

Thomas Lohse
Matrikel-Nr. 11777
SG 99/041/01
Allgemeine Informatik
HTW Dresden

Simulation adaptiver Rauschunterdrückung

Semesterbeleg im Fach Neuroinformationsverarbeitung

Ort, Datum: Dresden, den 31.07.2002

Inhaltsverzeichnis

1 Motivation	2
2 Grundidee eines neuronalen, adaptiven Filters	3
3 Simulation eines neuronalen, adaptiven Filters	4
3.1 Filtergraph.....	4
3.1.1 <i>Signal-/Rausch-Quellen</i>	5
3.1.2 <i>Mischer</i>	5
3.1.3 <i>Adaptiver Filter / Neuronale Netze</i>	6
3.2 Programmbedienung	6
3.3 Simulationsergebnisse.....	9
4 Technische Dokumentation	10
4.1 Systemanforderungen	10
4.2 Struktur des Projekt-Verzeichnisses	10
4.3 Bau der Applikation.....	11
4.4 Beschreibung der Systemarchitektur.....	11
4.5 Erweiterungsmöglichkeiten der Simulationsumgebung	13

1 Motivation

In der Signalverarbeitung steht man oft vor dem Problem, dass ein Signal bei der Wandlung oder Übertragung durch Störgrößen verfälscht wird und durch Einsatz von Filtern entstört werden muss. Ein typisches Beispiel dafür ist die Übertragung der Sprache von Piloten in einem Flugzeug, deren Sprachsignal stark durch Triebwerksgeräusche gestört wird.

Zur Realisierung notwendiger Entstörungsfilter stehen verschiedene Technologien zur Verfügung. Ein besonders interessanter Ansatz ist aber die Anwendung neuronaler Netze für die Filterung. Aufgrund der Lernfähigkeit neuronaler Netze ist es nicht erforderlich, dass der Zusammenhang zwischen Störgröße und Signalverzerrung mathematisch genau beschrieben werden kann. Außerdem bieten Filter mit neuronalen Netzen die Möglichkeit, sich dynamisch an veränderbare Störeinflüsse anzupassen.

Gegenstand dieser Arbeit ist die Simulation eines einfachen, adaptiven Signalfilters mit neuronalem Netz als Anwendungsgebiet neuronaler Netze. Dazu wird im folgenden Abschnitt zunächst die prinzipielle Funktionsweise eines solchen Filters erläutert. Die sich anschließenden Abschnitte befassen sich mit der Anwendung und den technischen Details der softwaremäßigen Implementierung des Filters und der notwendigen Simulationsumgebung zum Test und zur Veranschaulichung der Filterwirkung.

2 Grundidee eines neuronalen, adaptiven Filters

Ein adaptiver Filter besitzt neben dem Eingang für das gestörte Signal und dem Ausgang für das entstörte Signal einen zusätzlichen Eingang für die Messwerte der Störgröße. Diese Konfiguration hat zwar den Nachteil, dass zusätzlich die Störgröße (z. B. der Triebwerkslärm, s. Abschnitt 1) durch irgend eine korrelierte Messgröße erfasst werden muss, bietet dafür aber auch die Möglichkeit, Störungen im selben Frequenzband wie das Nutzsignal und mit deutlich größerer Amplitude als das Nutzsignal zu unterdrücken. Außerdem ist der Messwert der Störgröße Voraussetzung für die dynamische Anpassung an einen veränderlichen Einfluß der Störgröße.

Der einfachste und im Rahmen dieses Projektes betrachtete Fall ist die lineare Überlagerung von Nutzsignal und durch die Störgröße erzeugtem Rauschen:

$$S' = S + w * N$$

S'	... Pegel des gestörten Nutzsignals
S	... Pegel des ungestörten Nutzsignals
N	... Störgröße
w	... Koeffizient zur Kennzeichnung des Störgrößeneinflusses

Ist der Störgrößenkoeffizient bekannt, so kann der Filter das unverfälschte Signal durch einfache Differenzbildung wiederherstellen:

$$S = S' - w * N$$

S'	... Pegel des gestörten Nutzsignals
S	... Pegel des ungestörten Nutzsignals
N	... Störgröße
w	... Koeffizient zur Kennzeichnung des Störgrößeneinflusses

Die Trivialität dieses Problems verschwindet sobald sich der Koeffizient w zeitlich in komplexer Weise verändert. Dann ist eine dynamische Anpassung des Filters notwendig.

