

Echo Cancellation

Cyril Stoller


Marcel Bärtschi

Inhaltsverzeichnis

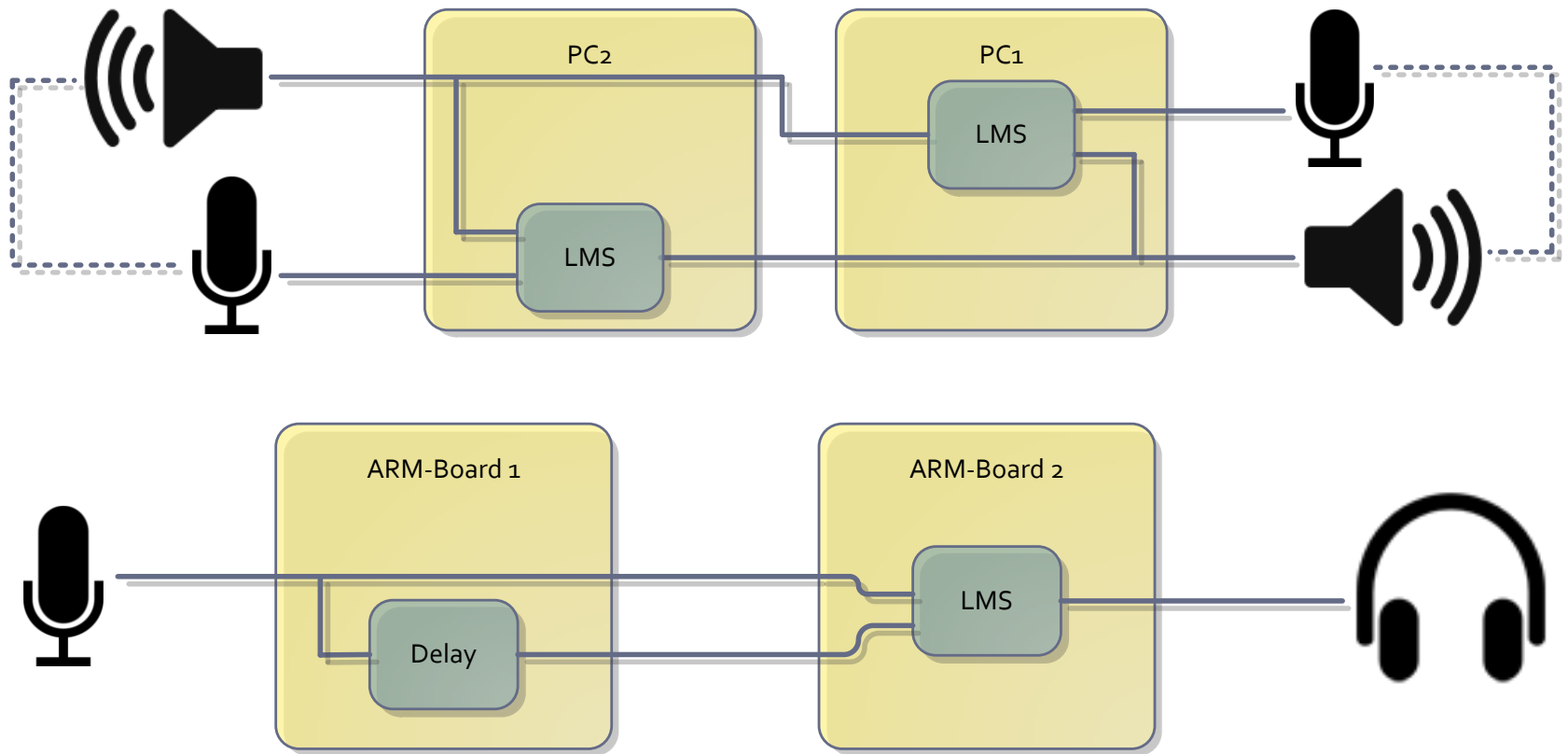
- Problematik
- Modellierung
- Lösungsansatz
- High-Level Implementation (Rauschen)
- High-Level Implementation (Stimme)
- Low-Level Implementation Echo Modellierung
- Low-Level Implementation Filterung

Phasendiagramm

Nach Vorgabe aus Modulplan POSIV:

