$\delta_{PNLMS} = \frac{\delta_{NLMS}}{L} \tag{2.16}$$

gewählt [4]. Die diagonalen Elemente von $\mathbf{G}[k]$ werden wie folgt berechnet:

$$g_l[k] = \frac{\gamma_l[k]}{\sum_{i=0}^{L-1} \gamma_i[k]}, \tag{2.17}$$

wobei $0 \leq l \leq L-1$ die einzelnen Koeffizienten adressiert und

$$\gamma_l[k] = \max\left\{\rho \max\left[\delta_p, \left|\hat{h}_0[k]\right|, \ldots, \left|\hat{h}_{L-1}[k]\right|\right], \left|\hat{h}_l[k]\right|\right\}. \tag{2.18}$$

Der Parameter δ_p reguliert die Anpassung, falls alle Koeffizienten Null wären, was z.B. bei der Initialisierung der Fall ist. Ein typischer Wert ist $\delta_p = 0.01$. Der Parameter ρ hingegen soll eine kontinuierliche Adaption auch für sehr kleine Koeffizienten sicherstellen und wird üblicherweise zu $\rho = 5/L$ gewählt [4]. In Tab. 2.3 findet man nochmal eine Zusammenfassung des PNLMS Algorithmus und in Quellcode A.3 den zugehörigen Matlab Code.

Wie in den späteren Simulationen gezeigt wird, konvergiert der PNLMS für deltaimpulsähnliche Impulsantworten schneller als der herkömmliche NLMS. Bei dispersiven, also rauschähnlichen Impulsantworten ist der PNLMS schlechter als der NLMS. Um dieses Problem zu beheben wurde der PNLMS++ Algorithmus eingeführt [4]. Dabei wird für ungerade Schritte k die Update-Gleichung des PNLMS (Gl. 2.14) verwendet und für gerade Schritte die Diagonalmatrix $\mathbf{G}[k]$ gleich der Einheitsmatrix

$$\mathbf{G}[k] = \mathbf{I} \tag{2.19}$$

für jedes k:	
1. FIR-Filterung:	$\hat{y}[k] = \hat{\mathbf{h}}^T[k]\mathbf{x}[k]$
2. Fehlersignal berechnen:	$e[k] = y[k] - \hat{y}[k]$
3. Adaption:	$\hat{\mathbf{h}}[k+1] = \hat{\mathbf{h}}[k] + \frac{\mu_n e[k]\mathbf{x}[k]}{\mathbf{x}^{\mathrm{T}}[k]\mathbf{x}[k] + \delta_{PNLMS}}$
mit	$\mathbf{G}[k] = \mathrm{diag}(g_0[k], \ldots, g_{L-1}[k])$
und	$g_l[k] = \frac{\gamma_l[k]}{\sum_{i=0}^{L-1}\gamma_i[k]}$
und	$\gamma_l[k] = \max\left\{\rho \max\left[\delta_p, \left\|\hat{h}_0[k]\right\|, \ldots, \left\|\hat{h}_{L-1}[k]\right\|\right], \left\|\hat{h}_l[k]\right\|\right\}$

Tabelle 2.3: Adaption mittels des PNLMS-Algorithmus

gesetzt, d.h. man erhält die Update-Gleichung des NLMS (Gl. 2.13). Ein Wechsel zwischen diesen beiden Algorithmen hat den Effekt, dass man bei nicht-deltaimpulsähnlichen Impulsantworten nicht viel schlechter als der NLMS ist. Der PNLMS++ Algorithmus ist aber keine zufriedenstellende Lösung, da er nicht die komplette Struktur der geschätzten Impulsantwort ausnutzt.

2.4 Improved PNLMS-Algorithmus (IPNLMS)

In diesem Abschnitt wird der IPNLMS Algorithmus von Benesty [4] behandelt. Er basiert auf dem PNLMS und hat das Ziel, eine bessere Robustheit zu erreichen, falls keine deltaimpulsähnliche, sondern eine dispersive Impulsantwort vorliegt. Die Tatsache, dass der PNLMS bei dispersiven Impulsantworten langsamer als der NLMS ist, bedeutet, dass die Gl. 2.18 verändert werden muss. Falls die Schätzung der Koeffizienten nicht genau ist, kann die Wahl des Maximums zwischen $\left|\hat{h}_l\right|$ und einer anderen positiven Zahl in Gl. 2.18 eine katastrophale Auswirkung auf die Konvergenz haben. Beim IPNLMS fällt diese Wahl nicht so extrem aus und man erhält nach kurzer Rechnung (vgl. [4]) die zugehörige Update-Gleichung

$$\hat{\mathbf{h}}[k+1] = \hat{\mathbf{h}}[k] + \frac{\mu_n \mathbf{K}[k]\mathbf{x}[k]e[k]}{\mathbf{x}^{\mathrm{T}}[k]\mathbf{K}[k]\mathbf{x}[k] + \delta_{IPNLMS}}, \tag{2.20}$$

wobei

$$\mathbf{K}[k] = \mathrm{diag}(\kappa_0[k], \ldots, \kappa_{L-1}[k]) \tag{2.21}$$

wieder eine Diagonalmatrix mit den Elementen

$$\kappa_l[k] = \frac{1-\alpha}{2L} + (1+\alpha)\frac{\left|\hat{h}_l[k]\right|}{2\left\|\hat{\mathbf{h}}[k]\right\|_1 + \epsilon} \tag{2.22}$$

ist. Der Parameter ϵ ist eine kleine positive Zahl und wird in der Praxis benötigt um eine Division durch Null zu verhindern, falls $\left\|\hat{\mathbf{h}}[k]\right\|_1 \approx 0$. Der Regularisierungsfaktor des IPNLMS sollte zu

$$\delta_{IPNLMS} = \frac{1-\alpha}{2L}\delta_{NLMS} \tag{2.23}$$

gewählt werden und α aus folgendem Intervall

$$-1 \leq \alpha < 1. \quad [4] \tag{2.24}$$

Falls man $\alpha = -1$ wählt, kann man leicht sehen, dass der IPNLMS dem NLMS entspricht. Für α nahe 1 verhält sich der IPNLMS wie der PNLMS. Eine Zusammenfassung des IPNLMS Algorithmus findet man Tab. 2.4 und im Quellcode A.4 ein Implementierungsbeispiel.

für jedes k:	
1. FIR-Filterung:	$\hat{y}[k] = \hat{\mathbf{h}}^T[k]\mathbf{x}[k]$
2. Fehlersignal berechnen:	$e[k] = y[k] - \hat{y}[k]$
3. Adaption:	$\hat{\mathbf{h}}[k+1] = \hat{\mathbf{h}}[k] + \frac{\mu_n \mathbf{K}[k]\mathbf{x}[k]e[k]}{\mathbf{x}^{\mathrm{T}}[k]\mathbf{K}[k]\mathbf{x}[k] + \delta_{IPNLMS}}$
mit	$\mathbf{K}[k] = \mathrm{diag}(\kappa_0[k], \ldots, \kappa_{L-1}[k])$
und	$\kappa_l[k] = \frac{1-\alpha}{2L} + (1+\alpha)\frac{\left\|\hat{h}_l[k]\right\|}{2\left\|\hat{\mathbf{h}}[k]\right\|_1 + \epsilon}$

Tabelle 2.4: Adaption mittels des IPNLMS-Algorithmus

2.5 Weitere LMS-artige Algorithmen

Neben den in diesem Kapitel beschriebenen Verfahren existiert noch eine Vielzahl weiterer auf dem LMS-Algorithmus aufbauender adaptiver Filtermethoden.

So gibt es zum Beispiel Methoden, die weitere Vereinfachungen im Adaptionsalgorithmus vornehmen. Da wären beispielsweise der *quantized error LMS* oder der *sign LMS* anzuführen. Der *sliding window LMS* wiederum nutzt die Erwartungswerte der Gradienten. Und wieder andere, wie z.B. der *FFT-LMS* oder der *DCT-LMS*, nehmen im Transformationsbereich eine Dekorrelation und Koeffizientenadaptierung vor, um somit eine bessere Konvergenz zu erreichen [11].