Durch Einsatz eines neuronalen Netzes lässt sich dieses Problem auf elegante Weise lösen. Das zu verwendende Netz besitzt nur ein Eingangs- und ein Ausgangsneuron. Als Input erhält es den Messwert N der Störgröße als Output liefert es einen „Schätzwert“ für den Rauschpegel $w * N$, den der Filter zur Entstörung vom Eingangssignal subtrahieren muss. Um dem Netz den Einfluß der Störgröße zu lernen, wird es online mittrainiert, d.h. parallel zum Filterbetrieb. Dazu schließt sich an jeden Recall, d.h. der Abfrage des Rauschpegels zum gegebenen Messwert der Störgröße, ein Lernschritt an. Für den Lernschritt wird ebenfalls der Messwert der Störgröße als Input verwendet, als teaching point wird der Pegel des gestörten Nutzsignals (Signaleingang des Filters) benutzt. Wenn das Netz richtig konditioniert ist, lernt es auf diese Weise den Zusammenhang zwischen Störgrößemesswert und Rauschpegel und nur diesen Zusammenhang. Über den eigentlichen Nutzsignalanteil erhält das Netz keine Information, deshalb bleibt der Nutzsignalanteil für das Netz nichtdeterministisch und erscheint als „Rauschen“ das nicht mitgelernt wird (bei richtiger Konditionierung!).

Diese Art des Online-Learnings macht den Filter adaptiv, d. h. das neuronale Netz und damit der Filter passen sich automatisch an den Einfluss der Störgröße auf das Nutzsignal an, ohne das irgend ein manueller Eingriff in den Filter notwendig wäre.

Die Schwierigkeit besteht in der Auswahl eines geeigneten Netzes und der Einstellung der Netzparameter. Denn das Netz soll einerseits besonders gut generalisieren, d.h. ausschließlich den Zusammenhang zwischen Störgröße und Rauschpegel erlernen, andererseits aber auch schnell auf zeitliche Änderungen des Störgrößeneinflusses reagieren. Und diese beiden Forderungen sind zum Teil widersprüchlich.

3 Simulation eines neuronalen, adaptiven Filters

Im Rahmen dieser Arbeit wurde eine JAVA-Anwendung zur Simulation des in Abschnitt 2 beschriebenen neuronalen, adaptiven Filters entwickelt. Diese Anwendung besteht aus der Implementierung des adaptiven Filters mit einer Auswahl verschiedener neuronaler Netzwerke und aus einer Simulationsumgebung, die den Filter in ein Netzwerk signalerzeugender / -verarbeitender Komponenten integriert, um dessen Funktion zu testen. Das der Simulationsumgebung zugrunde liegende Netzwerk signalerzeugender / -verarbeitender Komponenten wird auch Filtergraph oder Signalprozessor genannt und ist in Abschnitt 3.1 beschrieben.

Die Anwendung bietet für die Konfiguration des Filtergraphen und seiner Komponenten und für die Steuerung des Simulationsablaufs eine grafische Oberfläche. Diese gestattet auch die grafische Darstellung des Signalverlaufs an verschiedenen Stellen im Filtergraphen. Die Bedienung der Anwendung und die Durchführung von Simulationsläufen ist in Abschnitt 3.2 beschrieben.

In Abschnitt 3.3 werden kurz die Erkenntnisse zusammengefasst, die aus Experimenten mit der Simulationsumgebung gezogen werden können. Es werden Konfigurationen des Filtergraphen dargestellt, die besonders gut funktionieren, aber auch jene, die problematisch sind.

3.1 Filtergraph

Wie schon bemerkt, sind zur Darstellung und Untersuchung des Verhaltens eines adaptiven Filters weitere Komponenten notwendig, wie z. B. Signal- und Rauschquellen. Diese werden zusammen mit dem Filter in einem Netzwerk integriert. Das der Simulationsanwendung zugrunde liegende Netzwerk, der Filtergraph, ist in Abbildung 3.1 dargestellt.

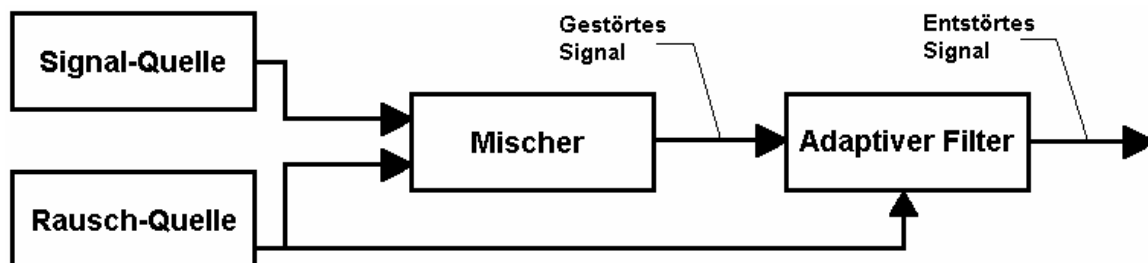


Abb. 3.1 Filtergraph der Simulationsumgebung.