1. Stand der Technik
2. Konzeptentwicklung
3. Umsetzung
4. Optimierung 
5. Validierung

Problematik

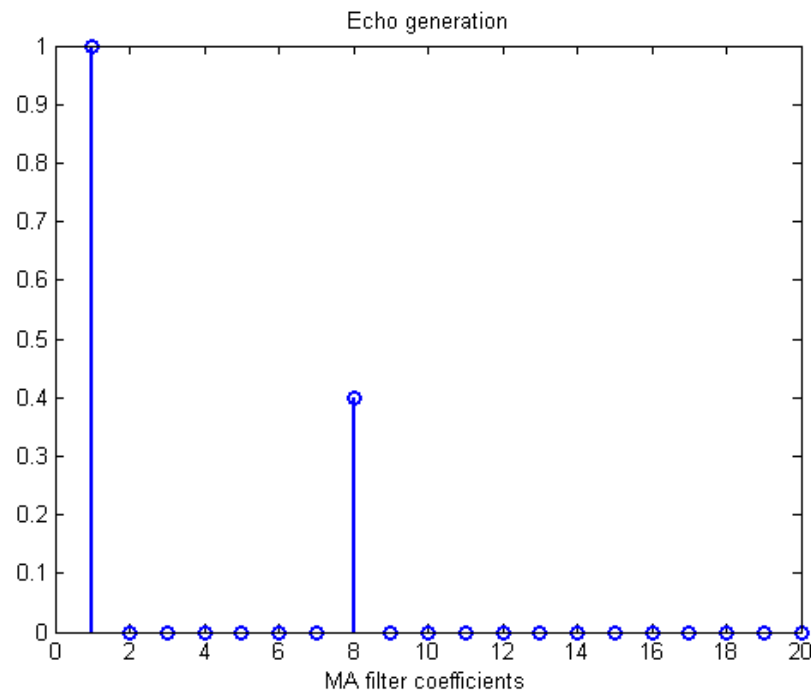


Modellierung

- Signal x wird mit einem Echo versehen
 - Echo: eine oder mehrere verzögerte und eventuell abgedämpfte Kopien von x .
- Signal x und resultierendes Signal y sind bekannt
- $x, y \rightarrow w$: Charakteristik des Echos,
- $w \rightarrow$ Wiederherstellung von x aus y

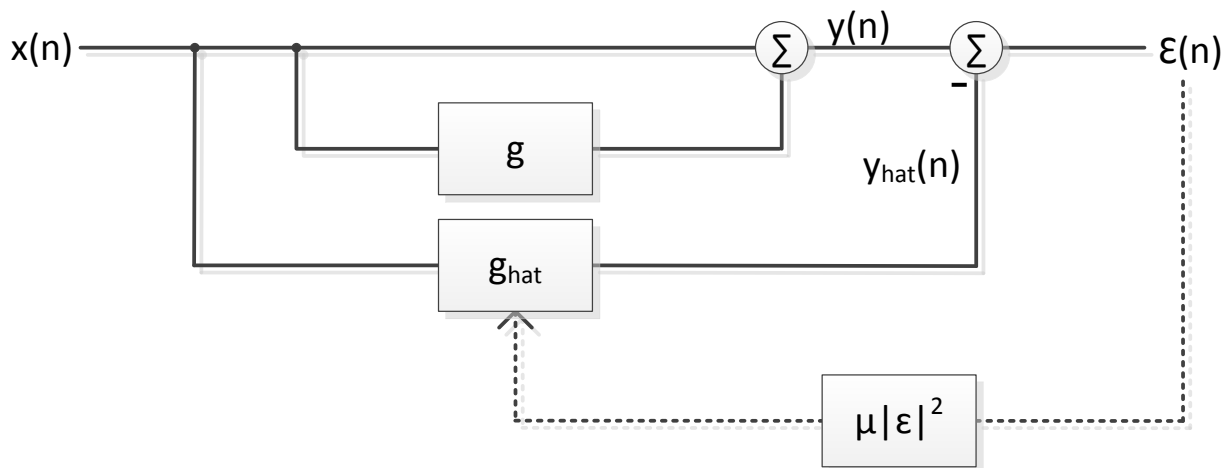
Modellierung

- Echo in Matlab-Simulation:
 - Verzögerung: 8 samples (Delay = $8/f_s$)
 - Amplitude: 0.4 des ursprünglichen Signals



Lösungsansatz

- **Adaptives LMS-Filter**
 - Signal y und Signal y_{hat} werden nun verglichen, quadriert und auf Filter zurückgeführt
 - Resultat: optimalerweise 0

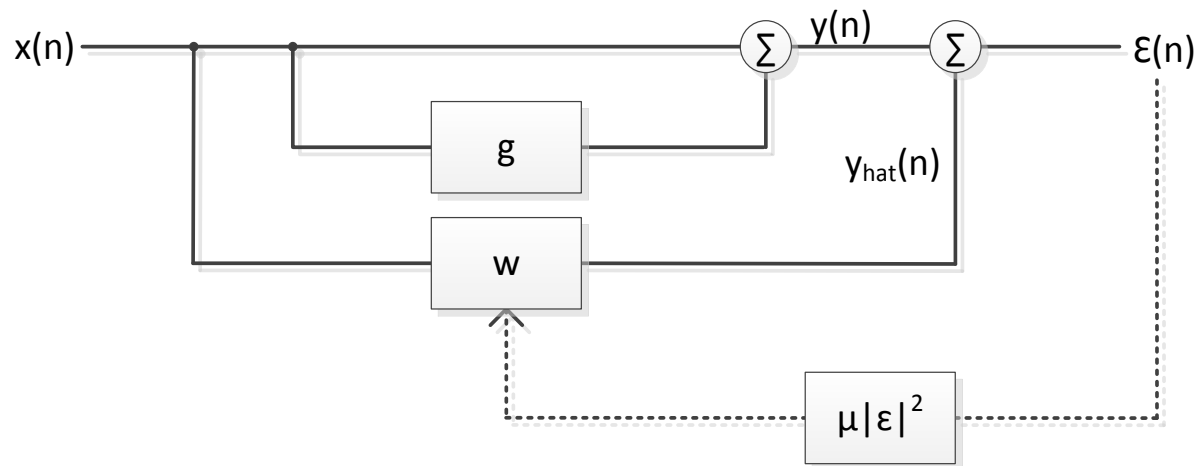
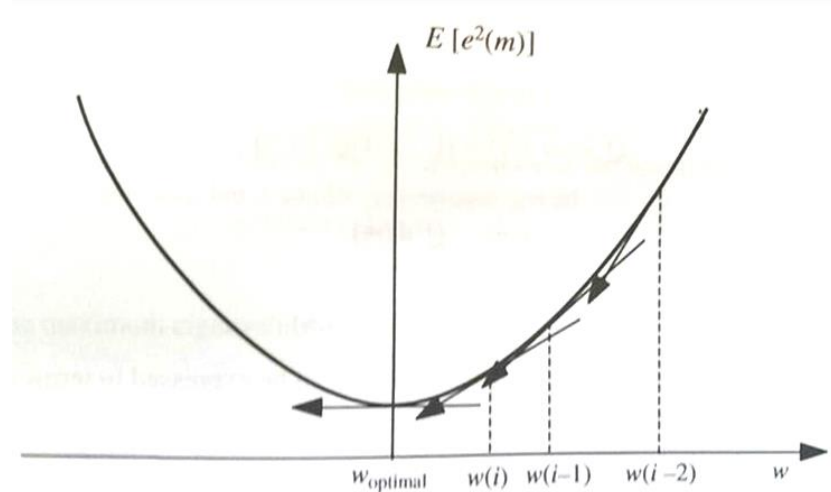