Einen breiten Überblick über diese und weitere Verfahren findet man z.B. in [7]. Im nächsten Kapitel werden die drei Algorithmen NLMS, PNLMS und IP-NLMS bezüglich ihres Verhaltens bei unterschiedlichen Raumimpulsantworten verglichen.

Kapitel 3

Vergleich der Algorithmen

Für einen objektiven Vergleich unterschiedlicher Adaptionsalgorithmen bzw. unterschiedlicher Parametereinstellungen bieten sich zwei Kriterien an, die sich auf die Güte der Systemidentifikation und auf die Höhe der Echodämpfung beziehen. Deshalb werden in diesem Kapitel zuerst diese Bewertungskriterien eingeführt und anschließend im Abschnitt 3.2 ein Vergleich der in Kapitel 2 vorgestellten Algorithmen durchgeführt.

3.1 Bewertungskriterien

3.1.1 Systemabstand

Der Systemabstand ist ein Maß für die Güte der Systemidentifikation und definiert als

$$D[k] = \frac{\left\| \hat{\mathbf{h}}[k] - \mathbf{h}[k] \right\|^2}{\|\mathbf{h}[k]\|^2}. \tag{3.1}$$

Dabei bezeichnet $\|\mathbf{h}[k]\|^2 = \mathbf{h}[k]^T \mathbf{h}[k]$ die quadrierte Vektornorm. Üblicherweise wird für den Systemabstand der logarithmische Wert, also $10 \log D[k]$ in dB angegeben. Die Impulsantworten sind jedoch bei realen System in der Regel unbekannt, weshalb der Systemabstand $D[k]$ in erster Linie eine wichtige Beurtei-

lungsgröße für Simulationen, bei vorgegebener Impulsantwort $\mathbf{h}[k]$ ist.

3.1.2 Echodämpfung

Ein weiteres Bewertungskriterium ist die sogenannte Echodämpfung, also die erzielbare Reduktion der Leistung des Echosignals $y[k]$, einschließlich eventueller Störungen. Es ist mit dem subjektiven Höreindruck korreliert und wird in der Literatur mit dem Begriff ERLE (*echo return loss enhancement*) bezeichnet. Dieser ist definiert als

$$\mathrm{ERLE}[k] = \frac{E\{y^2[k]\}}{E\{e^2[k]\}}. \tag{3.2}$$

Auch dieses Maß wird in der Regel in dB angegeben ($10 \log \mathrm{ERLE}\,[k]$).

Eine Variante des ERLE Maßes ist der ERLE_{true} Wert. Dieser Wert ist aber, ebenso wie der Systemabstand, auf Simulationen beschränkt, da man laut Definition

$$\mathrm{ERLE}_{true}\,[k] = \frac{E\{d^2[k]\}}{E\{(e[k] - n[k] - s[k])^2\}} \tag{3.3}$$

das Signal $d[k]$ benötigt. In der Praxis stehen jedoch, gemäß Abb. 1.5, nur das Lautsprechersignal $x[k]$ und das Mikrofonsignal $y[k]$ direkt zur Verfügung.

3.2 Vergleich der Algorithmen

In diesem Abschnitt werden der NLMS, PNLMS und IPNLMS bezüglich des Systemabstandes miteinander verglichen.

Als Eingangssignal wurde einmal weißes Gaußsches Rauschen mit der Länge 25000 Samples und einmal ein Sprachsignal mit der gleichen Länge verwendet. Wie in Abb. 3.1 gezeigt, wurden zwei verschiedene Impulsantworten $\mathbf{h}[k]$ genutzt. Eine davon ist deltaimpulsähnlich und die andere dispersiv; beide haben die Länge 500 Samples. Die Länge des adaptiven Filters $\hat{\mathbf{h}}[k]$ ist $L = 500$.

Außerdem gelten für die Parameter der Algorithmen folgende Einstellungen:

- SNR = 40 dB
- $\mu_n = 0.2$
- $\alpha = 0$
- $\rho = \frac{5}{L} = 0.01$
- $\delta_{NLMS} = 0.001$
- $\delta_{PNLMS} = \frac{\delta_{NLMS}}{L}$
- $\delta_{IPNLMS} = \frac{\delta_{NLMS}}{L}$

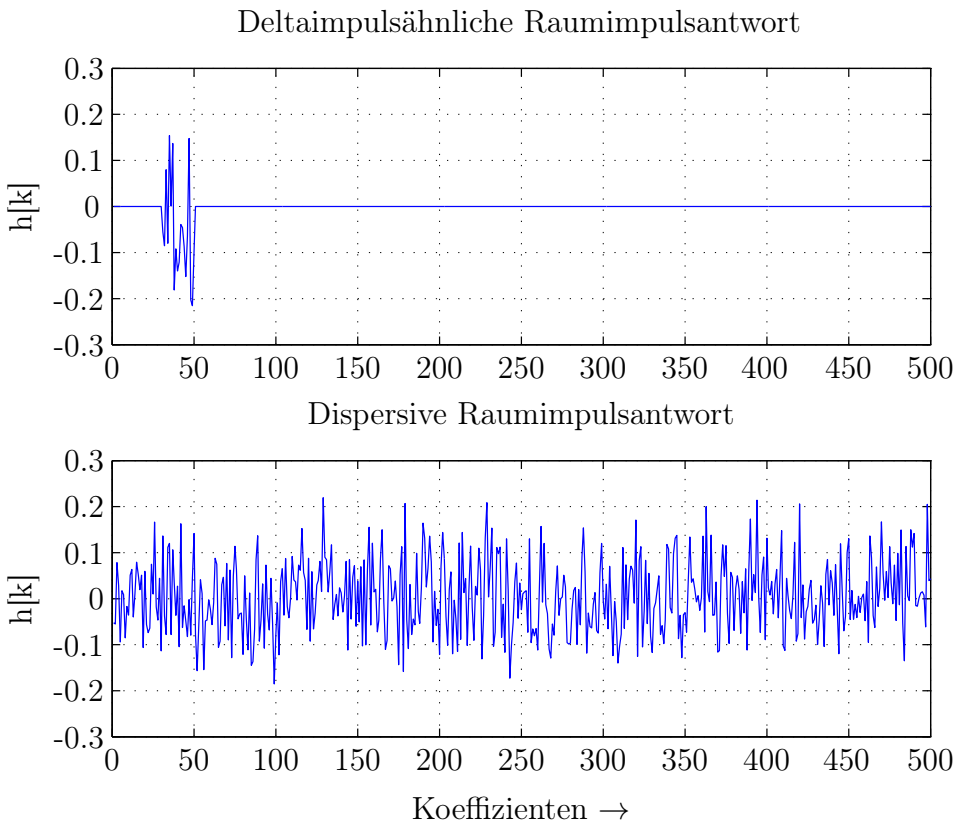

Abbildung 3.1: Zwei verschiedene Impulsantworten

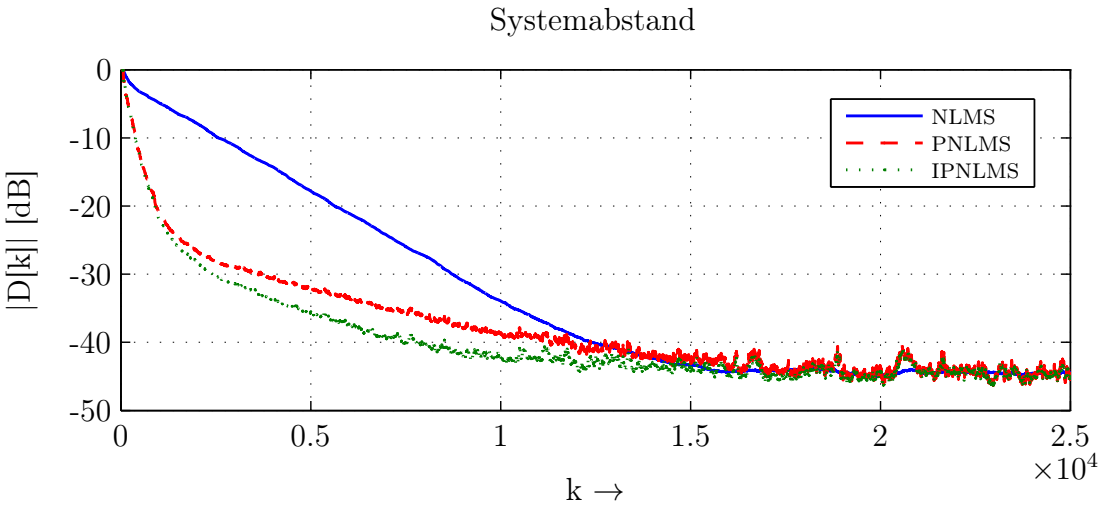

Abbildung 3.2: Systemabstände für weißes Rauschen bei einer deltaimpulsähnlichen Impulsantwort