Die Simulationsumgebung enthält für alle in Abbildung 3.1 dargestellten Filtergraph-Komponenten verschiedene Implementierungen, die beliebig kombiniert werden können, um das Verhalten des adaptiven Filters zu studieren.

Alle Komponenten erwarten als Input (sofern ein Eingang vorhanden) Werte zwischen 0 und 1. Ebenso liefern alle Komponenten, mit Ausnahme des adaptiven Filters, Output-Werte zwischen 0 und 1. Aus technischen Gründen ist es dem adaptiven Filter nicht möglich, den korrekten Faktor für die Normierung des Outputs zu bestimmen. Es besteht aber die Möglichkeit über das Gain-Attribut des Filters dessen Ausgangspegel manuell zu skalieren.

In den folgenden Unterabschnitten werden kurz die verschiedenen in der Simulationsumgebung enthaltenen Implementierungen der einzelnen Filtergraph-Komponenten vorgestellt.

3.1.1 Signal-/Rausch-Quellen

Für beide Quellen stehen die gleichen zwei Generator-Typen zur Verfügung:

a) Sine

... generiert ein Sinussignal, das eine primitive Periode von 6,28 Zeiteinheiten (ZE) besitzt. Nach einem Reset des Generators beginnt das Sampling zum Zeitpunkt 0.

b) Random

... die Sample-Werte dieses Generators sind gleichverteilte Zufallszahlen zwischen 0 und 1. Der Generator wird bei Erzeugung mit der Systemzeit initialisiert. Ein Reset des Generators hat keinen Effekt.

Die Sampling-Rate aller Generatoren wird nach der Erzeugung automatisch auf 10 / ZE eingestellt.

3.1.2 Mischer

Aufgabe des Mixers ist es, die Störung des Signals durch eine Störquelle zu simulieren. Der Mischer kombiniert dazu den Output der Signal- und der Rausch-Quelle in einer spezifischen (möglicherweise zeitlich veränderlichen) Art und Weise. In der Simulationsumgebung stehen zwei Mischer-Typen zur Verfügung:

a) LinearSuperpos

... realisiert eine lineare Überlagerung von Signal und Rauschen entsprechend folgender Formel:

$$O = (1/4) * (S + 3 * R)$$

O ... Mischer-Output
S ... Input von der Signalquelle
R ... Input von der Rauschquelle

b) SquaredSuperpos

... dito LinearSuperpos, allerdings wird der Output noch quadriert.

3.1.3 Adaptiver Filter / Neuronale Netze

Der adaptive Filter ist eine nicht austauschbare Komponente, die aber hinsichtlich des verwendeten neuronalen Netzes parametrisiert ist, d.h. das neuronale Netz ist austauschbar. Die Simulationsumgebung stellt verschiedene neuronale Netze zur Auswahl bereit. Allen gemeinsam ist, dass es zweistufige, vollverbundene Feed-Forward-Netze mit einem Input- und einem Output-Neuron sind. Die Schwellwerte der Aktivierungsfunktionen werden durch Bias-Neuronen realisiert und das Training der Netze erfolgt entsprechend dem Error-Backpropagation-Algorithmus.

a) 1-5-1_logistic

... Feed-Forward Netz mit fünf Neuronen in der Hidden-Schicht. Als Transferfunktion der Hidden- und Output-Schicht dient die Logistische Funktion:

$$y = 1 / (1 + \exp(-a * x)).$$

Als Transferfunktion für die Input-Schicht, sowie als Ausgabefunktion aller Schichten dient die Identische Abbildung.

b) 1-10-1_logistic

... dito 1-5-1_logistic, allerdings mit zehn Neuronen in der Hidden-Schicht.

c) 1-5-1_tanh

... Feed-Forward Netz mit fünf Neuronen in der Hidden-Schicht. Als Transferfunktion der Hidden- und Output-Schicht dient der Tangens hyperbolicus:

$$y = (1 - \exp(-a * x)) / (1 + \exp(-a * x)).$$

Als Transferfunktion für die Input-Schicht, sowie als Ausgabefunktion aller Schichten dient die Identische Abbildung.

d) 1-10-1_tanh

... dito 1-5-1_tanh, allerdings mit zehn Neuronen in der Hidden-Schicht.

Die Oberfläche des Simulationsprogramms erlaubt die Anpassung weiterer Parameter des adaptiven Filters und des neuronalen Netze (Parameter a der Aktivierungsfunktion, Lernrate des Netzes, Ausgangsverstärkung des Filters), siehe dazu Abschnitt 3.2.