Lösungsansatz

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \mu \left\{ -\frac{\partial (\varepsilon(n)^2)}{\partial \mathbf{w}_n} \right\}$$

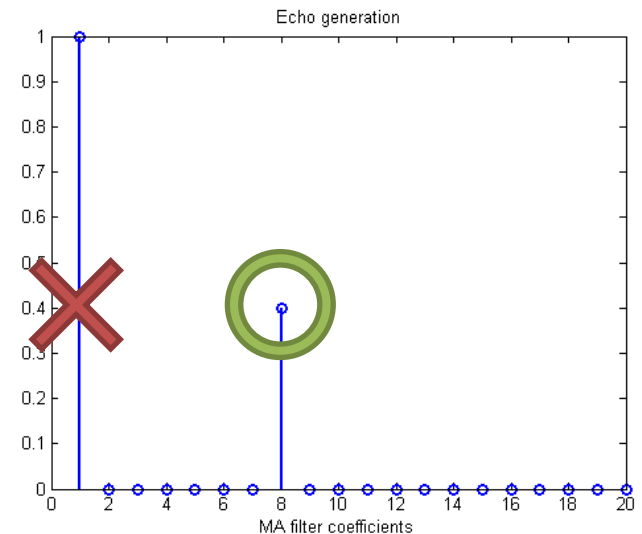
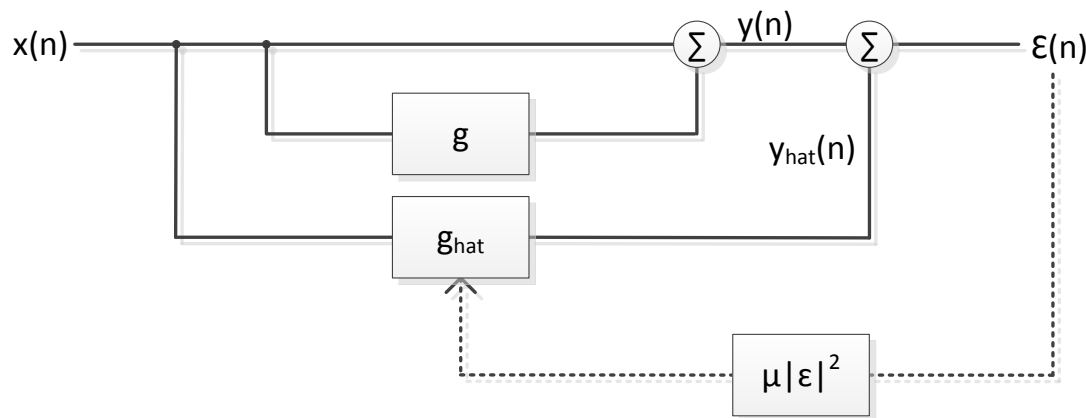
$$= \mathbf{w}_n + \mu \{ \mathbf{x}_n \varepsilon(n) \}$$

$$\varepsilon(n) = y(n) - \mathbf{w}_n^T \mathbf{x}_n^T$$



Lösungsansatz

- **Echo Cancellation** mit adaptivem LMS-Filter
 - Wenn bei g_{hat} nun das erste Filter-Tab ignoriert wird, erhält man als Resultat das Originalsignal, da bei y_{hat} nur noch das Echo nachgebildet und dieses dann von y subtrahiert wird:



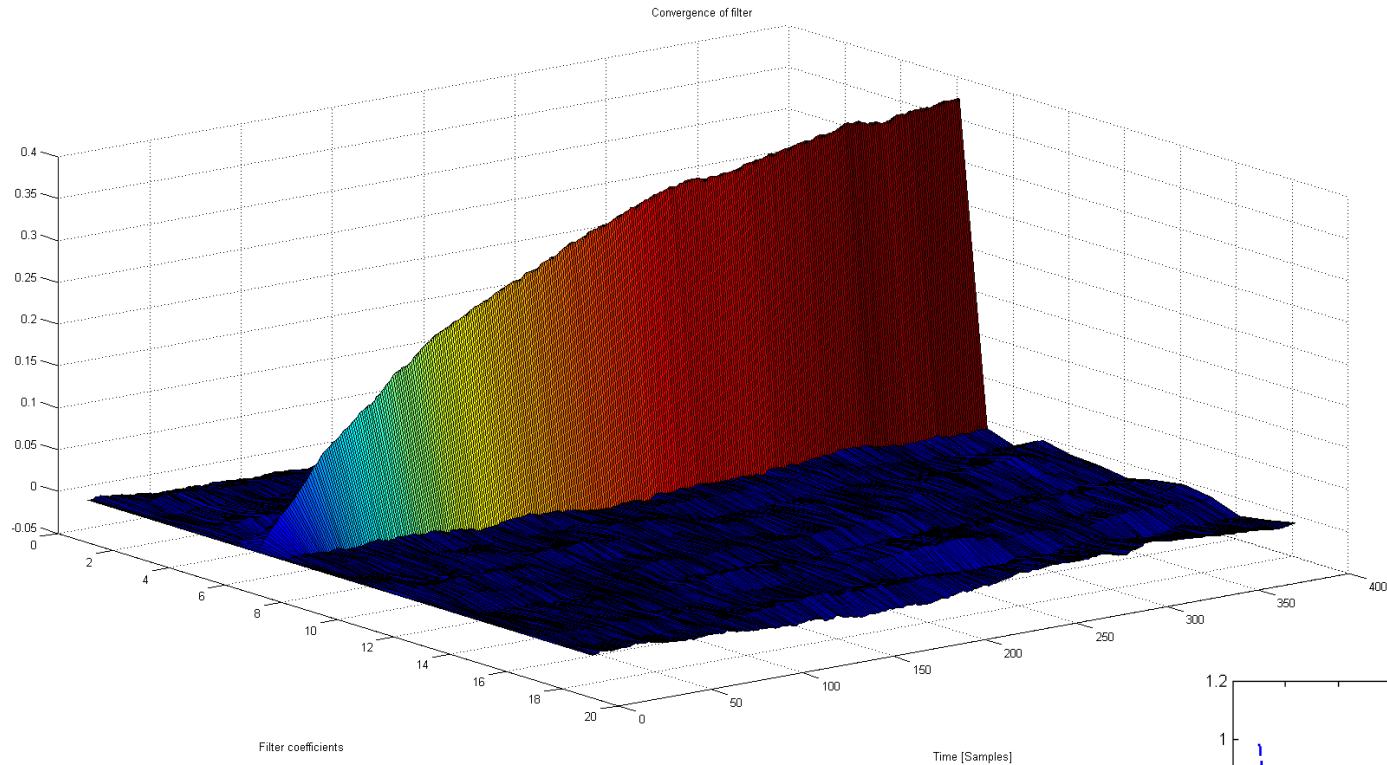
Lösungsansatz

- Echo Cancellation mit Adaptivem LMS-Filter:
 - Weit verbreitete und einfachste Methode.
 - Zahlreiche IEEE Papers und Erwähnung in praktisch jeder DSV-Literatur
 - Wird z.B. eingesetzt bei:
 - Skype
 - Teamspeak
 - Smartphone

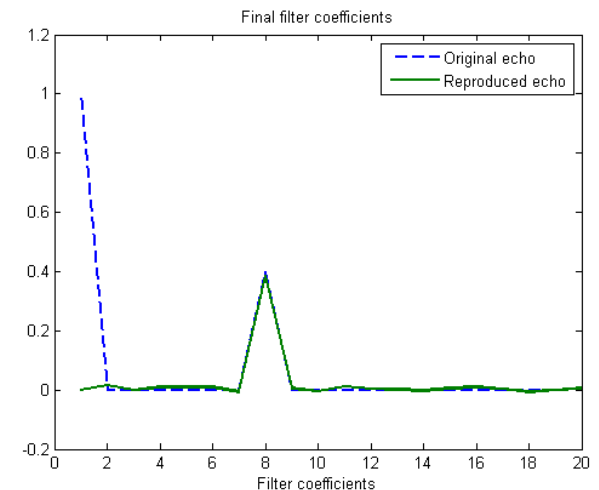
Implementation HL (noise)

- Weisses Rauschen
- Echo ist eine Verzögerung um 8 Samples und Amplitude von 0.4:
 - $g = [1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0.4]$
- Konvergenzgeschwindigkeit ist abhängig von μ
 - μ klein: genau aber langsam
 - μ gross: schnell aber ungenau (*und evtl. instabil*)

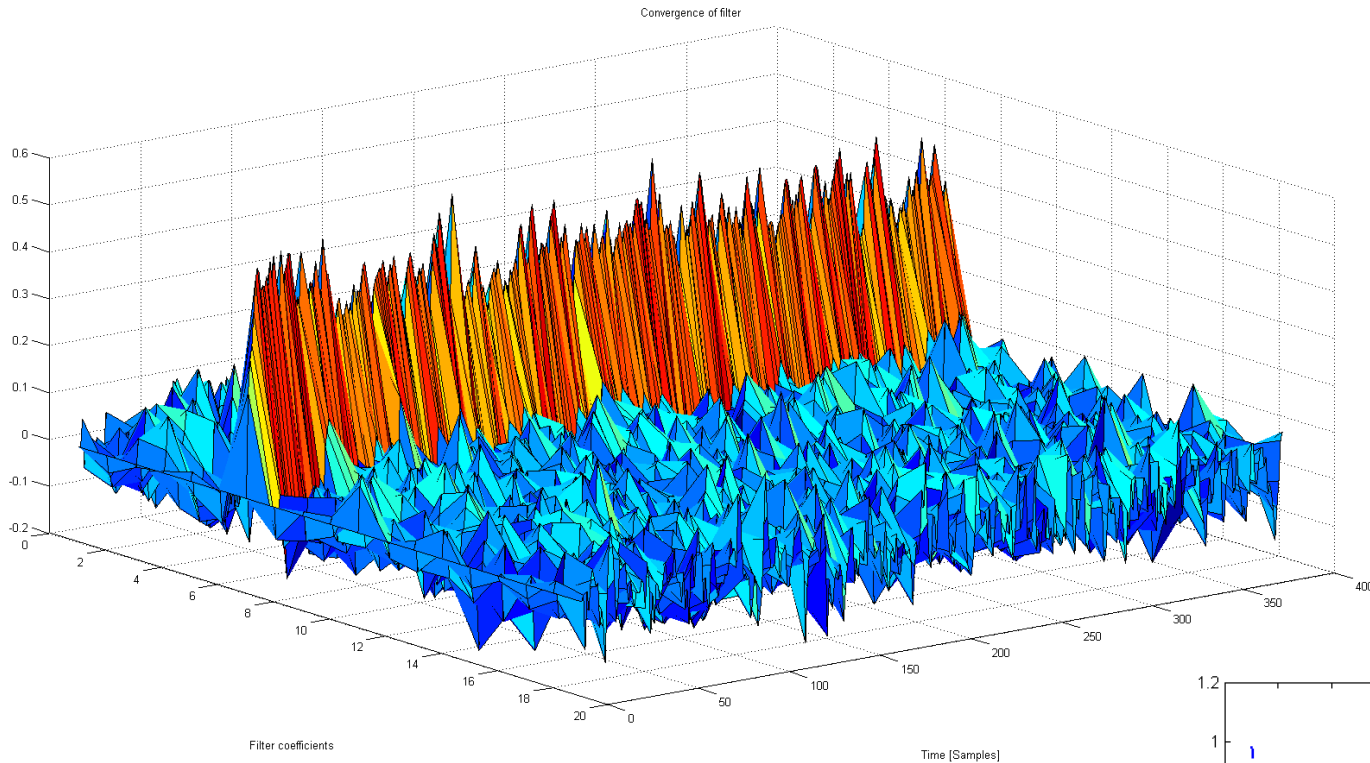
Resultate HL (Noise)