Zuerst wurden die Systemabstände der drei Algorithmen bei weißem Rauschen als Eingangssignal und einer deltaimpulsähnlichen Impulsantwort verglichen (Abb. 3.2). Man sieht deutlich, dass der PNLMS und der IPNLMS viel schneller als der NLMS konvergieren. Am Anfang sind der PNLMS und der IPNLMS noch gleich schnell, jedoch ist der IPNLMS danach etwas schneller, was auf die gleichbleibendere Konvergenz zurückzuführen ist, da bei diesem Algorithmus auch die (sehr) kleinen Koeffizienten zu jedem Zeitpunkt mitjustiert werden.

Abb. 3.3 zeigt das gleiche Szenario, nur dass in diesem Fall eine dispersive Impulsantwort gewählt wurde, bei der sich signifikante Koeffizienten über die gesamte Länge verteilen (Abb. 3.1 (unten)). Während der NLMS und der IPNLMS noch recht gut funktionieren, verschlechtert sich der PNLMS deutlich.

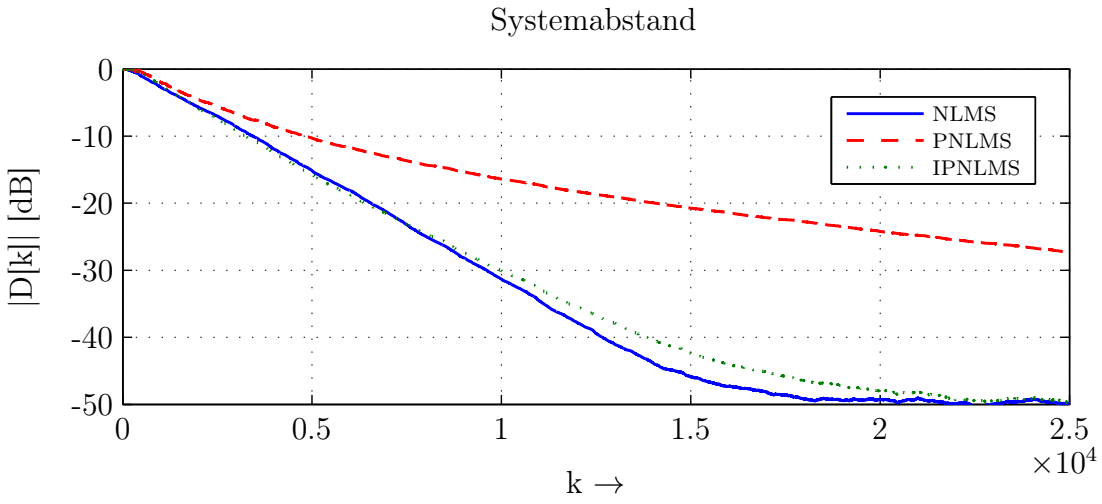

Abbildung 3.3: Systemabstände für weißes Rauschen bei einer dispersiven Impulsantwort

Als nächstes wurde die Folgefähigkeit der Algorithmen getestet, die zur Beurteilung zeitveränderlicher Systeme dienen soll. Es wird wieder die deltaimpulsähnliche Impulsantwort (Abb. 3.1 (oben)) gewählt und bei 16000 Samples ein Wechsel der Raumimpulsantwort simuliert. Dazu wird die Impulsantwort an dieser Stelle um 12 Samples nach rechts verschoben, was als Simulation eines vergrößerten Lautsprecher-Mikrofon-Abstandes interpretiert werden kann. Die anderen Einstellungen bleiben wie zuvor. In Abb. 3.4 sieht man, dass der PNLMS und der IPNLMS eine deutlich bessere Folgefähigkeit besitzen als der NLMS. Man kann beobachten, dass der NLMS beim Systemwechsel weniger stark einbricht, als die anderen beiden Algorithmen. Das ist vermutlich darauf zurückzuführen, dass er, nachdem sich die Koeffizienten durch die Verschiebung komplett geändert haben, die großen Koeffizienten weniger gut identifiziert als der PNLMS bzw. IPNLMS. Diese haben zunächst einen kleinen Vorteil, können danach aber die kleinen Koeffizienten weniger schnell verändern als der NLMS.

Als letztes wurden die drei Algorithmen nochmal mit einem Sprachsignal der Länge 25000 Samples und den beiden verschiedenen Impulsantworten getestet.

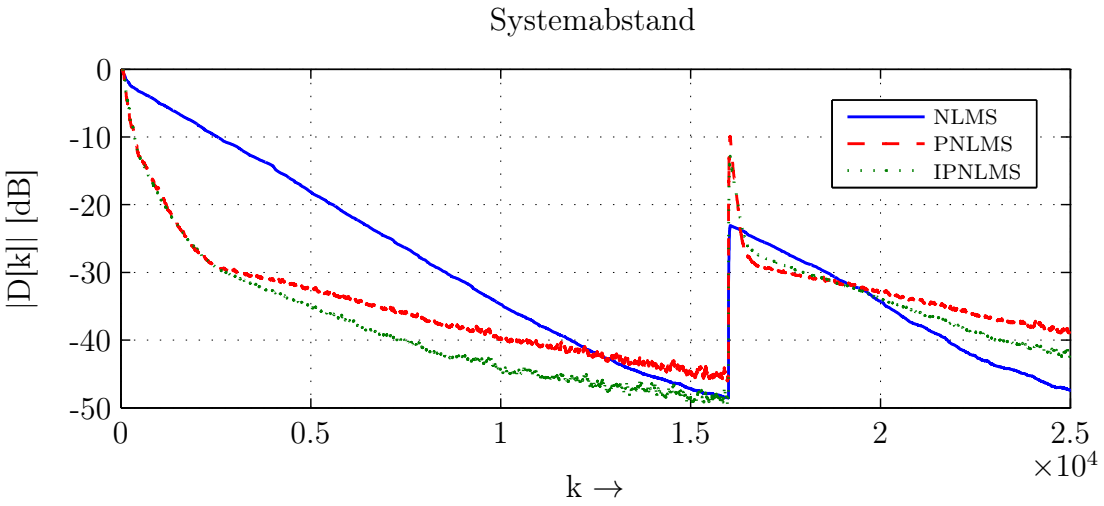

Abbildung 3.4: Systemabstände für weißes Rauschen, deltaimpulsähnlicher Impulsantwort und einem Systemwechsel bei 16000 Samples

In Abb. 3.5 und 3.6 erkennt man die gleichen Verhaltensweisen wie bei weißem Rauschen als Eingangssignal. Die Divergenz zu Beginn, also in dem Bereich, in dem die Kurve über 0 dB liegt, kommt daher, dass kein VAD (*voice activity detection*) verwendet wurde. Deshalb arbeitet die Adaption erstmal in die falsche Richtung.