3.2 Programmbedienung

Das Java-Programm kann entweder als Applikation durch Ausführung des JAR-Archivs *build/noisefilter.jar* gestartet werden oder als Applet, indem mit dem Appletviewer oder Java-fähigen Web-Browser die Seite *build/noisefilter.html* geöffnet wird (Systemanforderungen beachten, siehe Abschnitt 4.1). Sollten diese Dateien nicht existieren, können sie entsprechend den Angaben in Abschnitt 4.3 aus den Quellen erstellt werden.

In Abbildung 3.2 ist die Benutzeroberfläche des Programms dargestellt.

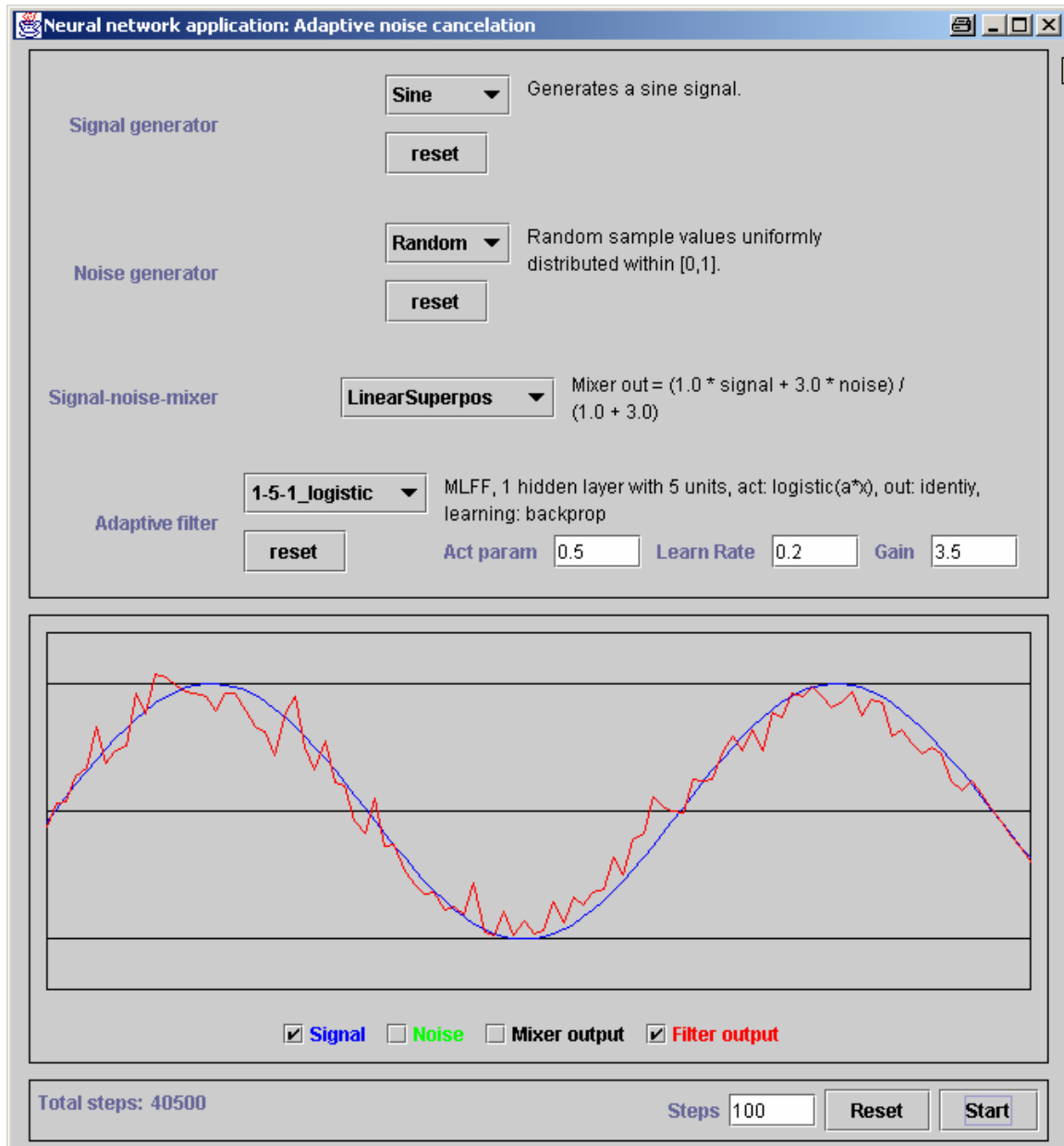


Abb. 3.2 Benutzeroberfläche des Simulators.