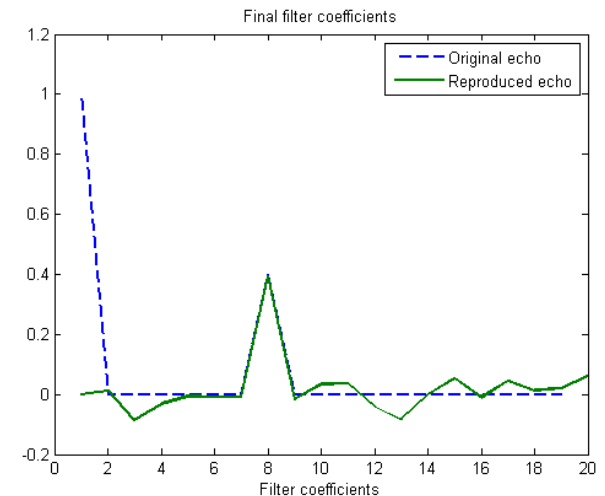
- Weisses Rauschen
- $\mu = 0.00008$ (klein)



Resultate HL (Noise)



- Weisses Rauschen
- $\mu = 0.008$ (gross)

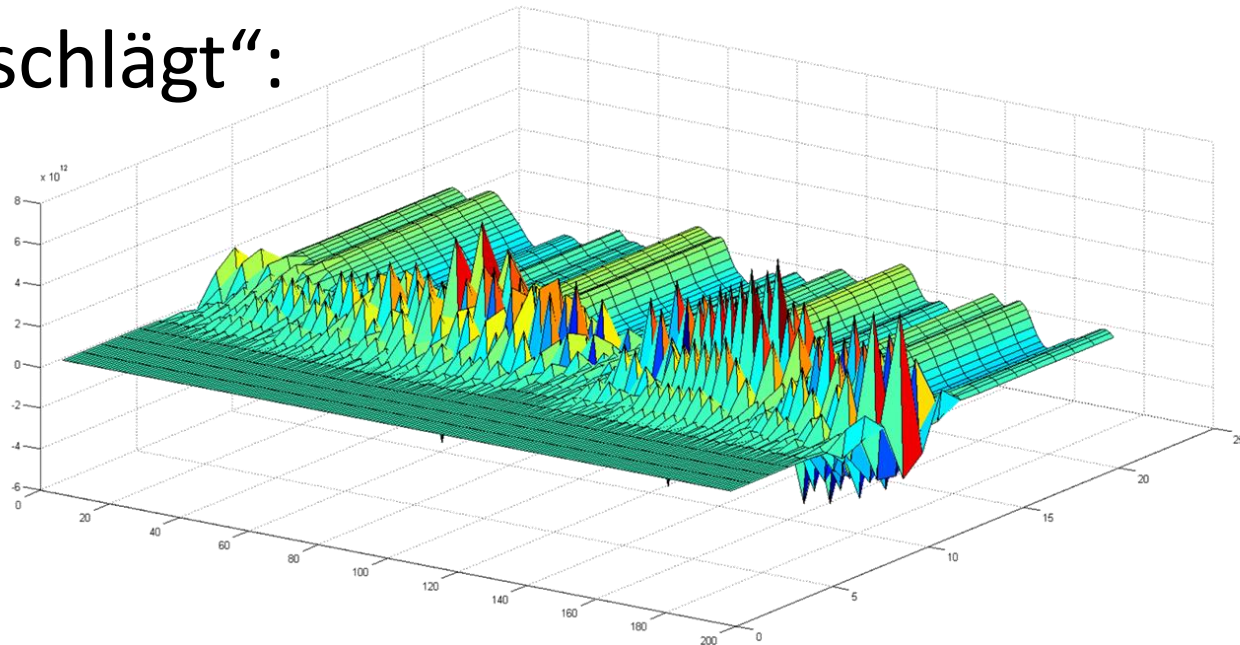


Implementation HL (voice)

- Lorem Ipsum, 2*5 Sekunden hintereinander
- Echo ist eine Verzögerung um 1470 Samples (200ms bei einer Samplefrequenz von 7350Hz) und Amplitude von 0.4:
 - $g = \text{zeros}(2000,1); \quad g(1) = 1; \quad g(1470) = 0.4;$
- Konvergenzgeschwindigkeit ist abhängig von μ
 - μ sollte $\frac{2}{(NFIR+1) \cdot \sigma_x^2}$ nicht überschreiten