3.3 Schlussfolgerung

Wie man den vorherigen Simulationen entnehmen kann, bereitet der PNLMS bei deltaimpulsähnlichen Impulsantworten keine Probleme und hat im Vergleich zum NLMS eine sehr schnelle Konvergenzrate. Leider gilt das nicht für dispersive Impulsantworten. Der IPNLMS hingegen verhält sich bei deltaimpulsähnlichen Impulsantworten wie der PNLMS, hat also genauso gute Konvergenzeigenschaften, funktioniert jedoch bei dispersiven Impulsantworten besser als der PNLMS. Der IPNLMS verbindet also die guten Eigenschaften des NLMS und PNLMS und verschlechtert sich nicht bei jeweils „unpassenden“ Impulsantworten.

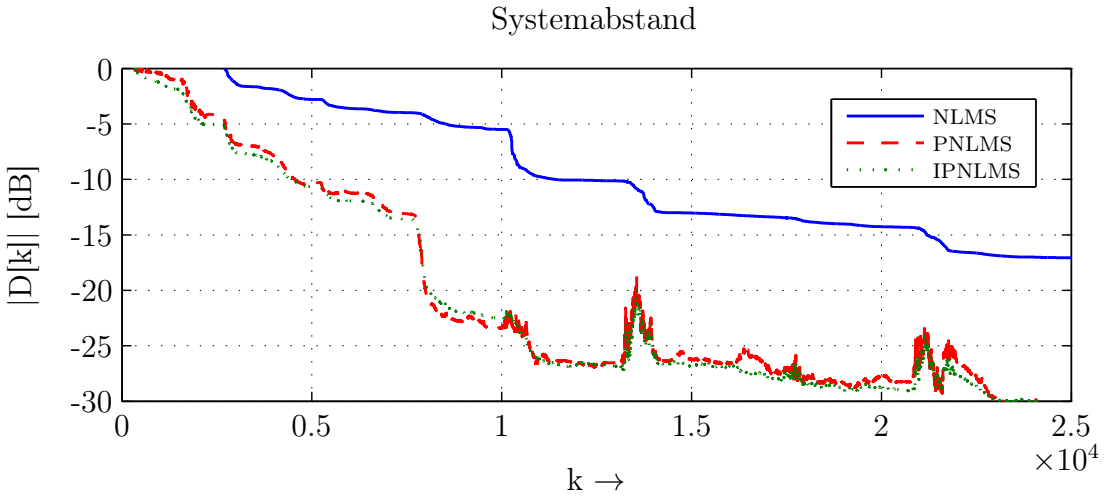

Abbildung 3.5: Systemabstände für Sprache bei einer deltaimpulsähnlichen Impulsantwort

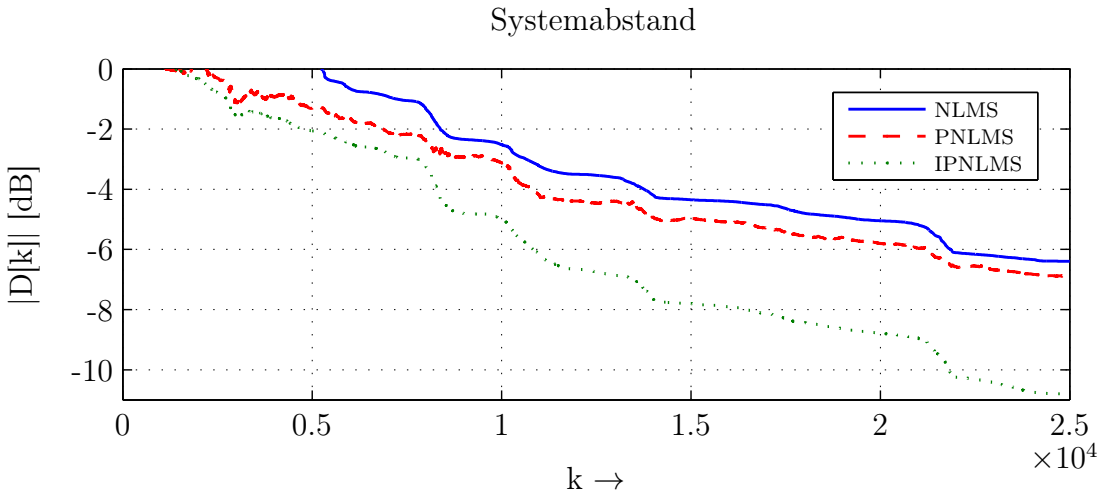

Abbildung 3.6: Systemabstände für Sprache bei einer dispersiven Impulsantwort

Kapitel 4

Dokumentation der GUI

In diesem letzten Kapitel wird eine Einführung in den Funktionsumfang der Matlab-GUI und eine Übersicht über sämtliche Einstellmöglichkeiten gegeben.

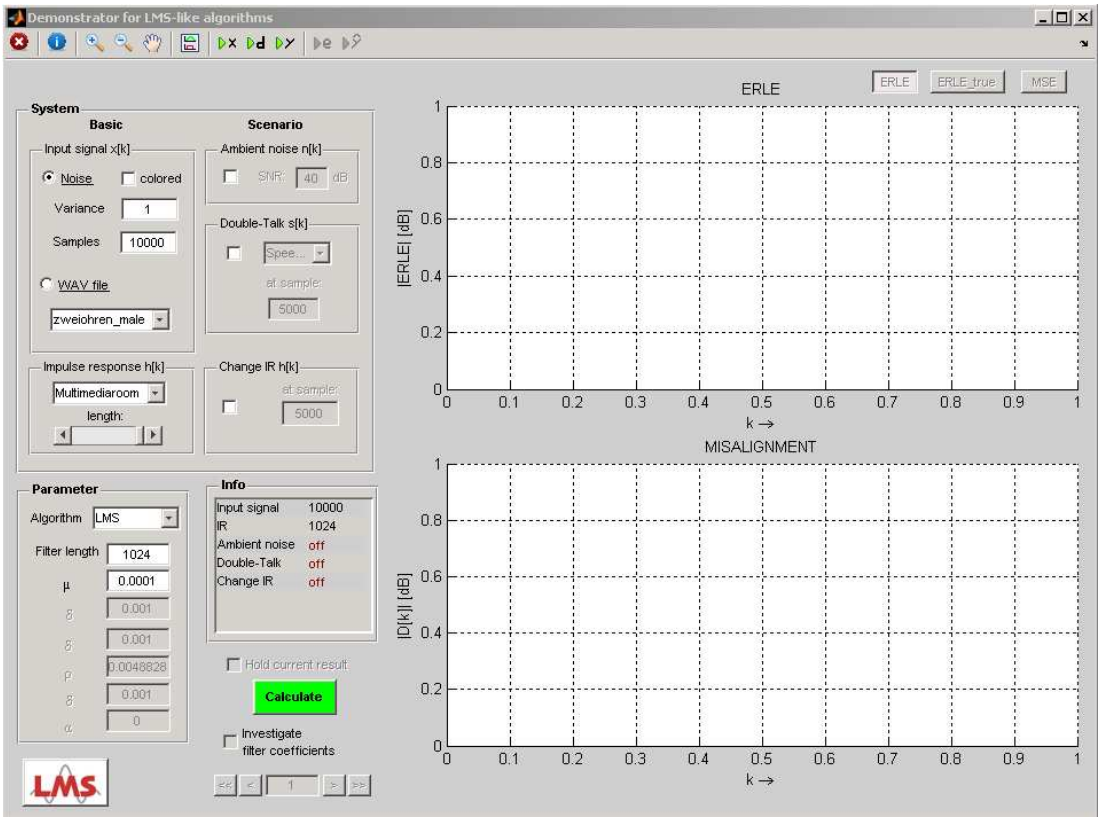

Abbildung 4.1: Gesamte Oberfläche

Bevor man die Oberfläche startet, wechselt man in Matlab in das Verzeichnis, in dem die Datei `LMS_GUI.m` liegt, damit auch die Tooltips vollständig angezeigt werden können. Danach startet man `LMS_GUI.m` und es erscheint die in Abb. 4.1 zu sehende Benutzeroberfläche.

