1. Software Implementation

Ziel der Projektarbeit ist es, ein Demo-Programm zur Visualisierung adaptiver Filter zu erarbeiten. Dieses Kapitel erläutert, wie das im Kapitel X erläuterte Konzept basierend auf der in Kapitel Y beschrieben Theorie implementiert wird. Auf der Software basierende Testfälle werden in Kapitel Z beschrieben.

1.1. Übersicht

Das Demo-Programm zeigt, wie mit einem adaptiven Filter ein Störton entfernt werden kann. Adaptive Filter spielen in verscheiden Bereichen der Signalverarbeitung eine wichtige Rolle und kommen insbesondere auch in Modems vor.

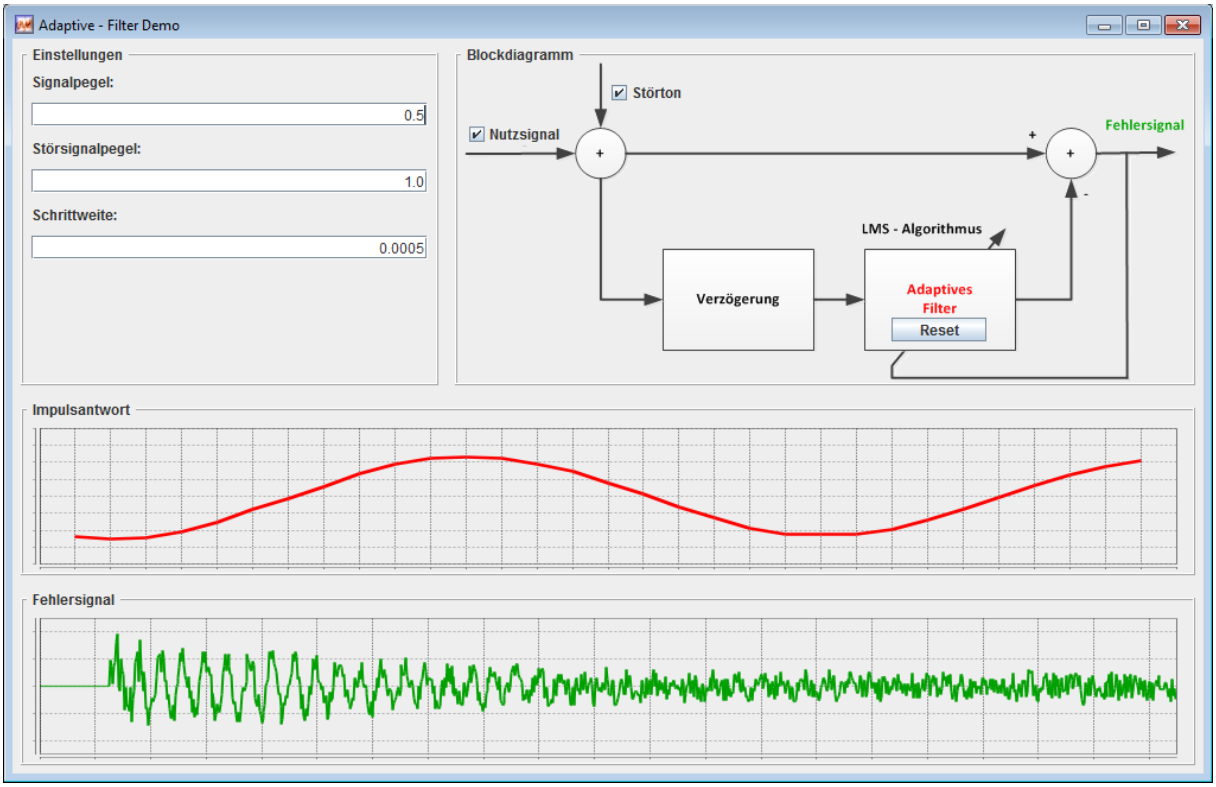

Abbildung 1: Gliederung des Programms *Adaptive - Filter Demo*.

Die Applikation ist im klassischen Model - View/Controller Entwurfsmuster programmiert [Referenz auf Klassendiagramm]. Die *View* ist in vier Panel gegliedert [Abbildung 1: Gliederung des Programms Adaptive - Filter Demo.]: Das *BildPanel* zeigt das zur Identifikation zugehörige Blockdiagramm und erlaubt die Signale ein- und auszuschalten sowie das adaptive Filter zurückzusetzten. Das *ParameterPanel* beinhaltete drei Textfelder zur Definition der Parameter. *ImpulsPlotPanel* und *FehlerPlotPanel* dienen der Darstellung der Impulsantwort des adaptiven Filters und des Fehlersignals. Das *Model* hat eine *SignalQuelle*, eine Verzögerung und ein *LMSFilter*. Die *SignalQuelle* erzeugt das mit einem Störton gestörte Nutzsignal und erbt zu diesem Zwecke von der Klasse *Thread*. Sie ruft in regelmässigen Abständen via *SignalListener* die Methode *processSignal(signal : double[])* mit einem Block von digitalen Signalwerten auf. Die Signalwerte werden dann, zur Estimation des Störtons

durch die Verzögerung und durch das *LMSFilter* geschickt und letzteres mittels Fehlersignal adaptiert. Der *Controller* delegiert im Wesentlichen die Aufgaben ans Model, welches wiederum via *Observer* - *Observable* Entwurfsmuster zum Aufdatieren der *View* führt.

1.2. View:

Die *View* ist als *GridBagLayout* organisiert und enthält die vier erwähnten Panels.

Das *BildPanel* hat den *Controller* und ist im Null-Layout organisiert. Es enthält das Blockdiagramm als Bild und zeichnet es in seiner natürlichen Grösse. Die bevorzugte Grösse des *BildPanel* wird mittels *setPreferredSize*() gleich der Grösse des Bildes gesetzt. Das *BildPanel* hat weiter zwei *JCheckboxen* und einen *JButton*. Jedes GUI-Element hat den *ActionListener* des *BildPanel*. Bei den entsprechenden Ereignissen werden die zugehörigen Methoden im *Controller* aufgerufen.

Das *ParameterPanel* hat den *Controller* und ist als *GridBagLayout* organisiert. Es enthält die drei Label und die zugehörigen Textfelder. Es verfügt über ein leeres *JLabel* um den überflüssigen Raum aufzunehmen. Jedes Textfeld hat den *ActionListener* des *ParameterPanel*. Bei den entsprechenden Ereignissen werden die zugehörigen Methoden im *Controller* aufgerufen.

Das *ImpulsPlotPanel* …

Das *FehlerPlotPanel* …

1.3. Controller:

Der *Controller* delegiert die entsprechenden Aufgaben an das Model und umfasst neben dem Konstruktor vier Methoden, wobei deren drei den Aufruf lediglich weiterleiten. Einzig die Methode *setParameter()* liest erst die entsprechende Information aus den Textfeldern aus und ruft dann die zugehörige Methode des Models auf.

1.4. Model:

Das *Model* hat wie eingangs erläutert eine *SignalQuelle*, einen *Delay* und ein *LMSFilter* sowie einige Attribute und Methoden. Der Methode *processSignal(signal : double[])* kommt dabei die zentrale Bedeutung zu. Sie realisiert die im Blockdiagramm gezeigte Struktur.

Die *SignalQuelle* hat einen Frame - Buffer, der jeweils via *SignalListener* an die Methode *processSignal(signal : double[])* weiter gegeben wird. Die entsprechenden Werte werden in der Methode *run()* in einer Endlos - Schleife generiert wobei der zugehörige *Thread* danach jeweils wieder mittels *sleep*() schlafen gelegt wird.

Das *FIRFilter* kapselt den Zustand und die Koeffizienten *coeffs* des Filters. Der Zustand ist als Zirkulärer-Buffer organisiert. Er enthält den momentanen und die vergangenen Signalwerte. Jeder neue Signalwert wird beim Aufruf der Methode *filter*() des *FIRFilters* in den Zirkulären-Buffer geschrieben und danach die Faltung berechnet. Die Faltung wird allgemein geschrieben als [Referenz auf die entsprechende Formel im Theorie - Kapitel]:

$$y[n]=\sum_{k=0}^{N} x[n-k]\cdot h[k]$$

wobei y[n] das Ausgangssignal zur Zeit n, x[n-k] den Eingangswert zur Zeit (n-k) und h[k] die Impulsantwort des Filters beschreibt. Die Berechnung der Faltung in der Methode *filter*() kann wie folgt skizziert werden:

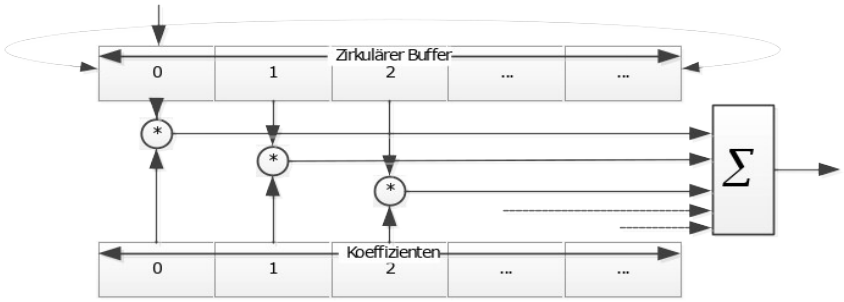

Abbildung 2: Struktur des zirkulären Buffers.

Die Berechnung für jeden Signalwert x[n] erfolgt dann im Model. Das heisst die Methode *filter*() berechnet nur die Summe über die Produkte zwischen Zirkulärem-Buffer und Koeffizienten.

Das *LMSFilter* erweitert das *FIRFilter* um die Fähigkeit sich zu adaptieren. Dazu wird für jeden der Koeffizienten die Update - Beziehung, die vektoriell geschrieben wird [Referenz …], berechnet:

$$\vec{h}[n]=\vec{h}[n-1]+e[n]*\mu*\vec{x}[n]$$

Die Berechnung des Updates in der Methode *lms*() kann wie folgt skizziert werden:

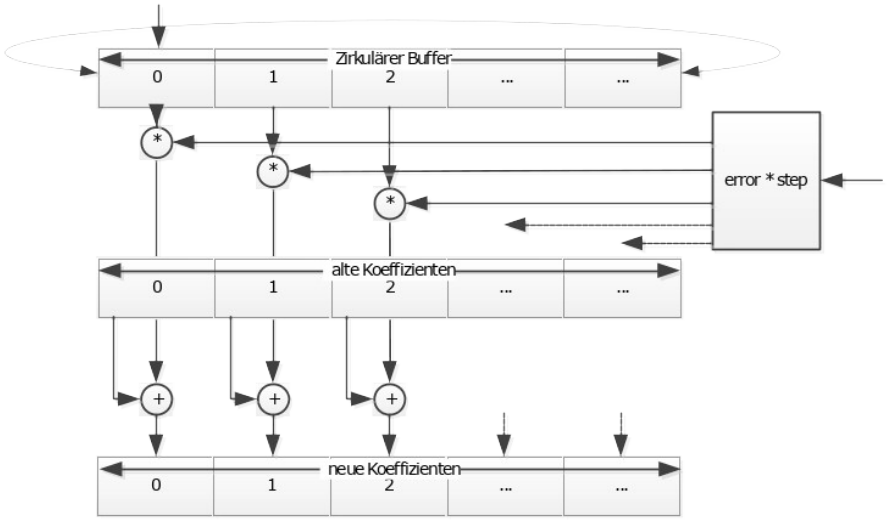

Abbildung 3: Struktur des LMS – Algorithmus.

Das *Model* …

1.5. Benutzungsfälle (Use-Cases):

(Hier beschreiben Sie lediglich ein Use - Case)

Use-Case: Der Benutzer gibt einen andern Wert für die Schrittweite ein:

- Enter im entsprechenden Textfeld löst die Methode *actionPerformed()* im *ParameterPanel* aus.
- Die Methode *actionPerformed()* ruft die Methode *setParameter*() des Controllers auf.
- Die Methode *setParameter*() liest die entsprechende Information aus dem GUI und ruft damit die Methode *setParameter*() des Models auf.
- Die Methode *setParameter*() des Models setzt die entsprechenden Parameter.
- Die darauf folgende Verarbeitung des Signales wird mit den neuen Werten ausgeführt und löst wie üblich via die Methode *notifyObservers*() den Update aus.
- Die Methode *update*() der View ruft die entsprechenden *update*() der darauf liegenden Panel auf.
- Die Methode *update*() der Panel holen sich die benötigte Information via Getter - Methoden vom Model und bringen sie zur Darstellung.

Die weiteren Use-Cases werden sinngemäss abgearbeitet.

Bei den Erläuterungen wird allenfalls auf die zugehörige Benutzerschnittstelle (Bild) und das Klassendiagramm Bezug genommen.