Resultate HL (voice)

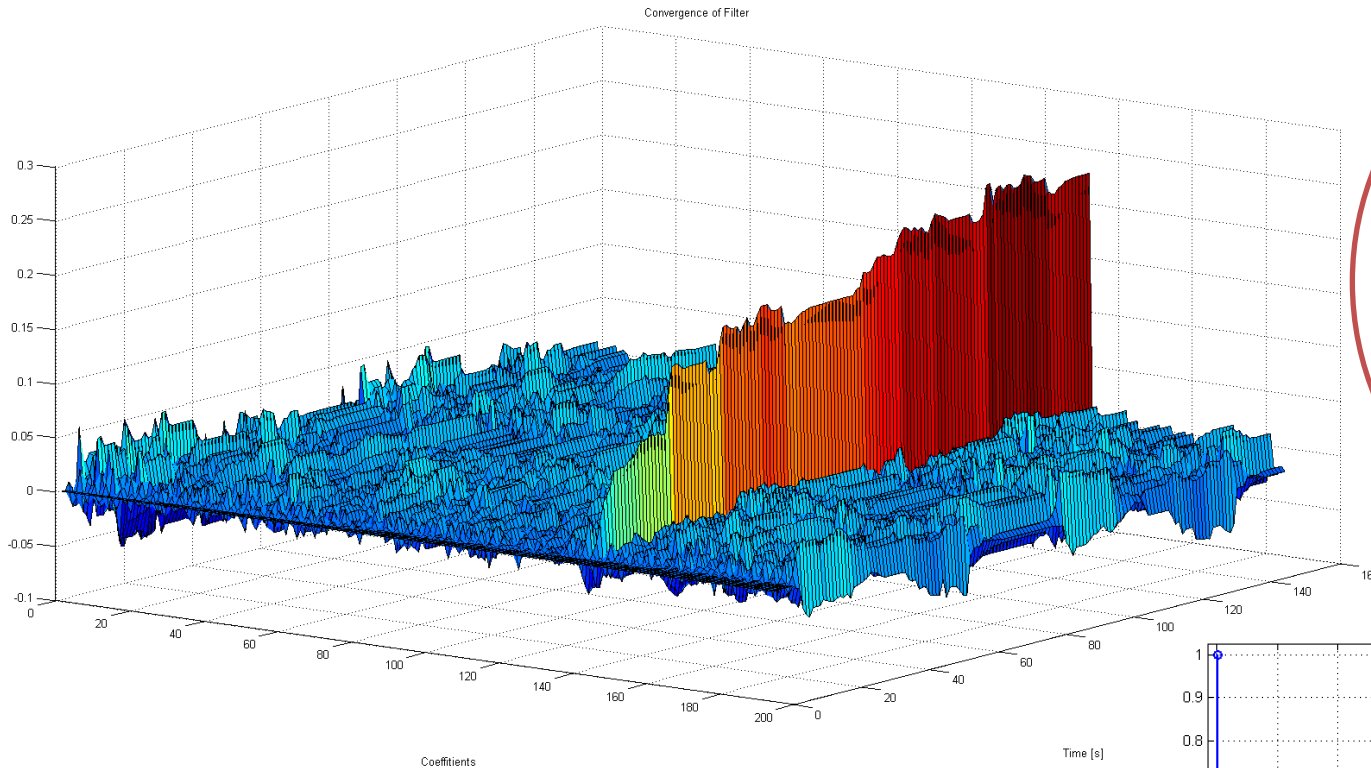
- Sprachsignal
- $\mu = \frac{2}{(NFIR+1) \cdot \sigma_x^2} = 0.1074$
- unstabil! → die Varianz ist nur quasi-stationär
- Wir haben ermittelt, ab wo das Filter „ausschlägt“:



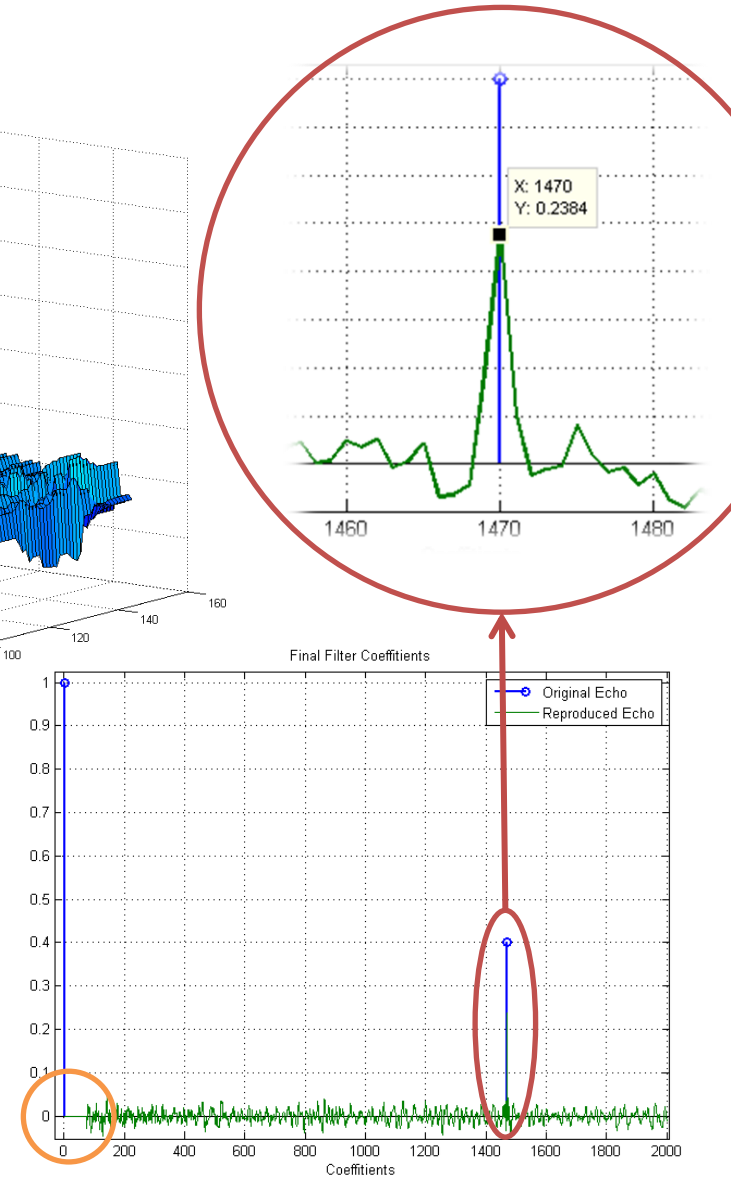
Resultate HL (voice)