Wie man dem Bild entnehmen kann, lässt sich die GUI in sechs essentielle Bereiche einteilen. Es gibt eine Toolbar mit verschiedenen Werkzeugen, zwei Bereiche zum Einstellen des Systems und der Parameter, einen Informationsbereich, einen kleinen Abschnitt für den Koeffizientenvergleich und einen großen Ausgabebereich.

4.1 Toolbar

In der Toolbar (Abb. 4.2) befinden sich die globalen Funktionen und Werkzeuge. So lässt sich z.B. durch Drücken von „X“ die GUI schließen und mit dem Info-Button ein weiteres Fenster mit zusätzlichen Informationen öffnen. Rechts daneben befinden sich drei Werkzeuge um in geplotteten Graphen zu navigieren, so kann man mit der Lupe hinein- und herauszoomen und mit der „Hand“ der Plot verschieben.

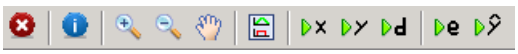

Abbildung 4.2: Toolbar

Rechts davon befindet sich ein Knopf, um relevante Signale zu plotten. Darauf öffnet sich ein neues Fenster (Abb. 4.3), in dem das Lautsprechersignal $x[k]$, die Raumimpulsantwort $\mathbf{h}[k]$, das Mikrofonsignal $y[k]$, das geschätzte Mikrofonsignal $\hat{y}[k]$, die geschätzte Impulsantwort $\hat{\mathbf{h}}[k]$ und der Restfehler $e[k]$ angezeigt werden.

Die nächsten, wieder rechts davon gelegenen, Buttons dienen dazu die eingestellten bzw. die berechneten Signale anzuhören. Die Signale $x[k]$, $d[k]$ und $y[k]$ lassen sich sofort anhören, die anderen erst, nachdem man etwas berechnet hat. Für den Fall, dass keine Störungen vorhanden sind, also $n[k]$ und $s[k]$ Null sind, gilt $y[k] = d[k]$.

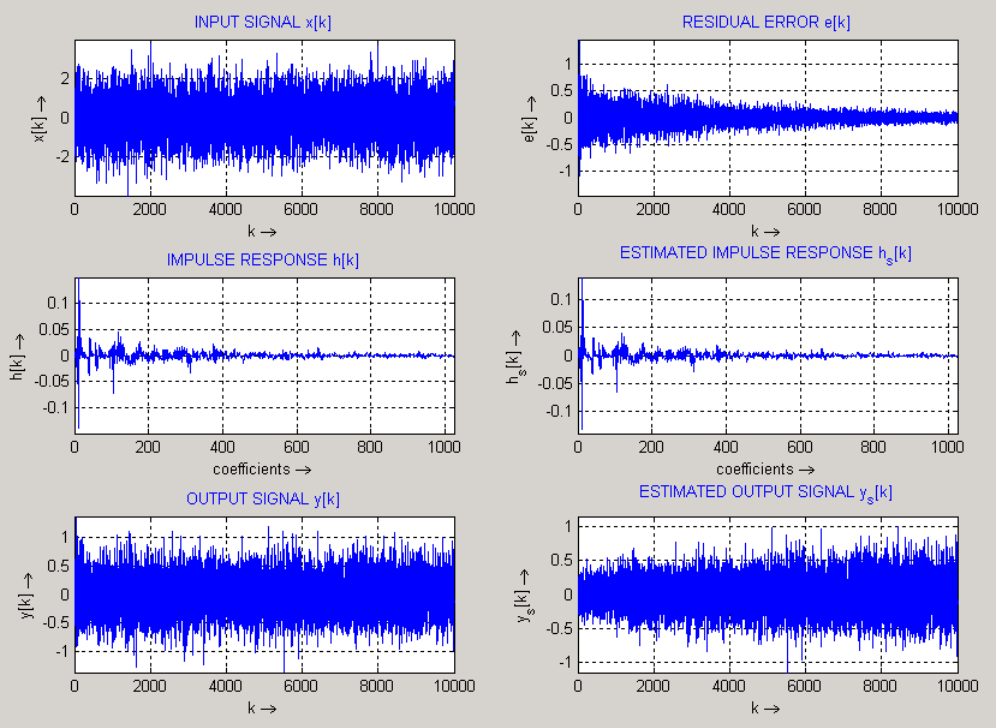

Abbildung 4.3: Plot relevanter Signale

4.2 Systemeinstellungen

In diesem Bereich hat man die Möglichkeit ein bestimmtes Szenario für die Systemidentifikation einzustellen (Abb. 4.4). So kann man zuerst das Basissystem einstellen und als Eingangssignal $x[k]$ beispielsweise Gaußsches Rauschen erzeugen und dafür die Varianz und die Länge einstellen. Außerdem kann man durch setzen eines Hakens bei *colored*, farbiges Rauschen aktivieren, das durch Filterung mit `filter([0.5 0.5],1,x)` aus dem weißen Rauschen erzeugt wird.

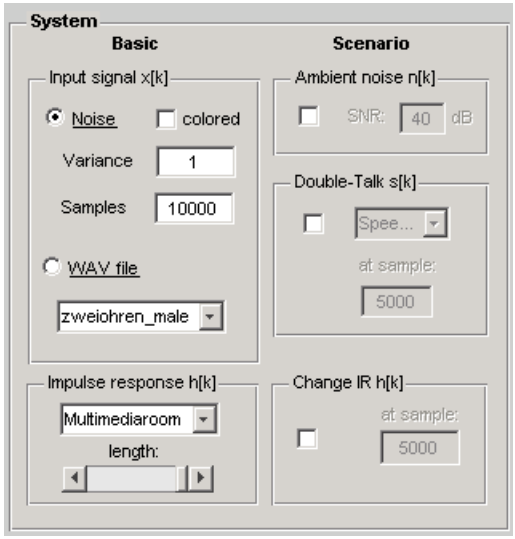

Abbildung 4.4: Systemeinstellungen

Als weiteres Eingangssignal stehen entweder zwei vorgegebene Sprachsignale zur Verfügung oder man fügt eigene WAV-Dateien hinzu. Das kann man durch das Popup-Menü unter *WAV file* auswählen. Bei längerem Verweilen mit dem Mauszeiger über dem Popup-Menü, erscheint ein Tooltip mit weiteren Informationen zum gewählten Signal. So kann man z.B. die Länge oder die Abtastfrequenz des Signals ablesen.

Ebenfalls zur Einstellung des Basissystems gehört die Wahl einer Raumimpulsantwort. Dies geschieht mit dem Popup-Menü bei *Impulse response h[k].* Es stehen fünf vorgegebene Impulsantworten zur Verfügung, nähere Informationen dazu erscheinen wieder als Tooltip. Natürlich hat man auch wieder die Möglichkeit eigene Dateien hinzuzufügen und zusätzlich kann man mit *length* die Länge der Impulsantwort beschränken, um z.B. eine schnellere Konvergenz zu erreichen.

Nachdem man die Basiseinstellungen gewählt hat, kann man ein zusätzliches Szenario einstellen. Um eine Störung hinzuzufügen, aktiviert man *Ambient noise n[k]*

und stellt einen SNR Wert in dB ein. Somit wird dem Echosignal $d[k]$ weißes Gaußsches Rauschen überlagert. Für einen „Double-Talk“ Fall setzt man einen Haken bei *Double-Talk s[k]*, wählt eine vorgegebene Sprachdatei mit einer bestimmten Länge aus dem Popup-Menü aus und stellt mit dem Feld *at sample* ein, ab welchem Zeitpunkt der lokale Sprecher s[k] aktiv sein soll. Als letzte Einstellmöglichkeit für ein Szenario steht die Funktion *Change IR h[k]* zur Verfügung. Damit kann man festlegen, zu welcher Zeit ein Wechsel der Impulsantwort statt finden soll. Dazu wird ab einem bestimmten Sample, die gerade benutzte Impulsantwort um 12 Samples nach rechts verschoben, was als Simulation eines vergrößerten Lautsprecher-Mikrofon-Abstandes interpretiert werden kann.

