OOP2

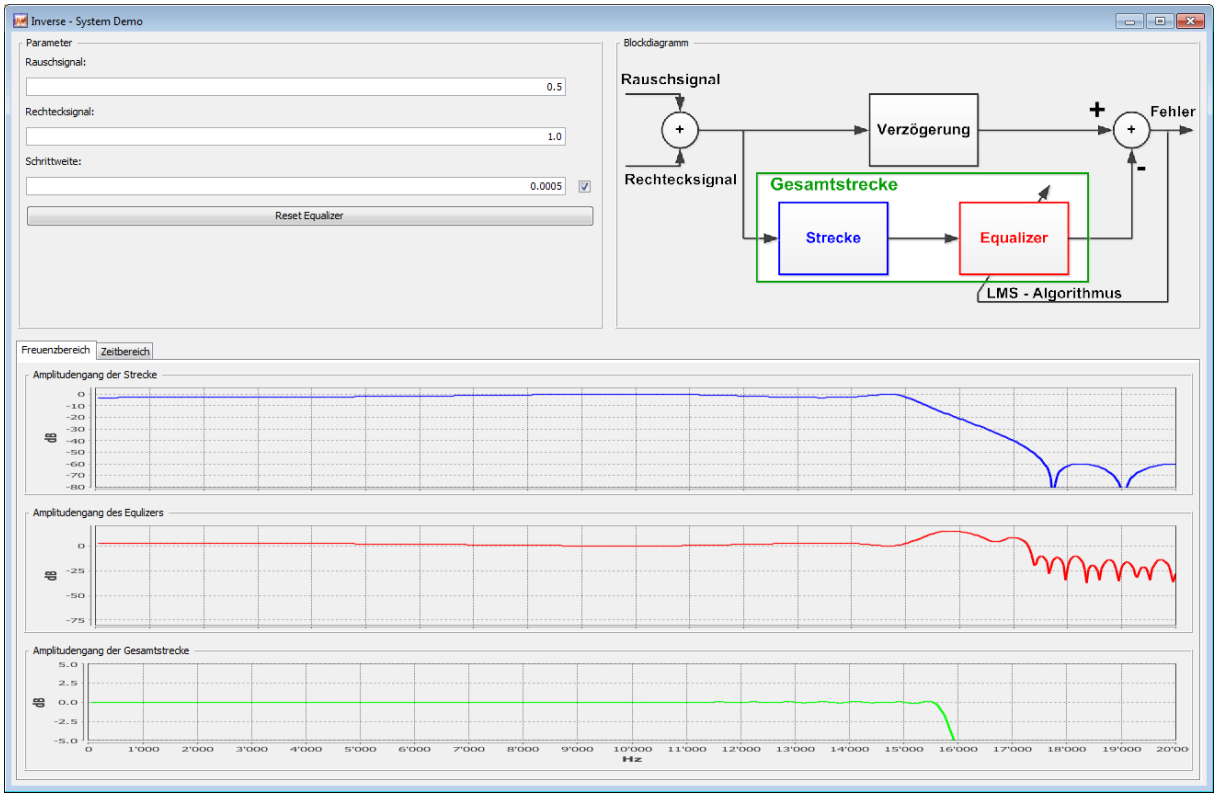
PRÜFUNG I

Bedingungen:

- Erlaubte Hilfsmittel: Unterrichtsunterlagen, Java Buch und Übungen.
- **Die Prüfung ist schrittweise, gemäss Aufgabenstellung lokal auf Ihrem Computer zu lösen. Kopieren sie zu diesem Zwecke den gesamten Ordner *P1_AdaptiveFilterDemo_FS17_Vorlage* auf Ihre lokale Harddisk und importieren sie das Projekt in Eclipse. Am Ende der Prüfung ist der Ordner *src* umbenannt in *NameVorname* abzugeben.**
- **Setzen sie als erstes Ihren Namen und Vornamen in die Dateien.**
- **Gegenseitiges Abschreiben in irgendeiner Form führt zur Note 1!**
- **Folgend sie bei der Wahl von Variablen exakt den Angaben in der Aufgabenstellung.**
- **Die Beilage mit dem Layout muss unterschrieben zurückgegeben werden!**