- Die Varianz dort lokal bestimmt und noch einmal $\frac{2}{(NFIR + 1) \cdot \sigma_x^2}$ ausgerechnet = **0.0241**
- Interessant: Bei empirischem Anpassen des Konvergenzparameters erhielten wir noch bis **0.0284** stabile Resultate
- Recht nahe beieinander
- Resultat bei stabiler Echo-Cancellation:

Resultate HL (voice)



- Sprachsignal
- $\mu = 0.0285$ (empirisch)

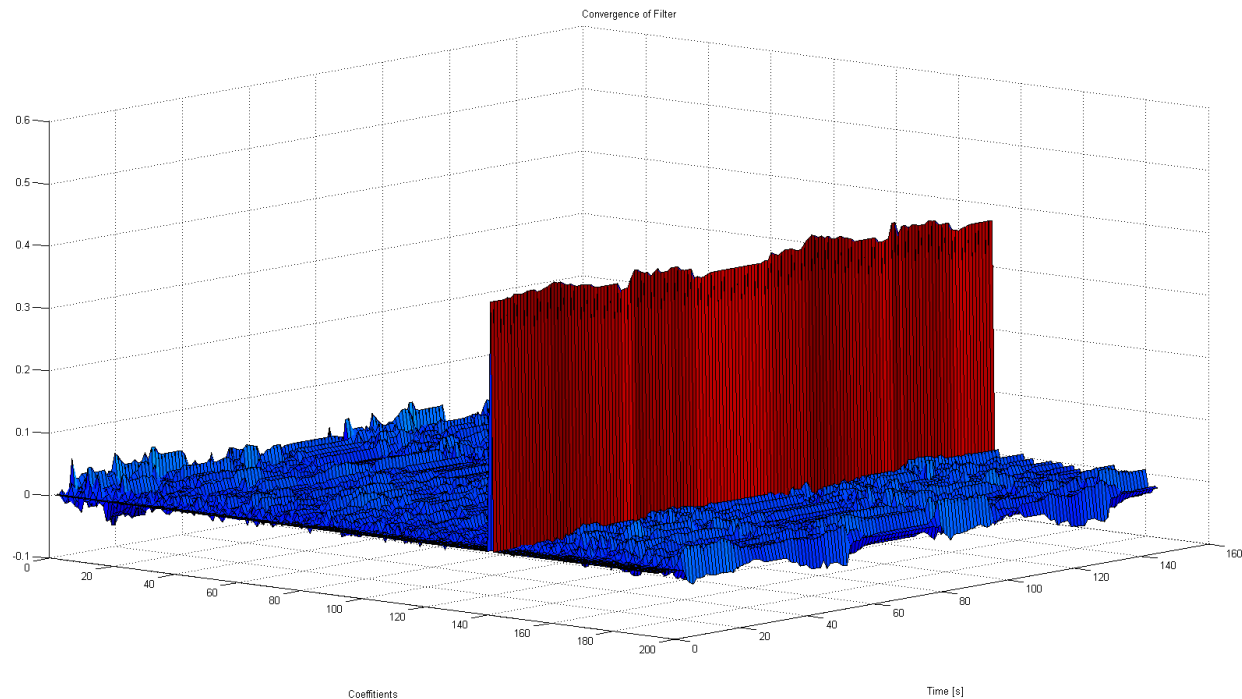


Resultate HL (voice)

- Leider nur ca. 60% des Echos herausgefiltert
- Relativ schnell: nach 2 Sekunden ist ca. 80% dieser Filterung erreicht