Die Oberfläche besteht aus drei Hauptkomponenten. Im oberen Teil befindet sich das Konfigurations-Panel, mit dessen Hilfe der Nutzer die Komponenten des Filtergraphen (siehe Abschnitt 3.1) manipulieren kann. Im mittleren Teil ist das Diagramm-Panel, mit dem der Signalverlauf an unterschiedlichen Punkten im Filtergraphen visualisiert werden kann. Und im unteren Teil des Fensters befindet sich das Kontroll-Panel, mit dem Simulationsprozesse gestartet und gesteuert werden können. Beendet werden kann die Anwendung nur über die entsprechenden Steuerelemente des Fensterrahmens.

a) Konfigurations-Panel

Das Konfigurations-Panel bietet für jede Komponente des Filtergraphen einen eigenen Abschnitt.

Mittels Select-Box kann für jede Filtergraph-Komponente eine Implementierung ausgewählt werden. Hinter der jeweiligen Select-Box wird eine kurze Beschreibung der ausgewählten Implementierung angezeigt. Die Reset-Buttons veranlassen ein individuelles Rücksetzen der jeweiligen Komponente. Ein Simulationsvorgang kann erst erfolgreich ausgeführt werden, wenn für jede Filtergraph-Komponente eine Implementierung ausgewählt wurde.

Für den adaptiven Filter können weitere Parameter eingestellt werden. Das Feld „Act param“ bestimmt den Wert des (möglicherweise vorhandenen) Parameters der Aktivierungsfunktion der Hidden- und Output-Schicht des verwendeten neuronalen Netzes. Im Falle der Logistischen Funktion oder des Tangens hyperbolicus ist dies der Koeffizient a (siehe Abschnitt 3.1.3). Das Feld „Learn rate“ dient der Einstellung der Lernrate des neuronalen Netzes. Beide Felder werden bei Auswahl eines neuen Netz-Typs mit den jeweiligen Standardwerten belegt. Das Feld „Gain“ bestimmt die Ausgangsverstärkung des adaptiven Filters und kann vom Nutzer zur Skalierung des Ausgangssignals benutzt werden, denn eine automatische Skalierung des Ausgangssignals auf das Intervall $[0,1]$ ist aus technischen Gründen nicht möglich.

b) Diagramm-Panel

Das Diagramm-Panel zeigt den zeitlichen Verlauf der Pegel am Ausgang des Signal- und Rausch-Generators, des Mischers und des adaptiven Filters an. Mittels der Checkboxes kann die Anzeige der einzelnen Pegelverläufe selektiv aktiviert bzw. deaktiviert werden. Standardmäßig ist nur die Anzeige des Signalverlaufs am Ausgang des Signal-Generators und des adaptiven Filters aktiviert. Die Farbe der Checkbox-Bezeichnungen entspricht der Farbe der jeweiligen Graphen im Diagramm.

Die Pegel 0, 0.5 und 1 sind im Diagramm durch waagerechte Achsen gekennzeichnet. Im Diagramm werden die Samples aller Schritte (siehe Kontroll-Panel) des letzten Simulationsvorganges angezeigt, d.h. die Zahl der Schritte eines Simulationsvorganges bestimmt die Einheit auf der Abszisse. Die Sample-Zeitpunkte werden bei hinreichend kleiner Schrittzahl durch senkrechte Achsen im Diagramm gekennzeichnet.

c) Kontroll-Panel

Mit dem Start-Button wird ein Simulationsvorgang gestartet. Die Länge des Simulationsvorganges wird durch die Schrittzahl im Steps-Feld festgelegt. Der Schritt ist die kleinste Einheit innerhalb eines Simulationsvorganges und bedeutet die Verarbeitung eines Samples der Signal- und Rauschquelle, d. h. von Signal- und Rauschquelle wird jeweils ein Sample geholt, beide Samples werden im Mischer kombiniert und das Ergebnis durch den adaptiven Filter propagiert.

Die Total-Steps-Anzeige summiert die Schrittzahl aufeinanderfolgender Simulationsvorgänge.

Mittels des Reset-Buttons kann der gesamte Filtergraph, d.h. alle Filtergraph-Komponenten, zurückgesetzt werden. Außerdem wird der Total-Steps-Zähler genullt und aus dem Diagramm-Panel werden alle Kurven gelöscht.

3.3 Simulationsergebnisse

Um die Qualität des adaptiven Filters beurteilen zu können, sollte im Diagramm-Panel nur die Anzeige der Pegel am Ausgang des Signal-Generators und des adaptiven Filters aktiviert sein. Dann lässt sich das Original-Signal gut mit dem entstörten Signal vergleichen. Am anschaulichsten stellt sich die Wirkung des adaptiven Filters dar, wenn als Signal-Quelle der Sine-Generator und als Rausch-Quelle der Random-Generator gewählt werden und als Mischer LinearSuperpos.

Aus durchgeführten Simulationen lassen sich folgende Erkenntnisse ziehen:

- 1) Wählt man für den adaptiven Filter ein neuronales Netz mit Logistischer Aktivierungsfunktion, so erzielt man eine gute Filterwirkung, wenn die Parameter „Act param“ auf 0.7, „Learn rate“ auf 0.2 und „Gain“ auf 3.5 eingestellt werden. Das gleiche gilt auch für neuronale Netze mit dem Tangens hyperbolicus als Aktivierungsfunktion, wenn der Parameter „Act param“ auf 0.5 vermindert wird. Der Output des adaptiven Filters unterscheidet sich dann vom Original-Signal des Signal-Generators nur noch durch die Überlagerung hochfrequenter Oberschwingungen mit kleiner Amplitude. Letztere könnten eventuell durch einen zusätzlichen Tiefpass-Filter beseitigt werden.
- 2) Die Geschwindigkeit, mit der sich die adaptiven Filter nach einem Reset wieder an den Einfluß der Störgröße anpassen, lässt sich durch Vergrößerung der Parameter „Act param“ bzw. „Learn rate“ verbessern. Allerdings geht das zu Lasten der Filterqualität, die Näherung des Original-Signals des Signal-Generators durch das Ausgangssignal des adaptiven Filters wird deutlich schlechter. Insbesondere lässt sich eine positive Phasenverschiebung des Filter-Outputs gegenüber dem Signal-Generator-Output konstatieren.
- 3) Filter mit neuronalen Netzen, deren Units den Tangens hyperbolicus als Aktivierungsfunktion besitzen, passen sich etwa um Faktor 10 – 15 schneller an den Einfluß der Störgröße an als Netze mit Logistischer Aktivierungsfunktion. Bezüglich der unter 1) genannten Einstellungen benötigt ein Filter, dessen Netz mit dem Tangens hyperbolicus als Aktivierungsfunktion arbeitet, nur 1000 – 1500 Schritte für die Anpassung. Hingegen benötigt ein Filter, dessen Netz mit Logistischer Aktivierungsfunktion arbeitet, etwa 15000 Schritte für die Anpassung.
- 4) Eine Erhöhung der Anzahl der Units in der Hidden-Schicht des neuronalen Netzes von fünf auf zehn bringt keine signifikante Verbesserung der Filterleistung.
- 5) Der adaptive Filter funktioniert auch noch bei Anwendung des SquaredSuperpos-Mischers, d. h. bei nichtlinearer Kombination von Signal und Rauschen. Allerdings ist die Leistung deutlich schlechter, denn das entstörte Signal weist hochfrequente Oberschwingungen mit relativ großer Amplitude auf.
- 6) Der Filter versagt, wenn als Signal-Quelle und als Rausch-Quelle jeweils der Random-Generator ausgewählt wird. Das „entstörte“ Signal besitzt in diesem Fall ein Vielfaches der Frequenz des Originalsignals. Als Ursache für dieses Verhalten wird die starke Korrelation zwischen Signal und Rauschen angesehen, so dass das neuronale Netz des adaptiven Filters nicht mehr zwischen Signal und Rauschen unterscheiden kann.

4 Technische Dokumentation

Das Simulationsprogramm ist eine komplett in Java geschriebene Applikation mit Swing basiertem Nutzerinterface. Es kann wie in Abschnitt 3.2 beschrieben als Stand-alone Applikation oder als Applet ausgeführt werden. Der Programmstart kann out-of-the-box erfolgen, es sind keine speziellen Konfigurationsarbeiten notwendig. Die Applikation führt keine Zugriffe auf das lokale Dateisystem aus und erzeugt auch keinerlei Konfigurationsdateien oder Einträge in Systemdatenbanken.

Die Systemarchitektur ist hinsichtlich einer einfachen Erweiterbarkeit der Simulationsumgebung um weitere Implementationen der Generator-, Mischer-, und Neuronales Netz – Komponenten konzipiert (siehe Abschnitt 4.5).

4.1 Systemanforderungen

- Betriebssystem mit grafischer Benutzeroberfläche
- Java-Runtime-Umgebung Version 1.2 oder größer / Web-Browser mit Java-Plugin Version 1.2 oder größer
- Java-Entwicklungs-Umgebung Version 1.2 oder größer (nur zum Bau der Applikation)

4.2 Struktur des Projekt-Verzeichnisses

Die folgende Tabelle gibt einen Überblick über den Inhalt der zum Projekt gehörigen Verzeichnisse und Dateien.

Tab. 4.1 Struktur und Inhalt des Projektverzeichnis.

Verzeichnis / Datei	Inhalt
<i>Makefile</i>	Makefile zum Bau der Applikation und der Klassen-Dokumentation (nur unter Unix verwendbar).
<i>build/noisefilter.jar</i>	Ausführbares JAR-Archiv, das die komplette Applikation enthält. Dieses Archiv ist auch für die Ausführung als Applet notwendig.
<i>build/noisefilter.html</i>	Startseite für die Ausführung des Simulationsprogramms als Applet.
<i>build/classes/**</i>	Baum aller zum Programm gehöriger Klassen.
<i>build/javadoc/**</i>	Javadoc generierte Dokumentation der Programm-Klassen.
<i>build/doc/*</i>	Handgeschriebene Projektdokumentation.
<i>src/**</i>	Source-Code und zugehörige Konfigurationsdateien.

4.3 Bau der Applikation

Im allgemeinen ist ein Neubau der Applikation nicht notwendig. Das Projekt-Verzeichnis enthält bereits im Verzeichnis *build* das ausführbare JAR-Archiv *noisefilter.jar*, die zur Ausführung als Applet benötigte Seite *noisefilter.html*, den Klassen-Baum und die komplette Javadoc generierte Dokumentation der Programm-Klassen.

Soll das Programm im Rahmen einer Erweiterung neu gebaut werden oder sollte entgegen aller Erwartung das *build*-Verzeichnis nicht vorhanden sein, so kann das *build*-Verzeichnis mittels *make* komplett neu erstellt werden (*make help* gibt eine kurze Beschreibung aller im *Makefile* verfügbaren Ziele aus). Die Generierung mittels *make* funktioniert aber leider nur auf einer Unix-Plattform.

4.4 Beschreibung der Systemarchitektur

Der Applikation liegt eine 2-Schichten-Architektur zu Grunde, die durch das Klassendiagramm in Abbildung 4.1 dargestellt ist. Die obere Schicht (View/Controller) enthält alle Klassen des Nutzerinterfaces und der Interaktions-Steuerung. Die untere Schicht enthält die Klassen des Filtergraphen (Model).

Die Klassen der Oberfläche befinden sich alle im Package *noisefilter.view*, die Klassen des Filtergraphen im Package *noisefilter.signalproc* und seinen Subpackages.

Die Klasse *Mediator* dient als Vermittler einerseits zwischen den Klassen der Oberfläche und andererseits zwischen den Oberflächenklassen und dem Filtergraphen. Sie wurde eingeführt, um die Komplexität des Beziehungsgeflechts zu begrenzen, die Oberflächenklassen voneinander zu entkoppeln und eine mögliche Erweiterung der Oberfläche zu erleichtern.

Die Klassen *Generator*, *Mixer* und *NeuralNet* sind Interfaces, die durch Klassen in den Packages *noisefilter.signalproc.generator*, *noisefilter.signalproc.mixer* und *noisefilter.signalproc.neuralnet* implementiert werden. Die Listen der verschiedenen Implementationen werden durch die Klasse *ComponentFactory* verwaltet und können durch Ergänzung neuer Interface-Implementationen erweitert werden (siehe dazu Abschnitt 4.5)

Für eine umfassende Dokumentation der Feinarchitektur der Anwendung sei der interessierte Leser auf die Javadoc generierte Klassendokumentation im Verzeichnis *build/javadoc* verwiesen.

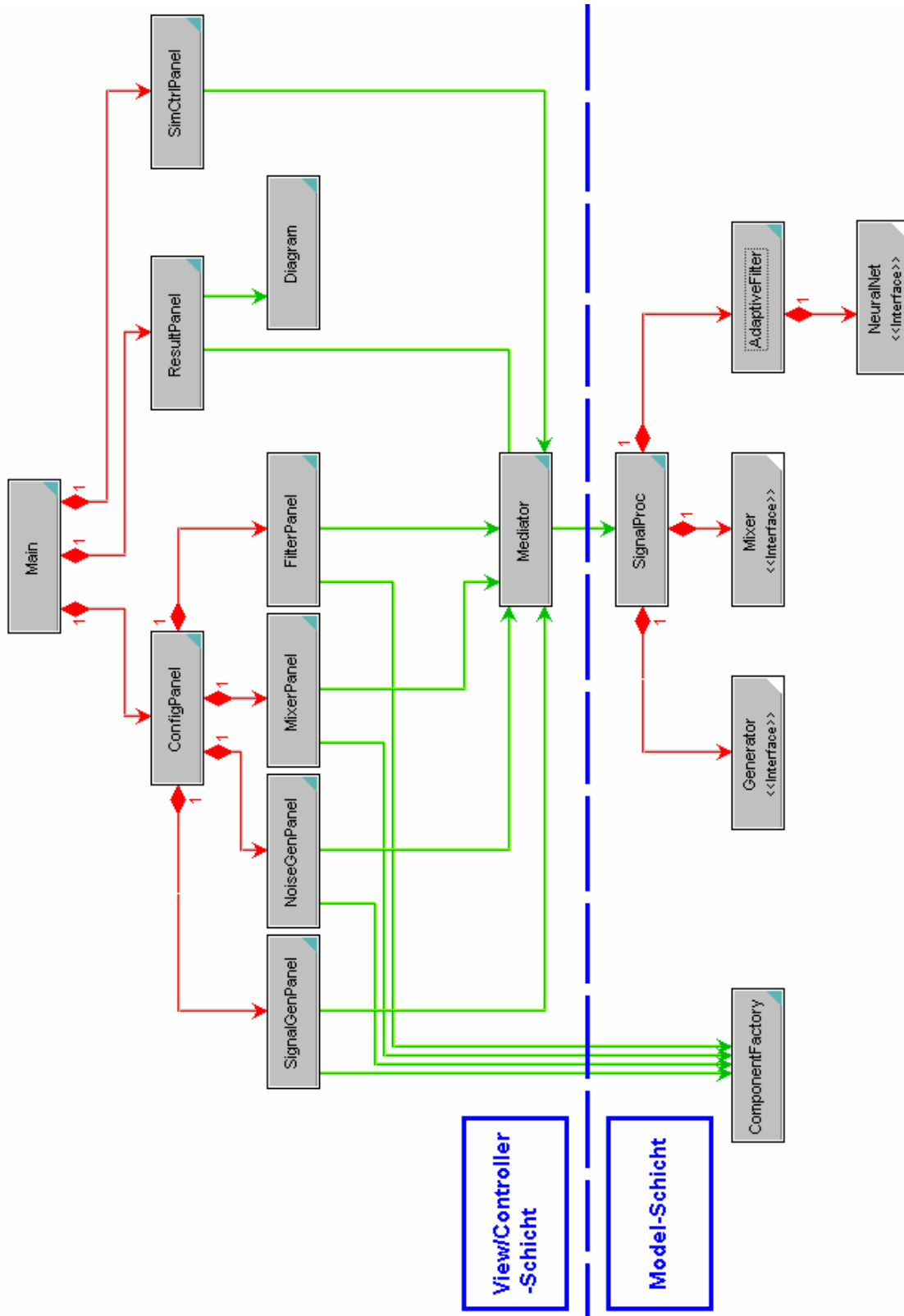


Abb. 4.1 Systemarchitektur der Anwendung.

4.5 Erweiterungsmöglichkeiten der Simulationsumgebung

Der Filtergraph, das Model, der Anwendung wurde so entworfen, dass auf einfachste Art und Weise neue Implementationen der Filtergraph-Komponenten hinzugefügt werden können.

Um z.B. einen neuronalen Netz-Typ zu ergänzen, muß nur eine neue Klasse zum Package *noisefilter.signalproc.neuralnet* hinzugefügt werden, die das Interface *noisefilter.signalproc.NeuralNet* implementiert. Dabei sollte man sich an der Implementierung der mitgelieferten Netz-Typen orientieren. Im Quellcode-Verzeichnis des Packages *noisefilter.signalproc.neuralnet* befindet sich eine Datei *net.list* in der alle *NeuralNet* – Implementationen registriert sind, damit sie von der Klasse *ComponentFactory* verwaltet werden können. Eine neue Implementation muss ebenfalls in dieser Datei eingetragen werden. Die Syntax ist in der Datei *net.list* beschrieben. Nachdem eine neue *NeuralNet*-Implementation hinzugefügt wurde, muss die Anwendung neu gebaut werden (siehe Abschnitt 4.3).

Die Ergänzung einer Generator-Implementierung oder einer Mischer-Implementierung geschieht auf analoge Weise. Tab. 4.2 fasst die notwendigen Informationen für die Implementierung einer Filtergraph-Komponente zusammen. Man beachte, dass keine zusätzlichen adaptiven Filter ergänzt werden können, sondern nur neuronale Netze, die der adaptive Filter dann benutzt.

Tab. 4.2 Rahmen für die Ergänzung neuer Implementierungen von Filtergraph-Komponenten. Package- und Verzeichnisangaben sind relativ zu *noisefilter.signalproc* bzw. *src/noisefilter/signalproc*.

Komponenten Typ	zu implement. Interface	Zielpackage	Registrierungsdatei
Generator	<i>Generator</i>	<i>generator</i>	<i>generator/gen.list</i>
Mischer	<i>Mixer</i>	<i>mixer</i>	<i>mixer/mix.list</i>
Neuronales Netz	<i>NeuralNet</i>	<i>neuralnet</i>	<i>neuralnet/net.list</i>