Beschreibung:

Ziel dieser Prüfung ist es, ein Programm zur Visualisierung des Adaptionsvorganges bei der inversen Systemmodellierung zu schreiben. Adaptive Systeme erlauben bei der digitalen Signalverarbeitung die Anpassung an ein unbekanntes oder sich änderndes System. Bei der inversen Systemmodellierung wird ein *Equalizer* der Art eingestellt, dass er den Amplitudengang und die Gruppenlaufzeit einer unbekannten Strecke egalisiert.



Die Applikation ist im klassischen Model - View/Controller und Observable/Observer Entwurfsmuster, gemäss Klassendiagramm in der Beilage, implementiert.

Die *View* beinhaltet das *ParameterPanel* zur Einstellung der Parameter, das *BDGPanel* mit dem Blockdiagramm sowie ein *JTabbedPane ReiterPanel* mit der *ViewFrequenzbereich* zur Darstellung aller drei Amplitudengänge und der *ViewZeitbereich* mit der Darstellung aller drei Impulsantworten. Amplitudengänge und Impulsantworten werden mittels der Klasse *XYPlot* repräsentiert.

Der *Controller* liest die Information aus dem *ParameterPanel* aus und delegiert allfällige Aufgaben an das *Model*. Der *Controller* verfügt dazu sowohl über die *View* wie auch über das *Model*.

Das *Model* beinhaltet die Signalquelle, die mittels *ScheduledExecutorService* in regelmässigen Abständen die Methode *processSignal*() des *Models* aufruft. Es verfügt weiter, entsprechend dem Blockdiagramm, über die Verzögerung, das Strecken-Filter in der Form eines *IIRFilters* und den Equalizer in der Form eine *LMSFilters*. *IIR*- und *FIRFilter* basieren auf *Filter*, das *LMSFilter* ist ein Derivativ der Klasse *FIRFilter*. Für die notwendigen Berechnungen stehen weiter die Klassen *Matlab* und *PicoMatlab* zur Verfügung, die je statische Methoden beheimaten.

Die Aufgabenstellung führt Sie schrittweise zum Ziel. Folgen Sie daher beim Lösen der Prüfung der Aufgabenstellung.

Achten Sie darauf, dass Ihr Code kompilierbar bleibt!

Aufgabe 1: Aspekt View: Klassen des User - Interfaces (~ 76 Pte.)

View:

Die View ist wie beschrieben aufgebaut und erlaubt mittels Reiter zwischen der Darstellung des Frequenzbereichs und des Zeitbereichs zu wechseln. Allfällige seitens des Models gibt die via entsprechend weiter.

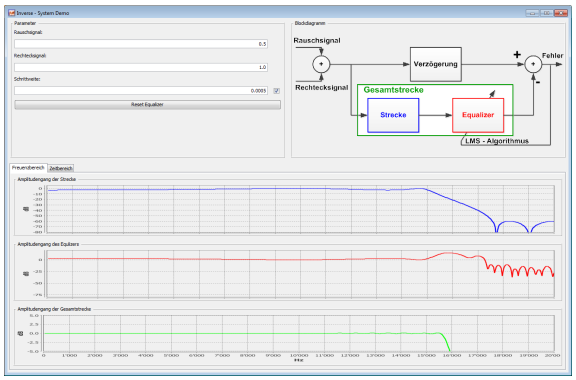


Abbildung 1: *Frequenzbereich*

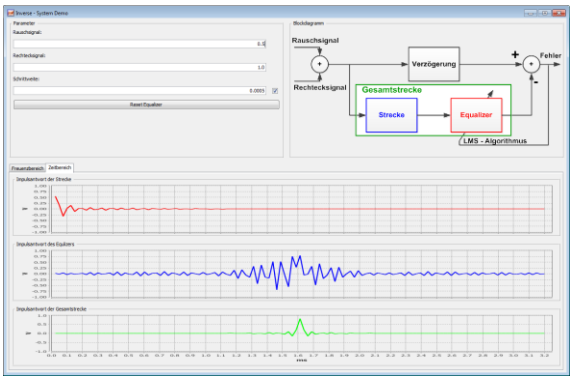


Abbildung 2: *Zeitbereich*

a) Implementieren Sie die Klassen *View*, *ParameterPanel*, *ReiterPanel*, *ViewFrequenzBereich*, und *ViewZeitBereich* gemäss Dokumentation und Beilage. Die Klasse *BDGPanel* ist bereits gegeben.