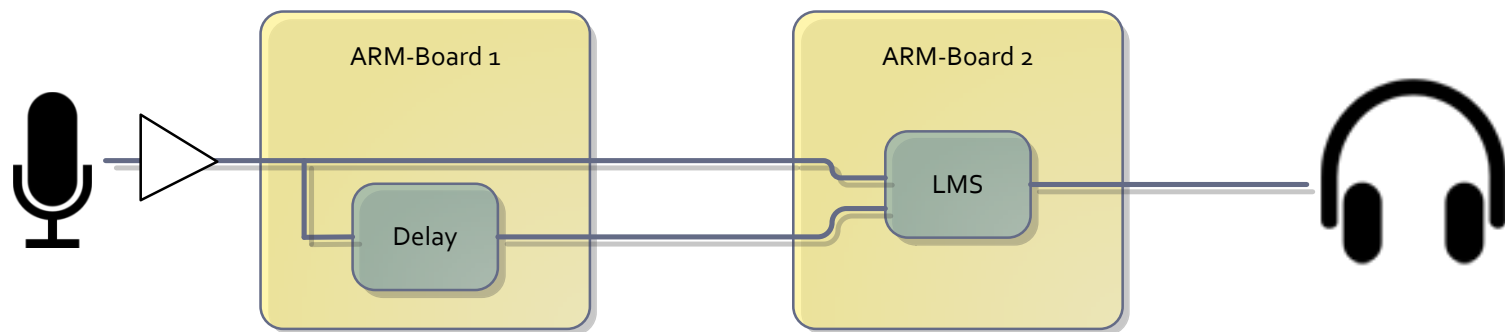
Resultate HL (voice)

- Filter mit richtigem Echo-Signal initialisieren:
 - Bleibt stabil
 - Emulation einer nahezu perfekten echo-cancellation: [MATLAB DEMONSTRATION]



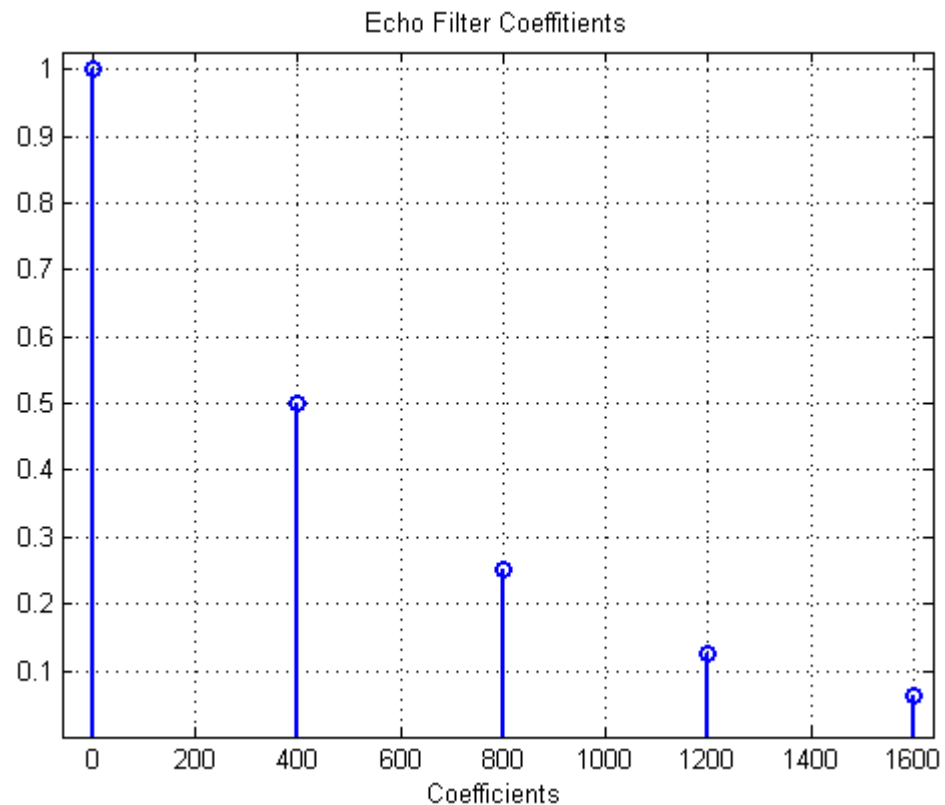
Implementation LL

- 2 ARM Boards + Ivo's Audio-Extension-Boards
 - Je eins für Delay und LMS
- Headset
- Mikrofon Preamp



Implementation Delay

- FIR-Filter (Raum-Simulation, «Reverb»)
- Max. 200ms \rightarrow NFIR = 1600 mit $f_s = 8\text{kHz}$
- `arm_fir_q15`
- funktioniert!



Implementation LMS

- Samplebasiert (DMA-Blocksize = 1)
 - Skalarprodukt des Eingangs und der Filterkoeffizienten

$$\hat{y}(n) = \sum_{i=0}^p w(i)x(n-i)$$

- Funktioniert noch nicht → Zeitmangel
- ABER: CMSIS DSP Library enthält bereits eine optimierte LMS-Implementation

Resultate LL

- Nur akustischer Test
- Built-In LMS-Funktion gut, aber:
 - 1600 Filtertabs * 8000 S/s
 - 1 cycle für MAC (Filter-Faltung)
 - 2 cycles für skalare Multipl. (Filter-Update)
 - Min. 3 cycles für Speicherzugriffe und anderes
 - ≈ 80 Mio Instr/s \rightarrow nur noch Faktor 2 von Systemtakt entfernt!!
- Jedoch schon mit einem sehr kleinen μ gute und schnelle Resultate

Schlussfolgerung

- Technische Schlussfolgerung
 - Konzept ist grundsätzlich einfach
 - Adaptives Filterkonzept ist die Basis vieler weiterer Anwendungen
 - Implementation auf Low-Level anspruchsvoll, zeitaufwändiger als gedacht
 - Stückweise Implementation durchaus sinnvoll ;)

Schlussfolgerung

- Persönliche Schlussfolgerung
 - Viel gelernt, Eigeninitiative wurde gefördert
 - Praxisbezug des DSV-Gebiets
 - Low-Level Implementation hat geholfen ein Gefühl dafür zu erhalten, wie viel Performance möglich ist und wo die Grenzen liegen

Demonstration

- Echo-Generierung
- Filterung mit built-in CMSIS Bibliothek LMS-Routine

Fragen?