4.3 Parametereinstellungen

In dem Panel *Parameter* kann man Einstellungen für die verschiedenen Algorithmen vornehmen. Dazu wählt man mittels des obersten Popup-Menüs den gewünschten Adaptionsalgorithmus aus und kann danach die spezifischen Parameter des jeweiligen Algorithmus einstellen. Als Tooltip erscheint bei diesem Popup-Menü die Update-Gleichung des gewählten Algorithmus und die Parameter, die man dabei einstellen kann.

Die ersten beiden Parameter sind für alle Algorithmen gleich. So kann man mit L die Filterlänge des adaptiven Filter $\hat{\mathbf{h}}[k]$ und mit μ die Schrittweite einstellen. Bei der Schrittweite ist darauf zu achten, dass beim LMS $0 < \mu < 2/\left\|\mathbf{x}[k]\right\|^2$ gilt und bei den anderen Algorithmen die normierte Schrittweite $0 < \mu < 2$.

Als nächstes folgen die spezifischen Parameter der jeweiligen Adaptionsalgorithmen. Bei Wahl des LMS-Algorithmus bleibt es bei den beiden Parametern *Filterlänge* und *Schrittweite*.

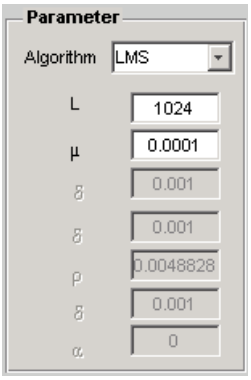

Abbildung 4.5: Parametereinstellungen

Falls man sich für den NLMS-Algorithmus entschieden hat, steht neben der *Filterlänge* und der *Schrittweite* noch ein weiterer Parameter zur Verfügung, der sogenannte *Regularisierungs-Parameter* δ. Dieser ist laut Update-Gleichung des NLMS (Gl. 2.13) dafür zuständig eine Division durch Null zu vermeiden, falls $\mathbf{x}^T[k]\mathbf{x}[k] \approx 0$. Den Regularisierungsfaktor gibt es jeweils auch beim PNLMS und IPNLMS.

Beim PNLMS gibt es ingesamt vier Parameter zum Einstellen. Die ersten beiden sind wieder *Filterlänge* und *Schrittweite*, dann der Regularisierungsparameter δ und zuletzt ρ, den man nach Abschnitt 2.3 am besten zu $\rho = 5/L$ wählt.

Für den IPNLMS bleibt es bei vier Parametern, wobei die ersten beiden wieder gleich bleiben, der dritte ebenfalls der Regularisierungsfaktor und der Vierte in diesem Fall α ist. Ein geeigneter Wertebereich dafür findet sich in Abschitt 2.4.

4.4 Infobereich

Nachdem man jetzt das Basissystem, ein eventuelles Szenario und den gewünschen Algorithmus mit den passenden Parametern gewählt hat, sieht man im Infobereich die wichtigsten Informationen und eventuelle Hinweise.

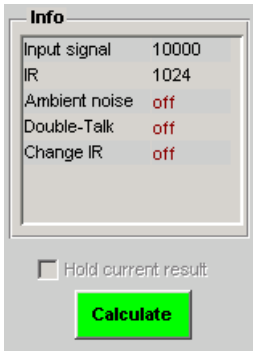

Abbildung 4.6: Infobereich

4.5 Berechnung und Hold-Option

Nach den jeweiligen Einstellungen kann man mit Druck des *Calculate-Button* (Abb. 4.6) die Berechnung starten. Es erscheint ein Fortschrittsbalken, der angibt, was im Moment berechnet wird und an dem man die ungefähre Restdauer ablesen kann. Falls man bereits ein Ergebnis aus vorherigen Berechnungen geplottet hat, kann man durch Setzen eines Hakens bei *Hold current result* die aktuellen Ergebnisse beibehalten und so für die neuen Berechnungen einen direkten Vergleich zu den Vorherigen bekommen. Man kann damit auf einfache und elegante Weise die verschiedenen Algorithmen mit jeweils verschiedenen Einstellungen gegeneinander vergleichen. Das gleiche gilt auch für die Betrachtung der Filterkoeffizienten (Abschnitt 4.6.3).

4.6 Ausgabebereich

In Abb. 4.7 sieht man, dass der Ausgabebereich zweigeteilt ist und zwar in einen oberen und einen unteren Plot.

4.6.1 Oberer Plot

Im oberen Bereich wird beim ersten Start der GUI bzw. nach der ersten Berechnung das ERLE-Maß angezeigt. Man hat jedoch die Möglichkeit per *Toggle-*

Button zwischen den drei Ergebnissen ERLE, ERLE$_{true}$ und MSE zu wechseln. Wie man das ERLE Maß berechnet steht in Abschnitt 3.1.2.

4.6.2 Unterer Plot

Im unteren Bereich hingegen wird der Systemabstand $D[k]$ angezeigt. Die Formel zur Berechnung findet man in Kapitel 3.1.1.

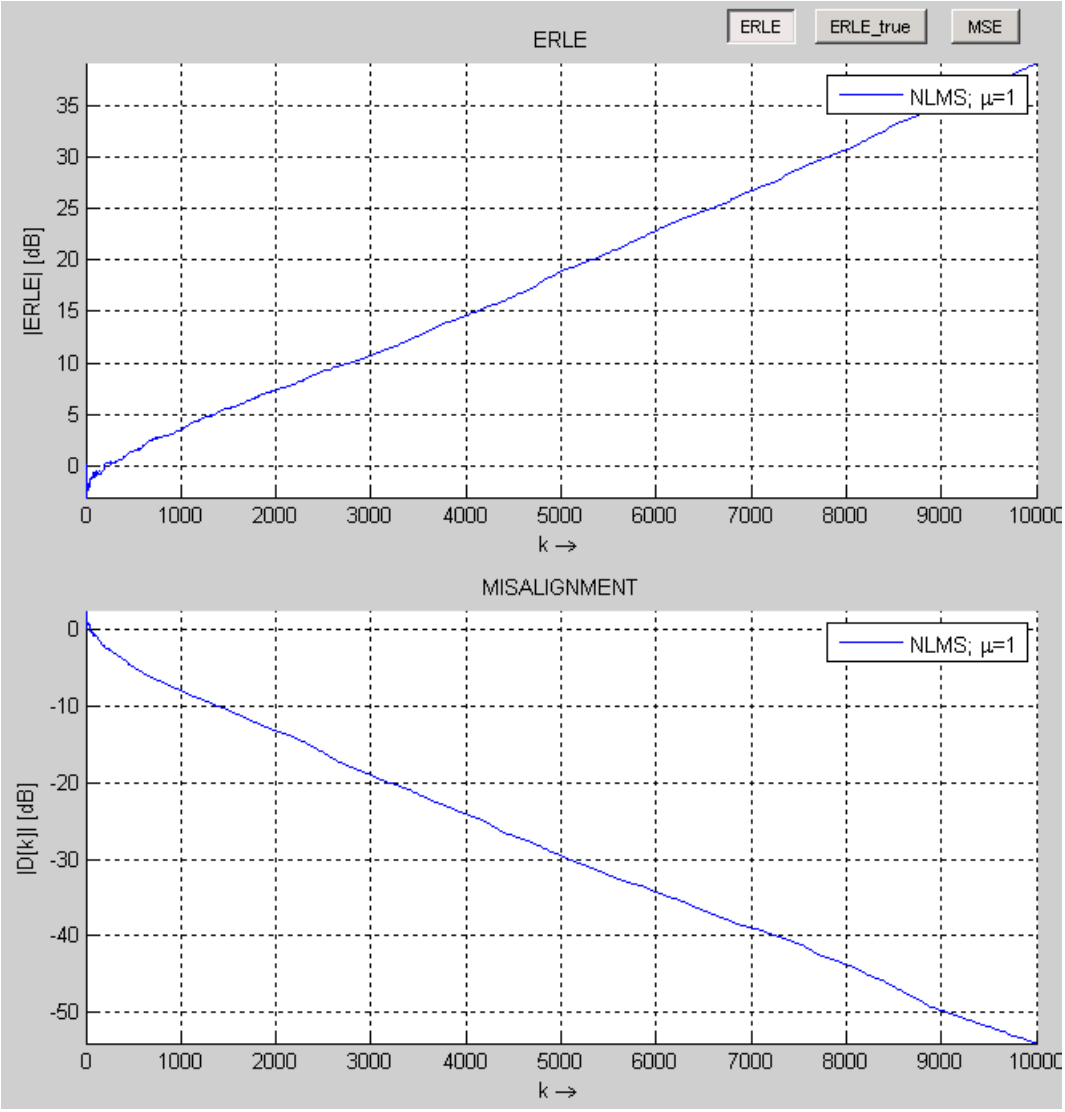

Abbildung 4.7: Ausgabebereich

4.6.3 Koeffizientenvergleich

Falls man bereits eine oder mehrere adaptive Filterungen berechnet hat, kann man duch Setzen eines Hakens bei *Investigate filter coefficients* in den Modus des Filterkoeffizientenvergleichs umschalten. Dabei ändert sich der obere Anzeigebereich zu dem Differenzsignal zwischen der geschätzten Impulsantworten und der „wahren“ Impulsantwort. Der untere Bereich hingegen dient nun dazu die einzelnen Impulsantworten als Balkenplot anzuzeigen. Dabei ist die originale Impulsantwort türkis gefärbt und die anderen befinden sich in ihrer jeweiligen Farbe daneben. Durch Wechseln der einzelnen Koeffizienten mit den Buttons aus Abb. 4.8 oder direkter Eingabe eines Wertes in das Eingabefeld, kann man die Adaption schrittweise beobachten.

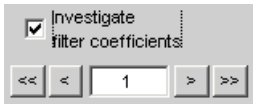

Abbildung 4.8: Koeffizientenvergleich

Um wieder in den „normalen“ Anzeigemodus zurückzukommen, muss man den Haken bei *Investigate filter coefficients* entfernen. Dadurch wird im oberen Plot wieder automatisch das ERLE Maß angezeigt und im Unteren der Systemabstand.

In Abb. 4.9 sieht man ein Beispiel für einen Koeffizientenvergleich. Dabei wurde weißes Rauschen mit 10000 Samples als Eingangssignal und *Car* als Raumimpulsantwort verwendet. Als einziges Szenario wurde Umgebungsrauschen mit 40 dB gewählt. Wie man der Legende entnehmen kann, wurde als Algorithmus der NLMS mit einer Schrittweite $\mu = 0.2$ eingestellt. Der Regularisierungsfaktor δ wurde zu 0.001 gewählt und die Filterlänge L beträgt 370 Samples. Abb. 4.9 zeigt nun eine gezoomte Darstellung an der Stelle $k = 3000$, d.h. in Abb. 4.8 wurde direkt 3000 eingegeben. Im unteren Plot sieht man die türkise, originale

Impulsantwort und dunkelblau daneben die aktuelle geschätzte Impulsantwort. Im oberen Plot ist die Differenz der beiden aufgetragen. Auf diese Weise kann man direkt beobachten, wie die einzelnen Koeffizienten angepasst werden.

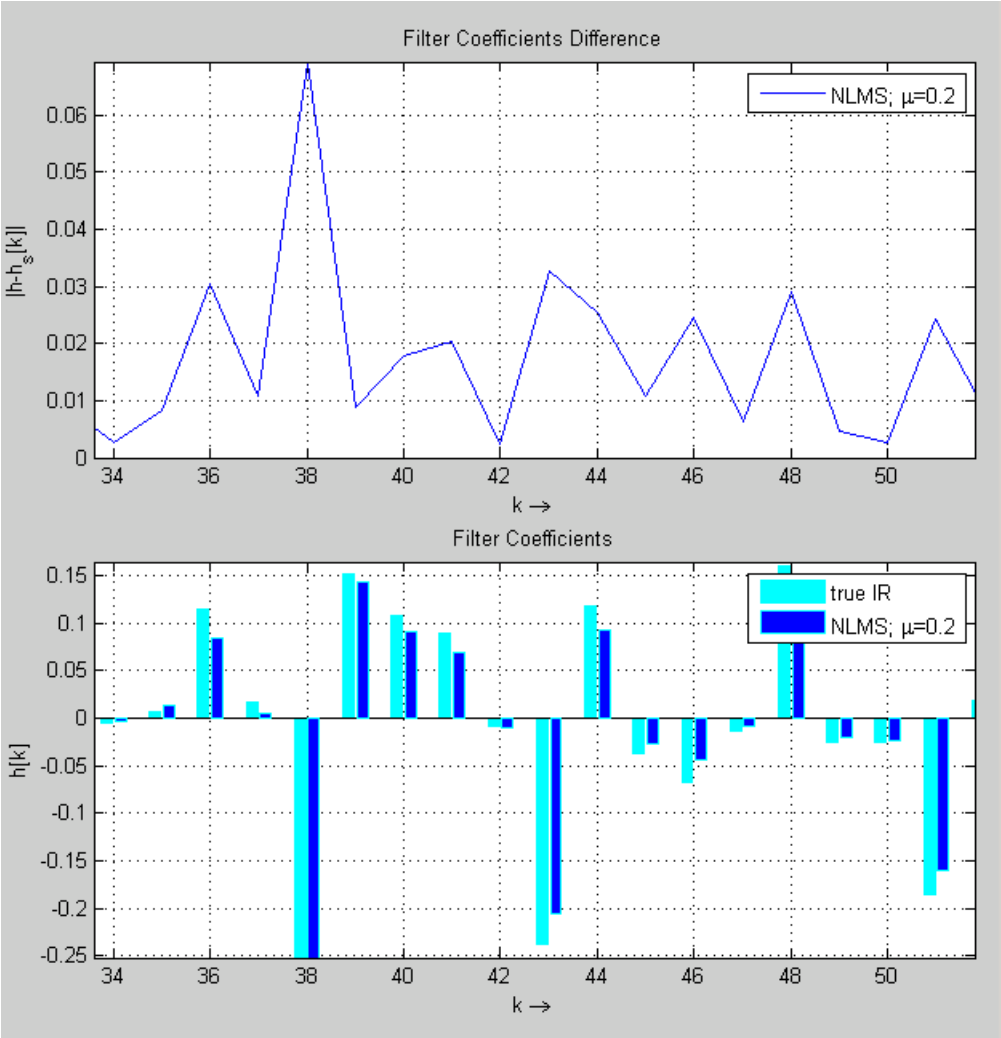

Abbildung 4.9: Beispiel eines Filterkoeffizientenvergleichs

Kapitel 5

Zusammenfassung

In dieser Arbeit wurde zuerst ein grober Überblick über adaptive Filter im Allgemeinen gegeben und anschließend anhand des speziellen Anwendungsbeispiels der Echokompensation die wichtigsten LMS-artigen Algorithmen, wie der LMS, der NLMS, der PNLMS und der IPNLMS, hergeleitet. Danach wurden einige Bewertungskriterien vorgestellt und die drei Algorithmen NLMS, PNLMS und IPNLMS anhand des Systemabstandes miteinander verglichen. Zum Abschluss wurden die vier angesprochenen Algorithmen in Matlab implementiert und mit dem in Matlab integrierten Tool *GUIDE* eine Oberfläche erstellt, mit der man bestimmte Szenarien mit den verschiedenen Algorithmen untersuchen kann. Das Hauptaugenmerk lag dabei auf der Visualisierung der einzelnen Adaptionsschritte, was durch den Koeffizientenvergleich (Abschnitt 4.6.3) gut möglich ist. Zusätzlich kann man per Hold-Option (Abschnitt 4.5) mehrere Szenarien nacheinander berechnen und direkt miteinander vergleichen.

Betrachtet wurden hier nur LMS-artige Adaptionsalgorithmen. Eine Alternative dazu wäre z.B. der RLS-Algorithmus, der sich durch eine hohe Konvergenzgeschwindigkeit auszeichnet, jedoch ist die rechentechnische Komplexität im Vergleich zum LMS-Algorithmus um ein Vielfaches höher, so dass den Vorteilen des RLS-Algorithmus auch erhebliche Nachteile gegenüberstehen.

Anhang A

Quellcode

A.1 LMS

```
function [e, Hs, ys] = lms(x, d, mu, L)

% INITIALISIERUNGEN
N = length(x);
X = zeros(L, 1);
Hs = cell(N,1);
hs = zeros(L 1);
ys = zeros(N,1);
e = zeros(N,1);

% START DES LMS ALGORITHMUS
for i = 1:N
    X = [x(i);X(1:end-1)];
    ys(i) = hs' * X;
    e(i) = d(i) - ys(i);
    hs = hs + mu * e(i) * X;
    Hs{i} = hs;
end
```

Quellcode A.1: lms.m

A.2 NLMS

```
function [e, Hs, ys] = nlms(x, d, mu, deltaN, L)

% INITIALISIERUNGEN
N = length(x);
X = zeros(L, 1);
Hs = cell(N,1);
hs = zeros(L, 1);
ys = zeros(N,1);
e = zeros(N,1);

% START DES NLMS ALGORITHMUS
for i = 1:N
    X = [x(i);X(1:end-1)];
    ys(i) = hs' * X;
    e(i) = d(i) - ys(i);
    hs = hs + (mu * e(i) * X) / (X' * X + deltaN);
    Hs{i} = hs;
end
```

Quellcode A.2: nlms.m

A.3 PNLMS

```
function [e, Hs, ys] = pnlms(x, d, mu, deltaP, rho, L)

% INITIALISIERUNGEN
N = length(x);
X = zeros(L, 1);
Hs = cell(N,1);
hs = zeros(L, 1);
ys = zeros(N,1);
e = zeros(N,1);
g = zeros(L,1);
G = zeros(L,1);
delta_p = 0.01;

% START DES PNLMS ALGORITHMUS
for i = 1:N
    X = [x(i);X(1:end-1)];
    ys(i) = hs' * X;
    e(i) = d(i) - ys(i);
    gamma = max(rho * max(delta_p,max(abs(hs))),abs(hs));
    g = gamma / sum(gamma);
    G = repmat(g,1);
    hs = hs + (mu * e(i) * G .* X) / ((X .* G)' * X + deltaP);
    Hs{i} = hs;
end
```

Quellcode A.3: pnlms.m

A.4 IPNLMS

```
function [e, Hs, ys] = ipnlms(x, d, mu, deltaI, alpha, L)

% INITIALISIERUNGEN
N = length(x);
X = zeros(L, 1);
Hs = cell(N,1);
hs = zeros(L, 1);
ys = zeros(N,1);
e = zeros(N,1);
k = zeros(L,1);
K = zeros(L,1);
epsilon = 0.0001;

% START DES IPNLMS ALGORITHMUS
for i = 1:N
    X = [x(i);X(1:end-1)];
    ys(i) = hs' * X;
    e(i) = d(i) - ys(i);
    k = (1 - alpha) / (2 * L) + ((1 + alpha) * abs(hs)) / (2 * sum(abs(hs
        )) + epsilon);
    K = repmat(k,1);
    hs = hs + (mu * e(i) * K .* X) / ((X .* K)' * X + deltaI);
    Hs{i} = hs;
end
```

Quellcode A.4: ipnlms.m

Anhang B

Abkürzungen und Formelzeichen

B.1 Abkürzungen

ADPCM	adaptive differential pulse code modulation
AEC	acoustic echo cancelation
ERLE	echo return loss enhancement
FIR	finite impulse response
GUI	graphical user interface
IIR	infinite impulse response
IPNLMS	improved proportionate normalized least mean square
LMS	least mean square
LPC	linear predictive coding
MSE	mean square error
NLMS	normalized least mean square
PNLMS	proportionate normalized least mean square
VAD	voice activity detection

B.2 Formelzeichen

k	Zeitindex
$x[k]$	Eingangssignal (Lautsprechersignal)
$\mathbf{h}[k]$	Raumimpulsantwort
$\hat{\mathbf{h}}[k]$	geschätzte Raumimpulsantwort
$y[k]$	Ausgangssignal (Mikrofonsignal)
$\hat{y}[k]$	geschätztes Ausgangssignal
$s[k]$	Double-Talk Signal (lokaler Sprecher)
$n[k]$	Störsignal
$d[k]$	Echosignal
$e[k]$	Fehlersignal
μ	Adaptionsschrittweite
μ_n	normierte Adaptionsschrittweite
δ_{NLMS}	Regularisierungskonstante des NLMS Algorithmus
δ_{PNLMS}	Regularisierungskonstante des PNLMS Algorithmus
δ_{IPNLMS}	Regularisierungskonstante des IPNLMS Algorithmus
∇	Gradient
$E\{.\}$	Erwartungswert-Operator
$\max\{.\}$	Maximum-Operator
$\lvert.\rvert$	Betrags-Operator
$\lVert.\rVert_1$	L_1-Norm
$\lVert.\rVert$	L_2-Norm

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] Walter Kellermann, *Stochastische Prozesse*, Skriptum zur Vorlesung, Friedrich-Alexander-Universität Erlangen-Nürnberg, April 2010.

[2] Walter Kellermann, *Praktikum Digitale Signalverarbeitung*, Skriptum zum Praktikum, Friedrich-Alexander-Universität Erlangen-Nürnberg, April 2010.

[3] D.L. Duttweiler, *Proportionate normalized least mean square adaption in echo cancelers*, IEEE Trans. Speech Audio Processing, vol. 8,pp. 508-518, September 2000.

[4] Jacob Benesty, Steven L. Gay, *An Improved PNLMS Algorithm*, Murray Hill, NJ 07974, USA , 2002.

[5] Martin Werner, *Digitale Signalverarbeitung mit MATLAB-Praktikum*, Friedr. Vieweg & Sohn Verlag, Wiesbaden, 2008.

[6] George Moschytz, Markus Hofbauer *Adaptive Filter*, Springer-Verlag Berlin Heidelberg, 2000.

[7] Simon Haykin, *Adaptive filter theory*, Third Edition, Prentice Hall, Englewood Cliffs, N.J., 1996.

[8] P. Vary, U. Heute, W. Hess, *Digitale Sprachsignalverarbeitung*, B.G. Teubner, Stuttgart, 1998.

[9] André Kaup, *Signale und Systeme II*, Skriptum zur Vorlesung, Friedrich-Alexander-Universität Erlangen-Nürnberg, April 2009.

[10] Constantin Paleologu, Jacob Benesty, Silviu Ciochină, *Sparse Adaptive Filters for Echo Cancellation*, B.H. Juang, Georgia Tech, 2010.

[11] Bert-Uwe Köhler, *Konzepte der statistischen Signalverarbeitung*, Springer-Verlag Berlin Heidelberg, 2005.

[12] Bernhard Schmidt, *Implementierung und Vergleich einer neuartigen Methode zur Unterdrückung akustischer Echos in Freisprechszenarios*, Studienarbeit, Friedrich-Alexander-Universität Erlangen-Nürnberg, März 2010.

[13] Bernard Widrow, Marcian E. Hoff, *Adaptive switching circuits*, IRE WESCON Convention Record, Vol. 4, Los Angeles, CA, pages 96-104, 1960