Aufgabe 2: Aspekt Controller: *Klasse zu Steuerung* (~ 8 Pte.)

Controller: Der *Controller* leitet allfällige Aufgaben an das *Model* weiter. Die allenfalls benötigten Informationen werden vom *Controller* via *View* ausgelesen.

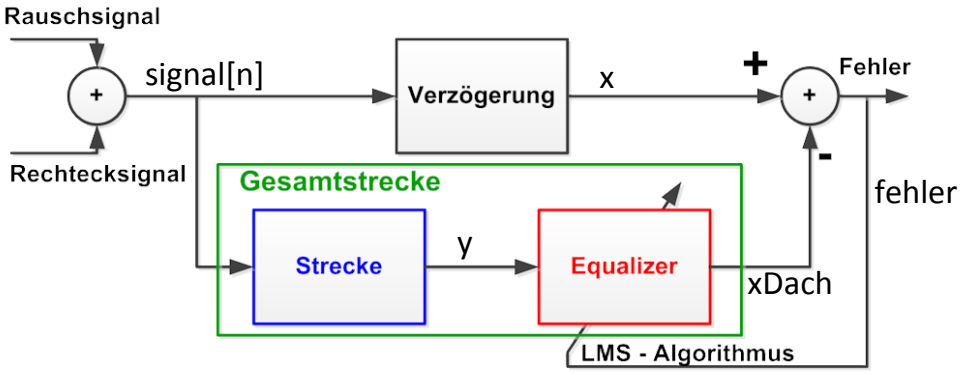
a) Implementieren Sie die Klasse *Controller* gemäss Dokumentation und Pseudo-Code.

Aufgabe 3: Aspekt Model: Klassen zur Berechnung (~ 38 Pte.)

Model: Das *Model* hat, wie eingangs erwähnt, die *SignalQuelle* sowie die Elemente des Blockdiagramms. Notwendige Berechnungen werden allenfalls mit den Klassen *Matlab* und *PicoMatlab* durchgeführt.

a) Implementieren Sie die Klasse *Model* gemäss Beschreibung und Pseudo-Code.

Hinweise:
- Schauen Sie sich die notwendigen Methoden in *Matlab* und *PicoMatlab* an, um deren Verwendung zu verstehen.
- Methode *processSignal*(): Implementieren Sie die Signalverarbeitung gemäss nachfolgendem Blockdiagramm und verwenden Sie gegebenen Variablenbezeichnungen:



b) Implementieren Sie die Klasse *SignalQuelle* gemäss Beschreibung und Pseudo-Code.

Hinweis:
Methode *start*(): Mittels *service. scheduleAtFixedRate(Runnable command, long initialDelay, long period, TimeUnit unit)* lässt sich ein *Runnable* nach einer einmaligen Verzögerung mit fixer Periode wiederholt abarbeiten. In unserem Falle ist die *Signalquelle* selber *Runnable* und der Start soll nach 100ms erfolgen. Nachher soll alle sleepTime ms *run*() ausgeführt werden. Mit *TimeUnit*. kann die Zeiteinheit gewählt werden.

Aufgabe 4: Clean Up (~ ? Pte)

Gelegentlich geht etwas vergessen …

a) Ergänzen Sie den Code um noch allfällig fehlenden Code.

Aufgabe 5: Challenge League (5 Pte)

a) Ändern Sie den Controller der Art, dass bei einer Fehleingabe in einem der Textfelder der folgende Dialog erscheint: