NEFCON-I

Hermann-Josef Diekgerdes

7. Februar 1994

Technische Universität Braunschweig
Institut für Rechnerverbund und
Betriebssysteme

Diplomarbeit

NEFCON-I: Entwurf und Implementierung einer Entwicklungsumgebung für Neuronale Fuzzy-Regler

von

cand.-inform. Hermann-Josef Diekgerdes

Aufgabenstellung und Betreuung:

Prof. Dr. R. Kruse und Dipl.-Inform. D. Nauck

Braunschweig, den 7. Februar 1994

Erklärung

Ich versichere, die vorliegende Arbeit selbständig und nur unter Benutzung der angegebenen Hilfsmittel angefertigt zu haben.

Braunschweig, den 7. Februar 1994

Abstract

To optimize fuzzy controllers they are often combined with neural networks. The disadvantage of these modells is that their learning algorithms don't retain the semantics of a fuzzy controller. In [Nauck und Kruse, 1992] a modell of a neural fuzzy controller is described that overcomes this disadvantage. It is possible to interpret this modell as a neural network with fuzzy sets as its weights as well as a fuzzy controller.

This thesis presents NEFCON-I, a simulation environment for neural fuzzy controllers, that implements the above described modell. It provides a graphical user interface to define the knowledge base of the fuzzy controller. Also the behavior of the controller is illustrated while it is controlling the dynamical system. NEFCON-I is able to manage arbitrary dynamical systems. As an example a simulation of an inverted pendulum has been implemented.

Kurzfassung

Zur Optimierung von Fuzzy-Reglern werden diese häufig mit Neuronalen Netzen kombiniert. Viele dieser Ansätze haben den Nachteil, daß ihre Lernresultate aus der Sicht der Fuzzy-Mengen-Theorie schwer zu interpretieren sind. Ein Modell, das diesen Nachteil umgeht, wird in [Nauck und Kruse, 1992] beschrieben. Es ist einerseits als Neuronales Netz mit Fuzzy-Mengen als Gewichten und andererseits als Fuzzy-Regler interpretierbar.

In dieser Arbeit wird NEFCON-I, eine Simulationsumgebung für Neuronale Fuzzy-Regler, vorgestellt, welches das oben beschriebene Modell implementiert. Es wird eine grafische Benutzeroberfläche in einer Unix-X11-Umgebung zur Verfügung gestellt, mit deren Hilfe die Wissensbasis des Reglers definiert werden kann. Außerdem wird das Verhalten des Reglers während des Regelungsvorgangs grafisch veranschaulicht. Mit NEFCON-I können beliebige technische Systeme geregelt werden. Als Beispiel für ein solches System, wurde die Simulation eines inversen Pendels implementiert.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

In der klassischen Regelungstechnik geht dem Entwurf eines Reglers die Erstellung eines mathematischen Modells für das zu regelnde System voraus. Das Modell liegt dann oft in Form einer Differentialgleichung vor und ist vor allem bei komplizierteren Systemen sehr komplex und dadurch schwer zu handhaben. Außerdem wird vorausgesetzt, daß die notwendigen physikalischen Kenntnisse über das zu regelnde System vorhanden sind. Ist dies nicht der Fall, muß auf den Einsatz klassischer Regler verzichtet werden.

Es ist jedoch auch möglich, komplexe technische Systeme zu beherrschen, ohne den physikalischen oder mathematischen Hintergrund zu kennen. Dies zeigt die Beobachtung, daß menschliche Regelungsexperten vielfach in der Lage sind, ihre Aufgabe zu erfüllen, ohne dabei im exakten mathematischen Modell zu denken. Sie können so auch Systeme regeln, für die ein solches Modell nicht angegeben werden kann.

Bei der kognitiven Regelung wird im Gegensatz zur klassischen Methode versucht, das Verhalten menschlicher Regelungsexperten nachzubilden. Es wird also kein Modell des zu regelnden Systems aufgestellt, sondern das eines menschlichen Bedieners.

In den letzten Jahren hat sich im Bereich der kognitiven Regelung die Theorie der Fuzzy-Mengen immer weiter durchgesetzt. Sie zeichnet sich dadurch aus, daß ein gewisser Anteil Impräzision, Vagheit und Unsicherheit in der Beschreibung des Systems toleriert wird. Die Komplexität des dadurch beschriebenen Modells verringert sich, wodurch es einfacher zu handhaben ist. Die Entwicklung von Fuzzy-Reglern birgt jedoch einige Schwierigkeiten. Es fehlt vor allem an formalen Methoden, die die Entwurfsphase des Reglers unterstützen. Diese findet zum Teil auf rein intuitiver Ebene statt und ist durch Versuch und Irrtum geprägt. Hier bietet sich der Einsatz von Neuronalen Netzen an. Sie können zur Optimierung der „vagen“ Teile von Fuzzy-Reglern herangezogen werden. Ein Modell eines solchen Neuronalen Fuzzy-Reglers wurde am Institut für Betriebssysteme und Rechnerverbund entwickelt.

Im Rahmen des Programmsystems NEFCON-I wurde dieser Neuronale Fuzzy-Regler implementiert. NEFCON-I bietet zusätzlich eine grafische Benutzeroberfläche, über die der Regler für beliebige Regelungsaufgaben trainiert werden kann. Es stehen verschiedene Grafikelemente zur Verfügung, um das Regelungsverhalten zu untersuchen. Als Beispiel für ein zu regelndes technisches System wurde die Simulation eines inversen Pendels implementiert.

Kapitel 2 befaßt sich zunächst mit der kognitiven Regelung. Hier werden die beiden Theorien kurz dargestellt, die in diesem Zusammenhang wichtig sind, nämlich die der Fuzzy-Mengen und Neuronale Netze. Die Kombination beider Modelle und der oben genannte Neuronale Fuzzy-Regler bilden das Thema von Kapitel 3.

Kapitel 4 versteht sich als Benutzerhandbuch für NEFCON-I. Hier wird beschrieben, welche Angaben für die Regelung eines technischen Systems nötig sind und wie sie mit Hilfe der Benutzeroberfläche erstellt werden. Die Implementierung von NEFCON-I wird in Kapitel 5 erläutert.

Auf die Bedienung und die Implementierung des Simulationsprogramms für das inverse Pendel wird in Kapitel 6 näher eingegangen.

Kapitel 2

Kognitive Regelung

In diesem Kapitel werden zwei verschiedene Ansätze zur kognitiven Regelung vorgestellt. Es handelt sich dabei zum einen um die Theorie der Fuzzy-Mengen und und zum anderen um Neuronale Netze.

Stellvertretend für andere regelungstechnische Aufgaben, werden hier Systeme betrachtet, die ihren Zustand durch *Ausgangs-* oder *Meßgrößen* beschreiben. Die Aufgabe der Regelung ist es, das System durch eine geeignete Wahl von *Stellwerten* in den gewünschten Zustand zu steuern.

2.1 Die Theorie der Fuzzy-Mengen

Eine Methode zur Modellierung komplexer Systeme bietet die Fuzzy-Mengen-Theorie. Bei diesem Ansatz wird ein gewisser Anteil an Ungenauigkeit in der Beschreibung des Systems toleriert und dadurch die Komplexität verringert. Angewandt werden kann diese Theorie in allen wissensbasierten Systemen, doch hat sie in den letzten Jahren vor allem im regelungstechnischen Bereich große Erfolge erzielt.

Im nächsten Unterabschnitt soll zunächst eine grundlegende Einführung in die Theorie der Fuzzy-Mengen gegeben werden. Der darauf folgende befaßt sich dann speziell mit der Fuzzy-Regelung.

2.1.1 Grundlagen

Was ist eine Fuzzy-Menge ?

Eine Möglichkeit, Eigenschaften mathematisch zu beschreiben, ist die Angabe der Menge von Elementen, auf die diese Eigenschaft zutrifft. Dies kann durch das Aufstellen von charakterisierenden Funktionen, auch Indikatorfunktionen genannt, geschehen. Jedem Element einer Grundmenge wird dabei entweder der Wert 1 oder 0 zugeordnet, abhängig davon, ob es zur entsprechenden Menge gehört oder nicht. Folgende Funktion repräsentiert beispielsweise die Eigenschaft *Postleitzahl aus Braunschweig* bezüglich der Natürlichen Zahlen:

$$f : I\!N \rightarrow \{0, 1\}$$
$$f(x) = \begin{cases} 1 & falls \quad 19050 \leq x/2 \leq 19065 \\ 0 & sonst \end{cases}$$

Zur Modellierung von linguistisch beschriebenen Eigenschaften, wie *groß*, *schnell* oder *alt*, ist diese Art der Darstellung jedoch weniger geeignet. Ihre Bedeutung ergibt sich aus dem Zusammenhang und kann von jedem Betrachter anders interpretiert werden. Es kann außerdem keine scharfe Grenze gezogen werden, ab der eine dieser Eigenschaften zutrifft oder nicht. Vielmehr ist dieser Übergang in der Vorstellung der meisten Menschen fließend. Um ihn zu modellieren, reichen die beiden Zugehörigkeitsgrade 0 und 1 nicht aus. Eine Möglichkeit zur Lösung dieses Problems ist die Verwendung von Zugehörigkeitsgraden aus dem kompakten Intervall [0,1].

Auf diese Weise kann eine verallgemeinerte charakterisierende Funktion

$$\mu : X \rightarrow [0, 1]$$

aufgestellt werden. Eine mögliche Funktion zur Beschreibung der Eigenschaft *alt* ist in Abb. 2.1 zu sehen. Jedem Element der Grundmenge X, in diesem Fall jedem Alter zwischen 0 und 100 Jahren, wird ein bestimmter Zugehörigkeitsgrad zugeordnet. Je näher sich dieser bei 1 befindet, desto sicherer ist man, daß die Eingenschaft *alt* auf das Element zutrifft.

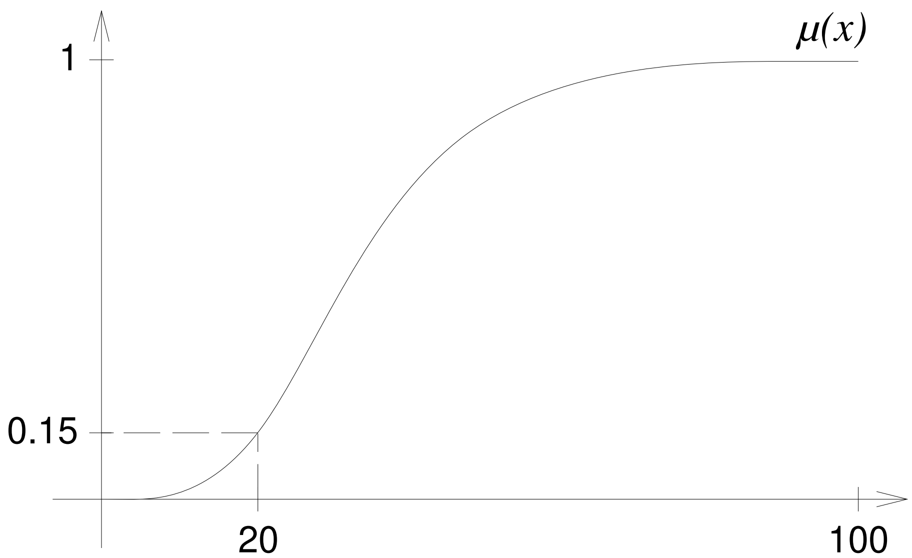


Abbildung 2.1: Verallgemeinerte charakterisierende Funktion

Verallgemeinerte charakterisierende Funktionen werden zur Beschreibung unscharfer Mengen, sogenannter *Fuzzy-Mengen*, verwendet. Mit ihrer Hilfe können vage Daten beschrieben werden, wobei aber zu beachten ist, daß die Zugehörigkeitsgrade nicht als Wahrscheinlichkeiten interpretierbar sind. Es ist jedoch möglich, den Zugehörigkeitsgrad einer Fuzzy-Menge als Akzeptanzgrad für den Wahrheitsgehalt einer Aussage zu betrachten. Im oben genannten Beispiel hätte die Aussage *20 ist alt* also den Akzeptanzgrad 0.15.

Operationen auf Fuzzy-Mengen

Die grundlegenden Verknüpfungen aus der Mengenlehre, nämlich Durchschnitt, Vereinigung und Komplement, sind auch auf Fuzzy-Mengen übertragbar. Durchschnitt und Vereinigung werden dabei auf die aus der mehrwertigen Logik bekannten Begriffe *t-Norm* und *t-Conorm* zurückgeführt. Das heißt, alle Funktionen, die die Axiome zur Definition einer t-Norm erfüllen, können als Schnittoperatoren verwendet werden. Dies gilt ebenso für die t-Conorm und den Vereinigungsoperator. Häufig werden folgende Funktionen als Verknüpfungsoperatoren verwendet:

$$\begin{array}{cccl} (\mu \cap \mu')(x) & \stackrel{\text{Df}}{=} & min(\mu(x), \mu'(x)) & \text{Durchschnitt} \\ (\mu \cup \mu')(x) & \stackrel{\text{Df}}{=} & max(\mu(x), \mu'(x)) & \text{Vereinigung} \\ \overline{\mu}(x) & \stackrel{\text{Df}}{=} & 1 - \mu(x) & \text{Komplement} \end{array}$$

Mit Hilfe dieser Verknüpfungen ist es nun möglich, verschiedene vage Aussagen miteinander zu kombinieren, so daß Ausdrücke der Form

(20 ist alt) und (15 ist jung)
(50 ist alt) oder (50 ist jung)

auswertbar sind. Voraussetzung dabei ist die Modellierung der linguistischen Werte *alt* und *jung* durch entsprechende Fuzzy-Mengen. Bei der Auswertung werden als Konjunktionen t-Normen und als Disjunktionen t-Conormen verwendet. Im folgenden werden diese Aufgaben von den Funktionen *min* und *max* übernommen. Für obige Ausdrücke ergibt sich damit folgende Auswertung:

$$\mathrm{min}(\ \mathrm{alt}(20)\ ,\ \mathrm{jung}(15)\ )$$
$$\mathrm{max}(\ \mathrm{alt}(50)\ ,\ \mathrm{jung}(50)\ )$$

Andere logische Konnektive, wie die Implikation oder die Biimplikation, können ebenfalls auf Fuzzy-Mengen übertragen werden. Dies gilt auch für Prädikate wie beispielsweise die Gleichheit. Durch die Erweiterung der Wahrheitswerte von der Menge $\{0,1\}$ auf das Intervall $[0,1]$ für das Prädikat $\in$ kann also die klassische zweiwertige Logik durch eine mehrwertige Logik ersetzt werden.

Auch Abbildungen der Form $\phi : X^n \rightarrow Y$ kann man auf Fuzzy-Mengen übertragen, wenn die Elemente der Grundmengen X bzw. Y durch entsprechende Fuzzy-Mengen modelliert sind. Diese *Extension* einer Abbildung ist definiert als:

$$\begin{aligned}
&\hat{\phi} : F(X)^n \rightarrow F(Y) \quad \text{mit} \\
&\hat{\phi}(\mu_1, \ldots, \mu_n)(y) \quad \stackrel{\text{Df}}{=} \quad sup\{min\{\mu_1(x_1), \ldots, \mu_n(x_n)\} \mid (x_1, \ldots, x_n) \in X^n \wedge \\
&\qquad\qquad\qquad\qquad y = \phi(x_1, \ldots, x_n)\}.
\end{aligned}$$

Dabei sei $sup\{\emptyset\} \stackrel{\text{Df}}{=} 0$ vereinbart. $F(X)$ bzw. $F(Y)$ bezeichnet die Menge aller Fuzzy-Mengen auf der Grundmenge X bzw. Y [Kruse et al., 1993].

Das Extensionsprinzip kann dazu dienen, ganze Bereiche der Mathematik zu „fuzzifizieren“. So ist es zum Beispiel möglich, arithmetische Operationen auf sogenannte *Fuzzy-Zahlen* zu übertragen. Die Subtraktion $\mu_1 \ominus \mu_2$ der Fuzzy-Zahlen μ_1 und μ_2 ist somit definiert als:

$$(\mu_1 \ominus \mu_2)(y) \quad = \quad sup\{min\{\mu_1(x_1), \mu_2(x_2)\} \mid x_1, x_2 \in I\!R \wedge x_1 - x_2 = y\}.$$

Für die Fuzzy-Zahlen 8 und 2 wird dies grafisch in Abb. 2.2 veranschaulicht.

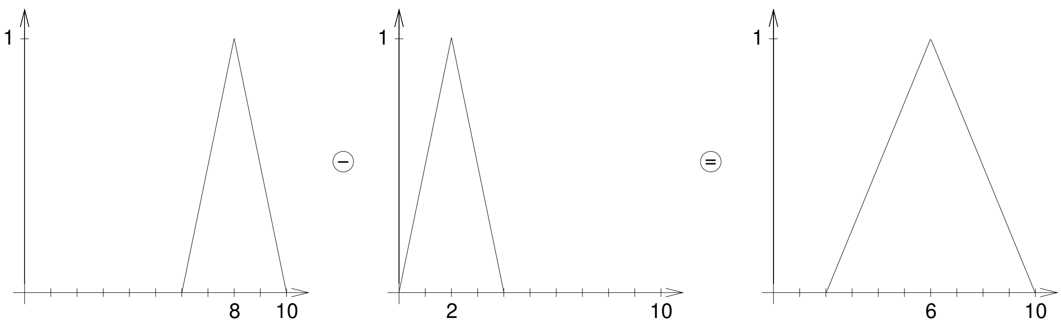


Abbildung 2.2: Subtraktion der Fuzzy-Zahlen 8 und 2

Approximatives Schließen

Es ist sinnvoll, die durch die Fuzzy-Mengen modellierte Vagheit und Unsicherheit auch in Expertensystemen zu integrieren. Dies geschieht durch die Verwendung einer Regelbasis aus unscharfen Regeln der Form:

R: **IF** ξ **IS** μ_x **THEN** η **IS** μ_y.

ξ und η sind dabei zwei sogenannte *linguistische Variablen*. Mit ihrer Hilfe werden bestimmte Objekttypen charakterisiert. μ_x und μ_y sind zwei Fuzzy-Mengen über den Wertebereichen, in denen die Objekttypen beschrieben werden. Im weiteren Verlauf dieser Arbeit wird unterschieden zwischen *Eingabevariablen*, in diesem Fall ξ, und *Ausgabevariablen*, in diesem Fall η.

Ein einfaches Beispiel bildet die Regel:

IF Alter der Wohnung = jung **THEN** Miete = hoch.

Das Alter der Wohnung und die Miete sind hier die linguistischen Ein- bzw. Ausgabevariablen, *jung* und *alt* die Namen zweier Fuzzy-Mengen. Als Wertebereich für das Alter könnte beispielsweise [0, 200] (Jahre) und für die Miete [100, 1000] (DM) gewählt werden.

Zur Darstellung vager Regeln können *Fuzzy-Relationen* verwendet werden, die man als mehrdimensionale Fuzzy-Mengen interpretieren kann. Die Regel aus dem obigen Beispiel ließe sich durch eine Funktion

$$\mu_R \,:\, [100, 1000] \,\times\, [0, 200] \,\rightarrow\, [0, 1]$$

ausdrücken.

Den Vorgang, aus einem System vager Regeln und konkret vorgegebenen Daten Schlußfolgerungen zu ziehen, nennt man *approximatives Schließen.* Der Inferenzmechanismus, der dazu benötigt wird, kann durch Verallgemeinerung des aus der zweiwertigen Logik bekannten *Modus Ponens* hergeleitet werden. Dazu wird nicht mehr die vollständige Übereinstimmung eines vagen Faktums mit der Prämisse einer Regel verlangt, sondern diese Übereinstimmung wird fuzzifiziert. Die Gültigkeit der Konklusion einer Regel hängt dann vom Grad dieser Übereinstimmung ab. Für nähere Informationen über den Inferenzmechanismus und seine Herleitung siehe [Kruse et al., 1993].

2.1.2 Fuzzy-Regler

Im Gegensatz zur klassischen Regelungstechnik, wo ein mathematisches Modell des zu regelnden Systems aufgestellt werden muß, wird beim Einsatz von Fuzzy-Reglern versucht, das Verhalten eines menschlichen Regelungsexperten zu simulieren. Man stellt dabei keine komplexen mathematischen Gleichungen auf, sondern formuliert das Wissen in Form von linguistischen Kontrollregeln der Form:

R: **IF** ξ_1 **IS** *ziemlich langsam* **AND** ξ_2 **IS** *sehr groß* **THEN** η **IS** *mittel stark*.

Diese Regel könnte zur Steuerung eines Systems herangezogen werden, das zwei Meßgrößen liefert und eine Stellgröße verarbeitet. Die Meßgrößen werden hier durch die beiden linguistischen Eingabevariablen ξ_1 und ξ_2, die Stellgröße durch die linguistische Ausgabevariable η dargestellt. In der Prämisse der Regel wird eine bestimmte Situation, in der sich das System befinden kann, unscharf beschrieben. Die Konklusion spezifiziert den dafür als geeignet angesehenen Stellwert ebenfalls unscharf. Für die linguistischen Terme *ziemlich langsam*, *sehr groß* und *mittel stark* müssen hier entsprechende Fuzzy-Mengen gefunden werden.

Auf der Grundlage einer gegebenen Menge von Kontrollregeln werden also aus den scharfen Meßwerten geeignete Stellwerte berechnet. Dazu wird eine Methode benötigt, die es erlaubt, die scharfen Meßwerte auf die Fuzzy-Mengen in der Prämisse der Regeln zu übertragen. Der Inferenzmechanismus liefert dann einen unscharfen Stellwert in Form einer Fuzzy-Menge. Um ihn für das zu regelnde System nutzbar zu machen, wird aus diesem ein scharfer Wert berechnet. Die Architektur eines allgemeinen Fuzzy-Reglers ist in Abb. 2.3 zu sehen.

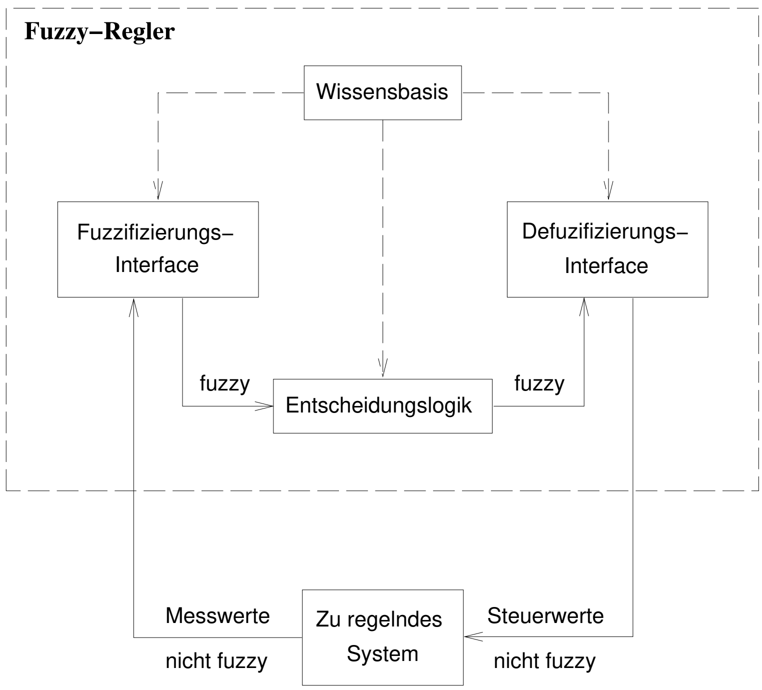


Abbildung 2.3: Architektur eines Fuzzy-Reglers

Ein Fuzzy-Regler besteht aus mehreren Komponenten, die folgende Aufgaben übernehmen:

- Das Fuzzifizierungs-Interface
  Diese Komponente übernimmt die Meßwerte des zu regelnden Systems und normiert sie gegebenenfalls. Außerdem werden die scharfen Meßwerte durch Fuzzy-Mengen modelliert. Im allgemeinen werden dazu sogenannte *Singletons* verwendet, das heißt, ein Meßwert x_0 wird durch die Indikatorfunktion der Menge $\{x_0\}$ dargestellt. Es ist aber auch möglich, andere Fuzzy-Mengen zu benutzen. So können beispielsweise eventuelle Ungenauigkeiten der Messung, sofern deren Größenordnungen bekannt sind, bei der Modellierung der Fuzzy-Mengen berücksichtigt werden.

- Die Entscheidungslogik
  Hier werden aus den Meßdaten, die vom Fuzzifizierungs-Interface geliefert werden, mit Hilfe der Wissensbasis Stellgrößen, meist in Form von Fuzzy-Mengen,

berechnet. Die Entscheidungslogik beinhaltet also im wesentlichen den Inferenzmechanismus.

- Die Wissensbasis
  Sie besteht aus der *Regelbasis* und der *Datenbasis*. In der Regelbasis sind alle linguistischen Kontrollregeln enthalten. Die Datenbasis enthält dagegen Informationen über die Wertebereiche der Meß- und Stellgrößen sowie eventuelle Normierungen. Außerdem enthält sie die Fuzzy-Mengen, mit denen die linguistischen Terme der Kontrollregeln modelliert werden.

- Das Defuzzifizierungs-Interface
  Die unscharfen Stellwerte der Entscheidungslogik werden hier in scharfe Werte umgewandelt. Falls diese normiert vorliegen, werden sie entsprechend skaliert.

Es existieren verschiedene Möglichkeiten, Fuzzy-Regler zu konstruieren. Zwei häufig benutzte werden im folgenden kurz vorgestellt. Anschließend wird näher auf die Optimierung von Fuzzy-Reglern eingegangen.

Der Einfachheit halber werden in diesem Kapitel nur noch Fälle mit nur einer linguistischen Ausgabevariable betrachtet. Das heißt, daß die zu regelnden Systeme nur eine Stellgröße verarbeiten.

Der Ansatz von Mamdani

Um linguistische Kontrollregeln für die Wissensbasis aufstellen zu können, müssen die Grundmengen der linguistischen Eingabevariablen $\xi_1, \ldots, \xi_n$ und der Ausgabevariablen η durch Fuzzy-Mengen *partitioniert* werden. Jede dieser Fuzzy-Mengen wird dann mit genau einem linguistischen Term assoziiert, der den Zustand genau einer linguistischen Variablen beschreibt. Jeder Variablen wird so eine Anzahl von Fuzzy-Mengen zugeordnet.

Häufig wählt man als Fuzzy-Mengen stückweise lineare Funktionen, weil sich dadurch die Berechnungen innerhalb des Reglers vereinfachen. Oft werden beispielsweise Dreiecks- oder Trapezfunktionen benutzt. In Abb. 2.4 ist die mögliche Partitionierung einer Grundmenge durch 7 Fuzzy-Mengen dargestellt. Man sollte zum einen darauf achten, daß die Partitionierung nicht zu grob wird. Es sollte also eine genügend große Anzahl von Fuzzy-Mengen definiert werden. Zum anderen sollte die Grundmenge vollständig durch Fuzzy-Mengen abgedeckt sein, damit möglichst alle Elemente der Grundmenge durch sie beschreibbar sind.

Nach Festlegung der Partitionierung kann die Regelbasis definiert werden. In Abb. 2.5 ist als Beispiel eine Regelbasis in Form einer Tabelle dargestellt. Die Spalten und Zeilen der Tabelle repräsentieren jeweils eine Fuzzy-Menge der Eingabevariablen ξ_1 bzw. ξ_2. Die Tabelle selbst enthält die Namen von den Fuzzy-Mengen der Ausgabevariablen η. Der durch einen Pfeil markierte Tabelleneintrag entspricht also der Regel:

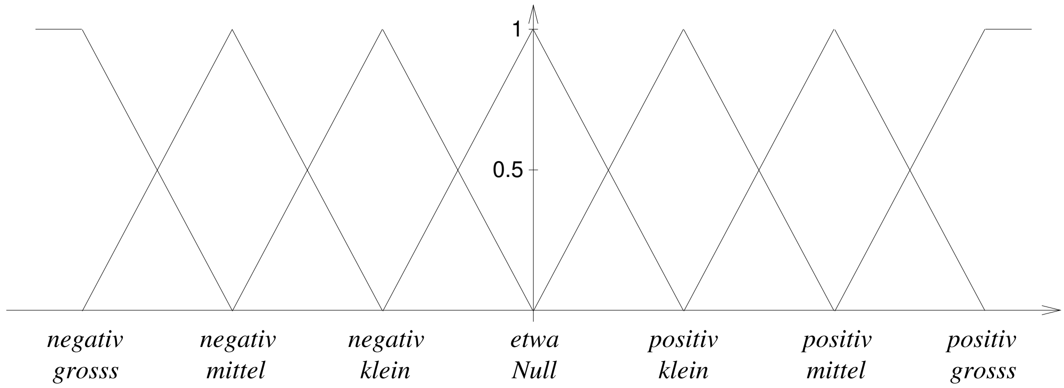


Abbildung 2.4: Partitionierung einer Grundmenge

$$\text{R: } \mathbf{IF}\ \xi_1\ \mathbf{IS}\ \mathit{NM}\ \mathbf{AND}\ \xi_2\ \mathbf{IS}\ \mathit{EN}\ \mathbf{THEN}\ \eta\ \mathbf{IS}\ \mathit{NM}.$$

Die Namen der in dieser Tabelle verwendeten Fuzzy-Mengen sind abkürzende Schreibweisen für die Namen aus Abb. 2.4.

ξ_1

		NG	NM	NK	EN	PK	PM	PG
	NG			PK	PG			
	NM				PM			
	NK	NM		NK	PK			
ξ_2	EN	NG	NM	NK	EN	PK	PM	PG
	PK				NK	PK		PM
	PM				NM			
	PG				NG	NK		

Abbildung 2.5: Beispiel einer Regelbasis

Bei einem System mit mehreren Ausgabevariablen sind auch Regeln denkbar, die in der Konklusion mehrere Fuzzy-Mengen enthalten. Solche Regeln können aber durch mehrere „einfache“ Regeln, also Regeln mit nur einer Fuzzy-Menge in der Konklusion, ersetzt werden.

Aufbauend auf die so definierte Wissensbasis, werden die Stellwerte in folgenden Schritten berechnet:

1. Das Fuzzifizierungs-Interface übergibt die Meßwerte nach einer eventuellen Normierung direkt an die Entscheidungslogik. Das heißt, sie werden nicht durch Fuzzy-Mengen modelliert, eine Fuzzifizierung entfällt damit.

2. Für jede einzelne Regel wird der Erfülltheits- oder Akzeptanzgrad bestimmt, der angibt, zu welchem Grad die Meßwerte $x_1, \ldots, x_n$ die Prämisse der Regel erfüllen. Dazu wird für jede Fuzzy-Menge $\mu_{v,i}$ aus der Prämisse von R_i der Wert $\mu_{v,i}(x_v)$ berechnet. Durch konjunktive Verknüpfung dieser Werte erhält man dann den Akzeptanzgrad α_i der Regel. Die Ausgabe der Regel ist die Fuzzy-Menge, die entsteht, wenn man die Fuzzy-Menge aus der Konklusion der Regel an der Stelle α_i „abschneidet". Sie berechnet sich wie folgt:

$$\begin{aligned} \mu_{x_1,\ldots,x_n}^{\text{ausgabe}(R_i)} &: Y \to [0,1] \\ y &\mapsto \min\{\mu_{1,i}(x_1), \ldots, \mu_{n,i}(x_n), \mu_i(y)\}, \end{aligned}$$

wobei μ_i die Fuzzy-Menge aus der Konklusion der Regel R_i repräsentiert.

In Abb. 2.6 wird der gesamte Vorgang grafisch veranschaulicht. Der Akzeptanzgrad wird hier durch eine gestrichelte Linie dargestellt, und die beiden Meßwerte werden durch Pfeile markiert.

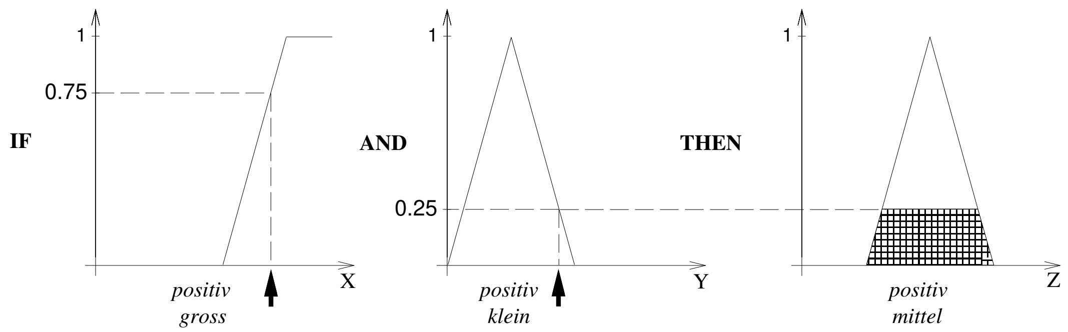


Abbildung 2.6: Auswertung einer linguistischen Regel

Die Fuzzy-Mengen, die man als Ausgabe für die einzelnen Regeln erhält, werden zu einem Gesamtergebnis vereinigt. Dieses ergibt sich also zu:

$$\begin{aligned} \mu_{x_1,\ldots,x_n}^{\text{ausgabe}} &: Y \to [0,1] \\ y &\mapsto \max_{i \in \{1,\ldots,k\}} \{\min\{\mu_{1,i}(x_1), \ldots, \mu_{n,i}(x_n), \mu_i(y)\}, \end{aligned}$$

wobei k die Anzahl der Regeln angibt.

3. Nachdem die Entscheidungslogik für jede Ausgabevariable einen Stellwert in Form einer Fuzzy-Menge berechnet hat, muß nun im Defuzzifizierungs-Interface daraus ein scharfer Wert berechnet werden (siehe Abb. 2.7). Dieses Problem wird häufig durch eine der drei folgenden Methoden gelöst:

   - Die Max-Kriterium-Methode
     Hier wird ein beliebiger Wert $y \in Y$ ausgewählt, dessen Zugehörigkeitsgrad zur Fuzzy-Menge $\mu_{x_1,\ldots,x_n}^{\text{ausgabe}}$ maximal ist. Für die in Abb. 2.7 dargestellte Fuzzy-Menge müßte der Wert dann zwischen 3 und 5 liegen.

- Die Mean-of-Maximum-Methode (MOM)
  Bei dieser Methode wird als scharfer Stellwert der Mittelwert der Werte berechnet, deren Zugehörigkeitsgrade zur Fuzzy-Menge $\mu_{x_1,\ldots,x_n}^{\text{ausgabe}}$ maximal sind.
- Die Center-of-Area-Methode (COA)
  Diese Methode berechnet den Schwerpunkt der Fläche, die durch die Funktion $\mu_{x_1,\ldots,x_n}^{\text{ausgabe}}$ begrenzt wird. Als scharfer Stellwert wird dann der Wert $y \in Y$ verwendet, der „unter“ diesem Schwerpunkt liegt.

Von diesen drei Methoden liefert die Center-of-Area-Methode die besten Ergebnisse. Sie zeigt ein relativ glattes Regelverhalten, benötigt aber aufwendige Berechnungen.

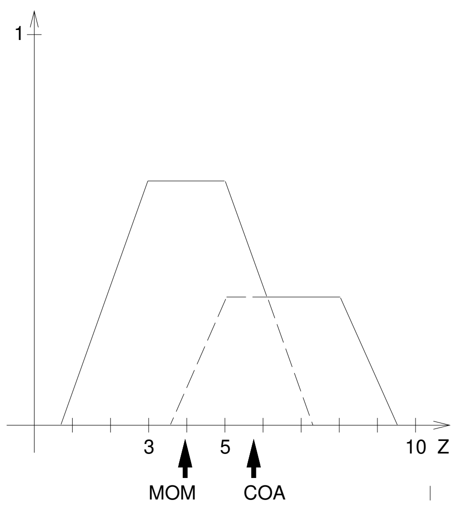


Abbildung 2.7: Defuzzifizieren einer Fuzzy-Menge

Der Ansatz von Sugeno

Bei diesem Ansatz werden nur die Grundmengen der linguistischen Eingabevariablen durch Fuzzy-Mengen partitioniert. Die k Kontrollregeln haben hier die Form:

$$R_i\text{: } \mathbf{IF}\ \xi_1\ \mathbf{IS}\ \mu_1\ \mathbf{AND}\ \xi_2\ \mathbf{IS}\ \mu_2\ \mathbf{THEN}\ \eta\ =\ f_i(\xi_1, \ldots, \xi_n),$$

wobei f_i eine Abbildung von $X_1 \times \cdots \times X_n$ nach Y ist. Jede Regel liefert als Ausgabe also einen scharfen Wert. Die Entscheidungslogik berechnet für jede einzelne Regel R_i den Akzeptanzgrad α_i nach derselben Methode wie beim Mamdani-Fuzzy-Regler. Der Stellwert y berechnet sich dann als der Mittelwert der mit den Akzeptanzgraden gewichteten Summe der einzelnen Regelausgaben:

$$y = \frac{\sum_{i=1}^{k} \alpha_i \, f_r(x_1, \ldots, x_n)}{\sum_{i=1}^{k} \alpha_i}$$

Die Optimierung von Fuzzy-Reglern

Der Entwurf eines Fuzzy-Reglers besteht im wesentlichen aus der Partitionierung der Wertebereiche der Meß- und Stellgrößen durch geeignete Fuzzy-Mengen und der Definition der Kontrollregeln. Es fehlt jedoch an verbindlichen Richtlinien, die festlegen, wie grob oder fein eine Partitionierung zu sein hat oder welche Form für die Fuzzy-Mengen am günstigsten ist. Für die Festlegung der Kontrollregeln wird im allgemeinen ein Experte herangezogen, der aufgrund seiner Erfahrungen die Regeln definiert.

Die Definition der gesamten Wissensbasis erfolgt also zum großen Teil auf rein intuitiver Basis. Dies ist einer der Vorteile von Fuzzy-Reglern, dies bedeutet aber auch, daß eine so definierte Wissensbasis Ungenauigkeiten und Fehler enthält. Durch kleine Änderungen in der Wissensbasis ist es fast immer möglich, das Regelverhalten zu verbessern. Diese Änderungen betreffen vor allem Form und Lage der Fuzzy-Mengen.

Diese Optimierungen manuell durchzuführen, ist sehr mühselig, daher liegt es nahe, Fuzzy-Regler so zu modifizieren, daß sie selbst dazu in der Lage sind. Hauptvoraussetzung dafür ist, Fuzzy-Regler um eine gewisse Lernfähigkeit zu erweitern.

Die Eigenschaft der Lernfähigkeit ist vor allem von Neuronalen Netzen bekannt, die das Thema des nächsten Abschnitts bilden. Auf die Optimierung von Fuzzy-Reglern mit Hilfe von Neuronalen Netzen wird in Kapitel 3 näher eingegangen.

2.2 Neuronale Netze

Durch den Einsatz immer leistungsfähigerer Computer ist es heute möglich, Aufgaben zu lösen, die dem Menschen schwer fallen. Dennoch versagen sie teilweise bei Alltagsaufgaben, die der Mensch fast mühelos beherrscht. Der Grund liegt in der unterschiedlichen Konstruktionsweise von Gehirn und Rechner.

Heutige Rechner basieren auf der sogenannten *von-Neumann-Architektur*. Sie enthalten einen oder wenige zentrale Mikroprozessoren, die mit extrem hoher Geschwindigkeit lange Programme sequentiell abarbeiten. Im Gegensatz dazu basiert die Informationsverarbeitung im Gehirn auf einer sehr großen Anzahl von Neuronen. Diese sind für sich genommen erheblich langsamer als elektrische Schaltelemente, sie gleichen diesen Nachteil allerdings durch massive Vernetzung und damit verbundene Parallelität wieder aus [Ritter, 1992, Rojas, 1993].

Biologische Neuronen-Netze, oder allgemein *Neuronale Netze*, zeichnen sich vor allem durch folgende Eigenschaften aus:

Lernfähigkeit: Die Verbindung der einzelnen Neuronen innerhalb des Netzes ist nicht statisch sondern dynamisch. Durch Änderungen in dieser Verbindungsstruktur ist das Netz in der Lage, sich bestimmten Aufgaben besser anzupassen.

Parallelität: Jedes Neuron kann als einfaches Verarbeitungselement betrachtet werden. Durch die Vernetzung ist es möglich, daß eine große Anzahl von Neuronen gleichzeitig aktiv ist.

Fehlertolleranz: Ein einzelnes Neuron spielt im Gesamtgeschehen eine relativ unbedeutende Rolle. Der Ausfall einzelner Neuronen bewirkt in der Regel lediglich eine Güteverschlechterung der gespeicherten Informationen, gefährdet aber nicht das Funktionieren des Gesamtsystems. Der Grund liegt darin, daß das Wissen des Systems in vielen Speicherelementen verteilt abgelegt wird.

Um diese Vorteile auch für Rechner nutzbar zu machen, werden *künstliche Neuronale Netze* definiert, sie werden im folgenden vereinfachend als Neuronale Netze bezeichnet. Sie können beispielsweise in der Mustererkennung dazu eingesetzt werden, Muster zu klassifizieren oder zu vervollständigen. Eine andere Möglichkeit ist die Verwendung von Neuronalen Netzen als assoziative Speicher, oder der Einsatz in der Robotik zur Steuerung mehrgelenkiger Roboterarme. Allgemein kann man sagen, daß ein Neuronales Netz als eine Art „Abbildungsmaschine“ arbeitet, die eine bestimmte Funktion modelliert [Rojas, 1993].

Der nächste Abschnitt beschäftigt sich zunächst mit den Grundlagen und der Definition Neuronaler Netze. In den folgenden Abschnitten wird dann näher auf verschiedene Modelle und Lernverfahren eingegangen. Abschließend wird der Einsatz Neuronaler Netze als Regler näher beschrieben.

2.2.1 Grundlagen

Grundlegend für die Konstruktion eines Neuronalen Netzes, ist die formale Modellierung der einzelnen Neuronen. Da die biologischen Neuronen dazu als Vorbild dienen, soll zunächst deren Aufbau beschrieben werden. Bei dem in Abb. 2.8 gezeigten vereinfachten Neuron erkennt man im wesentlichen drei Komponenten:

Der Dendritenbaum: Er dient dem Neuron als Eingabe. Durch eine verästelte Struktur von Zellvortsätzen (Dendriten) summiert er die elektrischen Impulse, die er von den mit ihm vernetzten Neuronen empfängt.

Der Zellkörper: Hier addiert sich das elektrische Potential. Überschreitet es einen bestimmten Schwellwert, erzeugt der Zellkörper einen elektrischen Impuls.

Das Axon: Über die Verzweigungen des Axons kann der Impuls zu anderen Neuronen übertragen werden. Die Kontaktstellen, durch die das Axon mit anderen Neuronen verbunden ist, nennt man *Synapsen.* Man unterscheidet zwei Arten, nämlich erregende (exzitatorische) und hemmende (inhibitorische) Synapsen. Erregende Synapsen bewirken, daß eintreffende Impulse das elektrische Potential erhöhen. Hemmende Synapsen wirken dagegen genau umgekehrt. Der Grad, in dem eine Synapse erregend oder hemmend wirkt, ist nicht statisch sondern dynamisch.

Die Synapsen werden hierbei als Ort der Informationsspeicherung angesehen. Die Stärke der Synapsen wird durch die Aktivität der jeweiligen Neuronen beeinflußt. Diese Veränderbarkeit der Synapsenstärke bildet die Grundlage für die Lernfähigkeit des Gehirns [HEBB, 1949].

Formal wird ein solches Neuron als Addierer mit mehreren Eingängen x_i betrachtet, die mit veränderlichen Gewichten ω_i versehen sind [WALOSZEK, 1988]. Der Zustand des Addierers z, auch Aktivität genannt, wird durch die sogenannte *Aktivierungsfunktion* festgelegt. Er wird oft als Summe der Produkte von Eingangssignalen und Gewichten berechnet. Zusätzlich kann ein Schwellwert T vorhanden sein, der die Aktivität herabsetzt. Bei i Eingängen ergibt sich dann folgende Berechnungsvorschrift:

$$z \quad = \quad -T + \sum_i \omega_i \, x_i$$

Das abstrakte Neuron hat genau einen Ausgang, dessen Wert y sich mit Hilfe der *Ausgabefunktion* S aus der Aktivität berechnet. Folgende Ausgabefunktion liefert beispielsweise nur binäre Ausgabewerte:

$$y \quad = S(z) = \begin{cases} 1 & falls \quad z > 0 \\ 0 & sonst \end{cases}$$

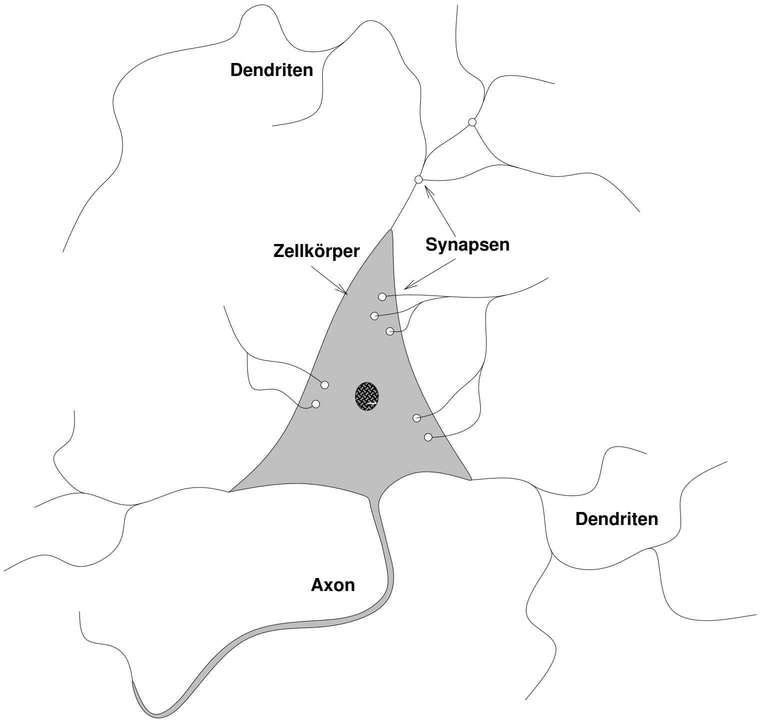


Abbildung 2.8: Schematischer Aufbau eines Neurons

Wesentlichen Einfluß auf das Verhalten eines Neuronalen Netzes hat die Netzwerkstruktur, über die die einzelnen Elemente verbunden sind. Meistens werden hierachische Strukturen verwendet, in denen die Elemente, also die Neuronen, mehrere Schichten bilden. Man orientiert sich dabei am biologischen Vorbild, denn auch im Kortex, der Hirnrinde, sind verschiedene Neuronenschichten erkennbar. Bei Netzen mit mehr als einer Schicht nennt man die äußeren beiden Schichten, die die Schnittstellen zur Umwelt bilden, Eingabe- und Ausgabeschicht (siehe Abb. 2.9).

Hierachische Netzwerkstrukturen teilen sich in drei Kategorien [KEMKE, 1988]:

Bottom-up-Systeme: Jedes Element kann nur von Elementen einer niedrigeren Schicht beeinflußt werden. Aktivierungen werden also nur von der Eingabeseite in Richtung der Ausgabeseite propagiert.

Top-down-Systeme: Hier können Elemente nur von Elementen aus höheren Ebenen beeinflußt werden. So ist es möglich, niedrigere Verarbeitungsprozesse von „oben“ zu steuern.

Interaktiv arbeitende Systeme: In diesen Systemen sind beide Richtungen möglich. Meist findet die Kommunkation nur zwischen zwei benachbarten Ebenen statt.

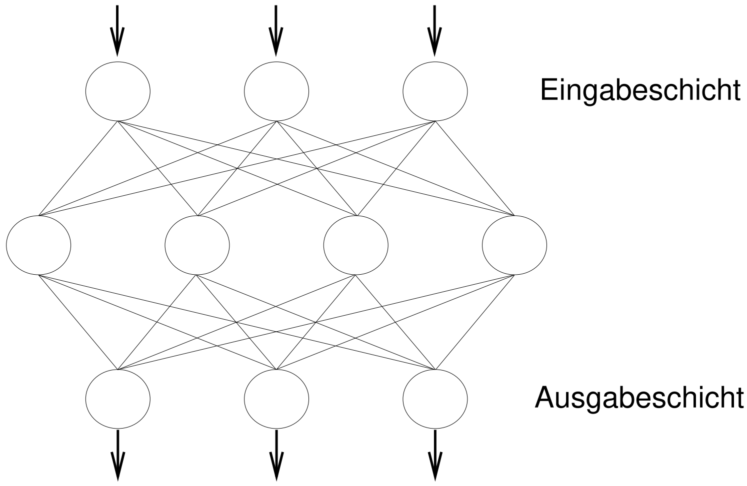


Abbildung 2.9: Dreischichtiges Netzwerk

Zur Klassifikation von Neuronalen Netzen können mehrere Kriterien herangezogen werden. Neben der Einteilung anhand der Netzwerkarchitektur, die oben vorgestellt wurde, kann beispielsweise auch nach Art der Aktivierungs- oder der Ausgabefunktion klassifiziert werden. Da die gesamte Forschungsrichtung noch stark in Bewegung ist, hat sich noch keine einheitliche Methode herausgebildet.

2.2.2 Lernen in Neuronalen Netzen

Neuronale Netze sind in der Lage, sich selbst zu organisieren. Das heißt, sie können sich durch Modifizieren ihrer Gewichte bzw. ihrer Verbindungsstruktur einer bestimmten Aufgabe anpassen. Die wichtigsten Aufgaben, für die Neuronale Netze eingesetzt werden sind [Waloszek, 1988]:

- Als *assoziative Speicher* assoziieren Netze Musterpaare, daß heißt, sie speicheren eines der beiden Muster und verwenden das andere als Abrufschlüssel.
- Beim Einsatz als *Autoassoziatoren* werden unvollständige Muster von Neuronalen Netzen vervollständigt.
- Um Muster in bestimmte Klassen einzuteilen, werden Netze als *Mustererkenner* benutzt.
- Als *Ähnlichkeitsdetektoren* erkennen sie Ähnlichketen in den dargebotenen Mustern.

Das Ändern der Verbindungstrukturen im Rahmen des Lernens beinhaltet das Löschen alter und das Einrichten neuer Verbindungen. Wenn man von einer vollständigen Verbindungsstruktur ausgeht, kann dies durch entsprechende Änderungen der Gewichte simuliert werden, die den Verbindungen zugeordnet sind.

Das Lernen findet in einer sogenannten *Lernphase* statt, die vor der eigentlichen *Arbeitsphase* des Netzes liegt. Während der Lernphase wird versucht, die Gewichte des Netzes so zu verändern, daß es auf alle Eingabemuster mit den gewünschten Ausgabemustern reagiert. Nach Abschluß der Lernphase soll das Netz aber auch in der Lage sein, auf unbekannte Eingabemuster korrekte Ausgaben zu liefern. Diese Fähigkeit nennt man *Generalisierung*.

Es existieren verschiedene Gruppen von Lernverfahren:

- Unüberwachtes Lernen
  Bei diesen Lernverfahren erhalten die Netze keinerlei Informationen über das gewünschte Ausgabemuster. Ein Beispiel für unüberwachtes Lernen bildet die Hebb'sche Lernregel, die auf der Hypothese beruht, daß die Verbindung zwischen zwei Elementen u_i und u_j verstärkt wird, wenn beide Elemente gleichzeitig aktiv sind. Die Änderung des Gewichtes ergibt sich damit zu:

  $$\triangle\omega_{ij} = \eta \; a_i(t) \; a_j(t),$$

  mit dem Lernfaktor η. $a_i(t)$ und $a_j(t)$ beschreiben die Aktivitäten der Elemente u_i und u_j zum Zeitpunkt t.

- Überwachtes Lernen
  Wenn Informationen über das zu lösende Problem bekannt sind, können diese in das Lernverfahren einbezogen werden. Die Informationen können beispielsweise aus Paaren von Eingabemustern mit den dazugehörigen korrekten Ausgabemustern bestehen. Mit ihrer Hilfe kann ein Fehler berechnet werden, der dann zum Justieren der Gewichte benutzt wird. Ein Beispiel für ein solches Verfahren bildet die *Widrow-Hoff-* oder *Delta-Regel*:

  $$\triangle\omega_{ij} = \eta \; (l_j - a_j(t)) \; o_i(t).$$

  Als Fehler wird hier die Differenz zwischen dem gewünschten Aktivierungsgrad l_j und dem aktuellen Aktivierungsgrad $a_j(t)$ des Elements u_j verwendet. $o_i(t)$ bezeichnet die Ausgabe des vorgeschalteten Elements u_i zum Zeitpunkt t.

  Bei mehrschichtigen Netzen ergibt sich jedoch das Problem, wie die Gewichte der inneren Elemente justiert werden sollen, da sich das gewünschte Ausgabemuster meist auf die Ausgabeschicht bezieht. Dieses Problem wird durch den Backpropagation-Algorithmus gelöst. Er stellt eine Erweiterung der Delta-Regel dar und ist damit auch für mehrschichtige Netze geeignet [Kemke, 1988, Rumelhart et al., 1986a, Rumelhart et al., 1986b].

  Der Backpropagation-Algorithmus ist die zur Zeit am häufigsten gebrauchte Lernfunktion. Ein anderes Beispiel für überwachtes Lernen bildet das Perzeptron-Modell.

- Verstärkendes Lernen
  Diese Methode ist ein Sonderfall des überwachten Lernens. Sie wird hier jedoch ausführlich behandelt, da später noch auf sie zurückgegriffen wird (siehe Abschnitt 3.2).

  Es gibt Fälle, in denen die Größe eines Fehler nicht meßbar ist, die aber dennoch eine qualitative Beurteilung einer Netzwerkausgabe erlauben. In diesen Fällen kann die Ausgabe eines Netzes durch einen externen Beobachter bewertet werden. Das Netz erhält dann ein Verstärkungssignal, das angibt, ob eine Ausgabe als „gut“ oder „schlecht“ beurteilt wird. Bei einer positiven Bewertung werden die Gewichte, die zu dieser Ausgabe beigetragen haben, verstärkt. Bei einer negativen Bewertung werden sie entsprechend abgeschwächt.

Ob ein Lernvorgang erfolgreich ist oder nicht, ist von den Parametern des Neuronalen Netzes und dem verwendeten Lernverfahren abhängig. So kann beispielsweise bei mehrschichtigen Netzen eine zu geringe Anzahl innerer Neuronen den Lernerfolg unmöglich machen.

Bei Lernverfahren, die ein Fehlermaß verwenden, wird oft versucht, den Fehler über ein Gradientenverfahren zu minimieren. Wegen möglicher lokaler Minima kann es bei diesen Verfahren bezüglich der Konvergenz zu Problemen kommen.

2.2.3 Neuronale Regler

Für den Einsatz Neuronaler Netze als Regler wird kein mathematisches Modell des zu regelnden Systems benötigt. Es ist ausreichend, wenn das gewünschte Verhalten des Reglers durch eine Menge von Beispieldaten beschrieben werden kann.

Im einfachsten Fall bestehen die Beispieldaten aus einer Menge von Eingabemustern mit den dazugehörigen korrekten Ausgabemustern. Mit Hilfe der Delta-Regel oder dem Backpropagation-Algorithmus kann dann ein Neuronales Netz trainiert werden. Dabei sind mehrere Durchläufe durch den vorgegebenen Satz an Beispieldaten nötig. Diese Art des Trainings wird als *Offline-Lernen* bezeichnet. So trainierte Netze bilden das Verhalten eines Bedieners nach und werden daher als *überwachte Regler* bezeichnet. Sie sind zur Regelung gedächtnisloser Systeme geeignet, die eine Erfassung der Meß- und Stellgrößen erlauben.

Ein anderes Modell wird durch Neuronale Regler dargestellt, die durch verstärkendes Lernen trainiert werden. Da sie während des Einsatzes in Realzeit lernen, spricht man hier von *Online-Lernen*. Regler dieser Art bestehen aus einem adaptiven Kritiknetz und einem Handlungsnetz, sie werden daher auch als *adaptive Kritiker* bezeichnet [Werbos, 1992].

In Abb. 2.10 ist der schematische Aufbau eines adaptiven Kritikers zu sehen. Von einem externen Bewerter wird ein Verstärkungssignal (external reinforcement) bereitgestellt, das gewöhnlich nur angibt, ob die Regelung fehlschlug oder nicht. Anhand dieses Signals und der Meßwerte des Systems lernt das Kritiknetz, die Bewertung des nächsten Systemzustands vorherzusagen. Das Handlungsnetz dagegen berechnet aus den eingehenden Meßwerten die Stellgröße für das zu regelnde System. Die Gewichte beider Netze ändern sich in Abhängigkeit der Vorhersage des Kritiknetzes (internal reinforcement) und der eingehenden Meßwerte.

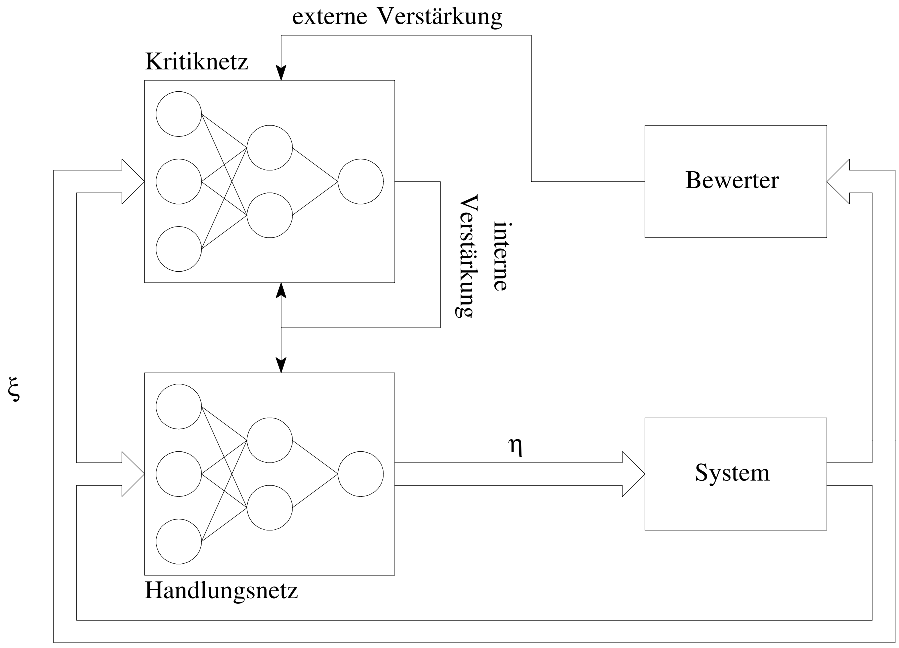


Abbildung 2.10: Adaptiver Kritiker

Ein adaptiver Kritiker lernt während des Einsatzes durch „Ausprobieren" verschiedener Stellwerte. Es ist also nicht nötig, das technische System bereits vorher zu beherrschen, sondern es genügt, wenn dem Regler die aktuellen Meßwerte zur Verfügung stehen. Dabei muß in Kauf genommen werden, daß die Regelung in der Lernphase auch fehlschlagen kann. Falls dies nicht akzeptabel ist, muß das technische System durch ein geeignetes Modell simuliert werden.

Die beiden angesprochenen Modelle Neuronaler Regler sind zur Regelung vieler Systeme ausreichend. Für komplexere Systeme sind aufwendigere Ansätze erforderlich, wie sie in [White und Sofge, 1992] beschrieben werden. Näheres zum Thema *Neuronale Regler* kann [Nauck, 1993] entnommen werden.

Kapitel 3

Die Kombination von Fuzzy-Reglern mit Neuronalen Netzen

Die beiden Modelle zur Regelung von Prozessen, die im vorhergehenden Kapitel beschrieben wurden, haben Vor- und Nachteile. In diesem Kapitel werden Verfahren vorgestellt, die versuchen, beide Modelle so miteinander zu kombinieren, daß möglichst alle Vorteile erhalten und die Nachteile vermieden werden.

Ein Vorteil von Fuzzy-Reglern liegt in der leichten Verständlichkeit ihrer Wissensbasis. Ihr Regelverhalten ist einfach zu interpretieren, da es auf linguistischen Kontrollregeln basiert. Die Information, die in der Wissensbasis steckt, muß ihnen allerdings von außen vorgegeben werden. Der Nachteil dabei ist, daß es kein festgeschriebenes Verfahren gibt, das eine Umsetzung von Expertenwissen in für einen Fuzzy-Regler verständliches Wissen vornimmt.

Vor allem die Wahl der einzelnen Zugehörigkeitsfunktionen geschieht auf rein intuitiver Basis, obwohl gerade sie für die Güte des Regelverhaltens sehr entscheidend sind. Zwar hat ein befragter Experte meist eine ungefähre Vorstellung davon, in welchem Bereich eine Zugehörigkeitsfunktion anzusiedeln ist, dennoch hat er die Wahl, die Form beispielsweise als Gaußkurve oder Dreiecksfunktion zu modellieren.

Neuronale Netze dagegen justieren ihre Gewichte während des Lernvorgangs selbst. Der Nachteil ist jedoch, daß das Wissen oder die Information, die in der Verbindungsstruktur eines solchen Netzes gespeichert ist, von außen nicht interpretierbar ist. Ein Neuronales Netz erscheint daher als „black box". Aus diesem Grund ist es umgekehrt auch nicht möglich, eventuell schon vorhandene Informationen in den Lernvorgang zu integrieren und ihn damit zu beschleunigen.

Man erkennt, daß in den Bereichen, in denen das eine System Schwächen aufweist, das andere seine Stärken hat. Es bietet sich also an, Fuzzy-Regler und Neuronale

Netze miteinander zu kombinieren. Dazu existieren verschiedene Ansätze, die im folgenden Unterabschnitt kurz skizziert werden (siehe dazu auch [Nauck et al., 1993, Nauck, 1993, Eklund et al., 1992]). Der im Rahmen dieser Arbeit implementierte Ansatz wird danach detailiert beschrieben.

3.1 Verschiedene Modelle

Die verschiedenen Ansätze kann man grob in zwei Klassen unterteilen:

- Kooperative Systeme
  Hier arbeiten gewöhnliche Fuzzy-Regler und Neuronale Netze zusammen, wobei ihre Aufgaben voneinander getrennt sind.
- Hybride Systeme
  Diese Ansätze verbinden die Architekturen von Fuzzy-Reglern und Neuronalen Netzen miteinander.

In kooperativen Systemen übernehmen Neuronale Netze die Aufgabe, bestimmte Teile eines Fuzzy-Reglers zu verändern. Das Problem, die richtige Form der Zugehörigkeitsfunktionen zu finden, kann als Optimierungsaufgabe angesehen werden. Es bietet sich also an, Neuronale Netze gerade für diese Aufgabe einzusetzen. Entsprechende Ansätze für Dreicksfunktionen wurden in [Nomura et al., 1992] und für Gaußfunktionen in [Ichihashi, 1991] gemacht. In beiden Fällen werden korrekte Paare von Ein- und Ausgabemustern, Kontrollregeln und vorläufige Definitionen der Zugehörigkeitsfunktionen als Vorgabe benötigt. In [Hayashi et al., 1992, Takagi und Hayashi, 1991, Takagi et al., 1992] sind Methoden beschrieben, die in den Eingabevariablen mit beliebigen Zugehörigkeitsfunktionen arbeiten.

Eine andere Einsatzmöglichkeit für Neuronale Netze bietet sich in diesem Zusammenhang beim Erlernen von Kontrollregeln an. In [Pedrycz und Card, 1992] ist ein Ansatz beschrieben, bei dem aus einer Menge korrekter Ein- und Ausgabemuster entsprechende Kontrollregeln erlernt werden.

In hybriden Systemen werden die Architekturen beider Ansätze miteinander verschmolzen. Abhängig davon, auf welchen der Ansätze man den Schwerpunkt legt, unterscheidet man zwei Fälle:

- Neuronale Netze als Fuzzy-Regler
- Neuronal orientierte Fuzzy-Regler

Ein Beispiel für den ersten Fall, also ein Neuronales Netz, das als Fuzzy-Regler interpretiert werden kann, bildet das in [Berenji, 1992] beschriebene System, das hier kurz skiziert wird. Diese ARIC-Architektur *approximate reasoning intelligent* *control*) ist fähig, zu lernen und gleichzeitig sein Wissen in Form von Fuzzy-Regeln zu repräsentieren. Es ist also in der Lage, auf eventuell vorhandene Kontrollregeln aufzubauen und dadurch schneller zu einem Lernerfolg zu gelangen.

Das ARIC-System besteht dabei aus zwei getrennten mehrschichtigen Neuronalen Netzen, dem AEN (*action-state evaluation Network*) und dem ASN (*action selection network*). Das ASN modelliert den Fuzzy-Inferenzmechanismus. Die Neuronen aus der Eingabeschicht repräsentieren hier die Fuzzy-Mengen der Eingabevariablen. Die verschiedenen Kontrollregeln werden durch die Neuronen der inneren Schicht dargestellt. Aus den einzelnen Regelausgaben wird durch die Neuronen der Ausgabeschicht ein scharfer Ausgabewert berechnet.

Das AEN berechnet aus dem Zustand des zu regelnden Systems, also im allgemeinen aus den Meßwerten, und einem externen Fehlersignal ein Verstärkungssignal. Mit dessen Hilfe werden dann die Gewichte innerhalb des ASN modifiziert. Daher handelt es sich hier um verstärkendes Lernen (vergleiche Abschnitt 2.2.2).

Im wesentlichen werden durch diesen Regler alle eingangs erwähnten Vorteile erzielt. Es ist fähig zu lernen, vordefiniertes Wissen zu verwenden und die Struktur des ASN kann aufgrund der Modellierung von Fuzzy-Regeln leicht interpretiert werden. Dennoch sind einige Nachteile zu erkennen. Der wichtigste ist, daß es durch die Änderung der Gewichte innerhalb des ASN zu Situationen kommt, in denen identische Fuzzy-Mengen von verschiedenen Regeln mit unterschiedlichen Gewichten versehen werden. Aus Sicht der Fuzzy-Mengen-Theorie ist dies semantisch schwer zu interpretieren.

Durch den im folgenden Abschnitt beschriebenen Ansatz wird versucht, auch diesen Nachteil zu umgehen.

3.2 Ein neuronal orientierter Fuzzy-Regler

Das hier beschriebene Modell eines Neuronalen Fuzzy-Reglers von Detlef Nauck ist im Rahmen dieser Diplomarbeit implementiert worden und bildet den Kern des Programmsystems NEFCON-I [Nauck und Kruse, 1992].

Das Hauptanliegen bei der Definition dieses neuronal orientierten Fuzzy-Reglers ist, wie der Name schon sagt, die Struktur eines Fuzzy-Reglers zu erhalten. Der Lernalgorithmus ändert lediglich die Wissensbasis, ohne strukturelle Veränderungen nach sich zu ziehen. Dem Lernverfahren liegt ein sogenanntes *Fuzzy-Fehlermaß* zugrunde, das sich aus dem Zustand des zu regelnden Systems berechnet. Abhängig davon, ob sich der Zustand des Systems verbessert oder verschlechtert, wird die Form der Zugehörigkeitsfunktionen geändert, die an der letzten Ausgabe beteiligt waren.

Der Neuronale Fuzzy-Regler besitzt also die Fähigkeit, auf eine vorgegebene Menge von Kontrollregeln aufzubauen und sein Regelverhalten zu verbessern. Wenn keine

Kontrollregeln bekannt sind, besteht außerdem die Möglichkeit, diese vom System erlernen zu lassen. Der dafür zuständige Lernalgorithmus wird später in diesem Abschnitt noch näher beschrieben. Zunächst soll hier auf die Architektur des Reglers und das Lernverfahren zur Optimierung der Zugehörigkeitsfunktionen eingegangen werden.

3.2.1 Die Architektur des Neuronalen Fuzzy-Reglers

In der folgenden Beschreibung wird von einem zu regelnden System ausgegangen, das n Meßwerte $x_1 \in X_1, \ldots, x_n \in X_n$ liefert und einen Stellwert $y \in Y$ benötigt.

Für den Lernalgorithmus ist es, wie wir später noch sehen werden, wichtig, daß die Defuzzifizierung eindeutig invertierbar ist. Aus diesem Grund beschränkt man sich bei der Modellierung der linguistischen Terme von Ausgabevariablen auf monotone Zugehörigkeitsfunktionen. Es werden die sogenannten *Tsukamoto's monotonic membership functions* verwendet. Eine solche Funktion μ wird durch zwei Punkte a, b mit $\mu(a) = 0$ und $\mu(b) = 1$ charakterisiert. Sie ist definiert als:

$$\mu(x) \;=\; \begin{cases} \frac{-x+a}{a-b} & \text{falls} \quad (x \in [a,b] \,\wedge\, a \leq b) \,\vee\, (x \in [b,a] \,\wedge\, a > b) \\ 0 & \text{sonst} \end{cases}$$

Die Defuzzifizierung vereinfacht sich hier zu einer Berechnung des Inversen:

$$y \;=\; \mu^{-1}(r) \;=\; -r(a-b) + a,$$

wobei r den Treffergrad angibt.

Eine mögliche Struktur für einen solchen Regler ist in Abb. 3.1 zu sehen. Das dargestellte System enthält zwei Eingabevariablen, fünf Kontrollregeln und eine Ausgabevariable. Man kann diesen Regler nun einerseits als Neuronales Netz mit Fuzzy-Mengen als Gewichten und andererseits als Fuzzy-Regler interpretieren. Die gewählte Darstellung legt eine Interpretation als Neuronales Netz nahe.

Man erkennt drei Schichten, nämlich eine Eingabeschicht, eine mittlere Schicht und eine Ausgabeschicht. Die Elemente der Eingabeschicht entsprechen den beiden Eingabevariablen ξ_1 und ξ_2. Sie sind über die Fuzzy-Mengen $\mu_1^{(1)}, \ldots, \mu_3^{(1)}$ bzw. $\mu_1^{(2)}, \ldots, \mu_3^{(2)}$ mit den Elementen der mittleren Schicht $R_1, \ldots, R_5$ verbunden. Die Elemente $R_1, \ldots, R_5$ modellieren die fünf Kontrollregeln. Sie können zur Eingabeschicht mehrere Verbindungen haben und sind über eine der Fuzzy-Mengen $v_1, \ldots, v_3$ mit der Ausgabeschicht verbunden. Diese besteht nur aus einem Element, das die Ausgabevariable η repräsentiert.

Die Fuzzy-Mengen $\mu_1^{(1)}, \ldots, \mu_3^{(1)}$ bzw. $\mu_1^{(2)}, \ldots, \mu_3^{(2)}$ partitionieren hier die Wertebereiche X_1 und X_2 der Eingabevariablen ξ_1 und ξ_2. Für den Wertebereich Y der Ausgabevariablen η sind entsprechend die Fuzzy-Mengen $v_1, \ldots, v_n$ vorgesehen. Die

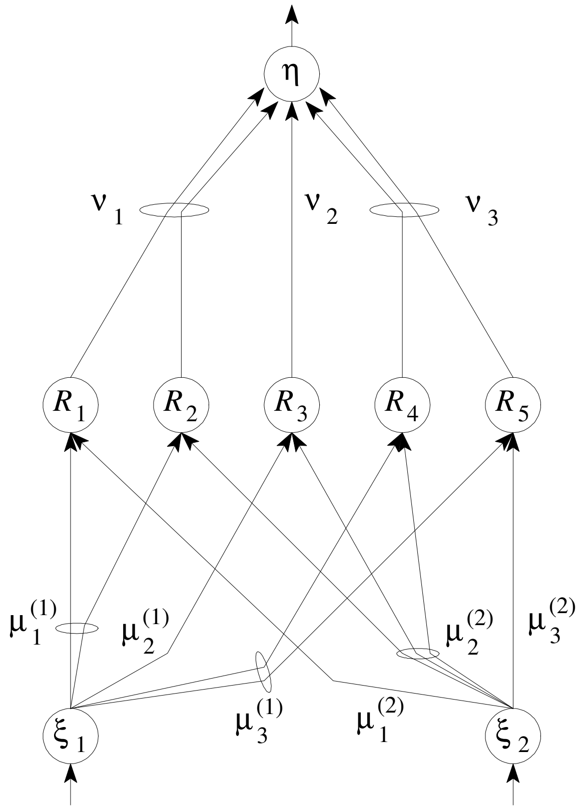


Abbildung 3.1: Struktur des Neuronalen Fuzzy-Reglers

einzelnen Regelknoten sind so zu interpretieren, daß die Verbindungen zur Eingabeschicht die Prämisse und die zur Ausgabeschicht die Konklusion der Regel darstellen. Das Element R_4 der mittleren Schicht kann man also mit folgender Kontrollregel gleichsetzen:

$$\text{R4: } \textbf{IF } \xi_1 \textbf{ IS } \mu_3^{(1)} \textbf{ AND } \xi_2 \textbf{ IS } \mu_2^{(2)} \textbf{ THEN } \eta \textbf{ IS } v_1$$

Die Abläufe innerhalb dieses Reglers entsprechen denen in einem vorwärtsgerichteten Neuronalen Netz. Eingehende Meßwerte werden als Aktivierungen in den Elementen der Eingabeschicht repräsentiert. Die Zugehörigkeitsgrade $\mu_i^{(j)}(x_j)$ der Meßwerte x_j zu den jeweiligen Fuzzy-Mengen werden dann an die Regelknoten weitergeleitet.

Die Aktivierungen der Elemente $R_1, \ldots, R_5$ sind als Erfülltsheitsgrade der einzelnen Regeln zu verstehen. Sie berechnen sich durch die konjunktive Verknüpfung der eingehenden Zugehörigkeitsgrade und werden an das Ausgabeelement weitergeleitet. Dieses berechnet für jede Regel R_i eine scharfe Regelausgabe o_i, indem die Zugehörigkeitsfunktion der Fuzzy-Menge v_{R_i} an der Stelle o_i invertiert wird. Die Gesamtausgabe y ergibt sich nun als der mit den einzelnen Erfülltsheitsgraden gewichtete Mittelwert dieser scharfen Ausgaben:

$$y = \frac{\sum_{i=1}^{n} o_i \ v_{R_i}^{-1}(o_i)}{\sum_{i=1}^{n} o_i},$$

wobei durch n die Anzahl der Regeln angegeben wird.

Eine Defuzzifizierung wie bei einem Mamdani-Regler entfällt also, da hier als Ergebnis des Inferenzmechanismus keine Fuzzy-Menge vorliegt. Die Berechnung des Stellwertes entspricht eher dem Ansatz von Sugeno.

Bei dieser Architektur ist außerdem noch zu beachten, daß es sich bei den Fuzzy-Mengen, wenn man sie als Gewichte einzelner Verbindungen betrachtet, um gemeinsame Gewichte handelt. Das heißt, jede Fuzzy-Menge existiert nur einmal, kann aber als Gewicht für mehrere Verbindungen eingesetzt werden.

3.2.2 Der Lernalgorithmus

Der hier beschriebene Lernalgorithmus hat zum Ziel, das Regelverhalten durch Veränderung der Fuzzy-Mengen zu verbessern. Eine Basis von Kontrollregeln wird dabei als bekannt vorausgesetzt. Später in diesem Kapitel wird ein Algorithmus beschrieben, mit dem auch eine solche Basis von Regeln erlernt werden kann.

Die Definition des Fehlers

Die Aufgabe des Reglers ist es, das zu regelnde System in einen gewünschten Zustand zu steuern. Die Güte des Regelverhaltens mißt sich daran, inwieweit ihm das gelingt. Um sie während der Lernphase zu verbessern, muß der Regler in der Lage sein, jeden der berechneten Stellwerte zu beurteilen. Er benötigt einen Fehler, der angibt, wie *falsch* oder wie *richtig* ein einzelner Wert ist. Am einfachsten wäre es, diesen Fehler, ähnlich wie beim Backpropagation-Algorithmus, aus der Differenz des aktuellen Stellwertes zum optimalen Stellwert zu berechnen. Da dieser optimale Wert aber meist nicht bekannt ist, berechnet sich der Fehler in dem hier beschriebenen Ansatz aus dem Zustand des zu regelnden Systems, also letztlich aus den Meßwerten. In dieser Beziehung ähnelt der Lernalgorithmus eher dem verstärkenden Lernen.

Da man keine feste Grenze ziehen kann, ab der ein bestimmter Systemzustand als *gut* bzw. als *nicht gut* bezeichnet werden kann, bietet es sich an, seine Güte durch Fuzzy-Mengen zu modellieren. Daraus läßt sich dann ein *Fuzzy-Fehler* ableiten, der die Leistung des Reglers beschreibt.

Bei der Festlegung der Systemzustände, die als *gut* zu bezeichnen sind, lassen sich zwei Fälle unterscheiden:

1. Die Meßwerte $x_1, \ldots, x_n$ entsprechen in etwa den optimalen bzw. den gewünschten Werten. Für diesen Fall wird eine *optimale Fuzzy-Güte* G_{opt} wie folgt definiert:

$$G_{opt} \stackrel{\text{Df}}{=} min\{\mu_{opt}^{(1)}(x_1), \ldots, \mu_{opt}^{(n)}(x_n)\},$$

   wobei durch die Fuzzy-Menge $\mu_{opt}^{(i)}$ der gewünschte Wert oder Wertebereich der Meßgröße ξ_i beschrieben wird.

2. Die Meßwerte weichen vom Optimum ab, kompensieren sich aber dahingehend, daß das System dem optimalen Zustand entgegenstrebt. Bei s kompensatorischen Situationen $\mu_{komp}^{(1)}$ wird die *kompensatorische Fuzzy-Güte* G_{komp} definiert als:

$$G_{komp} \stackrel{\text{Df}}{=} min\{\mu_{komp}^{(1)}(x_1, \ldots, x_n), \ldots, \mu_{komp}^{(s)}(x_1, \ldots, x_n)\}$$

Aus diesen beiden Maßen definiert sich die *System-Fuzzy-Güte* G als

$$G \stackrel{\text{Df}}{=} g(G_{opt}, G_{komp}).$$

Die Funktion g ist dabei in Abhängigkeit des zu regelnden Systems zu wählen. Der Fuzzy-Fehler E kann nun als

$$E = 1 - G$$

berechnet werden.

Neben diesem Fehler ist noch eine weitere Information notwendig, um durch Änderungen der Fuzzy-Mengen die Regelgüte verbessern zu können. Es muß bekannt sein, ob der optimale Stellwert η_{opt} für einen gegebenen Systemzustand positiv oder negativ ist. In NEFCON-I wird diese Information zusammen mit dem Fuzzy-Fehler zu einem *Gesamtfehler* $E_{ges} \in [-1, 1]$ zusammengefaßt.

$$E_{ges} = E \cdot sgn(\eta_{opt})$$

Dieser Fehler wird mit Hilfe von linguistischen Regeln festgelegt, die von einem separaten Fuzzy-Regler ausgewertet werden. Als Eingaben benötigt er die Meßwerte des Systems und liefert als Ausgabe den Fehler E_{ges}.

Das Optimieren der Fuzzy-Mengen

Der Lernalgorithmus ist anhand des Fehlers E_{ges} nun in der Lage, den zuletzt ausgegebenen Stellwert zu beurteilen. Ein Fehler E_{ges} mit positivem Vorzeichen bedeutet, daß bei den aktuellen Meßwerten ein Stellwert optimal wäre, der größer ist als Null. Daraus läßt sich folgern, daß der Stellwert, der aus den vorangegangenen Meßwerten berechnet wurde, nicht groß genug war. Es muß deshalb versucht werden, den

Stellwert für diesen Systemzustand zu vergrößern. Im Falle eines negativen Fehlers E_{ges} wird umgekehrt vorgegangen.

Da jede Regeleinheit R_r einen scharfen Ausgabewert liefert, kann entschieden werden, ob sie einen positiven oder einen negativen Einfluß auf die Regelung hatte. Um den Stellwert für einen bestimmten Systemzustand zu erhöhen, werden alle Regeln „belohnt“, deren Ausgabe größer als Null war. Regeln, deren Ausgabe kleiner war, werden entsprechend „bestraft“. Um den Stellwert zu verringern, geht man umgekehrt vor, das heißt, man „belohnt“ die Regeln mit einer negativen Ausgabe und bestraft die, deren Ausgabe positiv war.

Beim Belohnen wird versucht, eine Regel dahingehend zu verändern, daß sich erstens ihr Erfülltheitsgrad o_r bei gleichem Systemzustand erhöht. Zweitens soll der Betrag ihrer Ausgabe $v_i^{-1}(o_r)$ bei gleichem Erfülltheitsgrad steigen. Beim Bestrafen gilt dies entsprechend umgekehrt. Die eigentlichen Änderungen finden dabei in den Fuzzy-Mengen statt, die in der Prämisse und der Konklusion der Regel benutzt werden. In Abb. 3.2 wird darstellt, wie durch Änderungen in ihrer Breite die gewünschten Effekte erzielt werden.

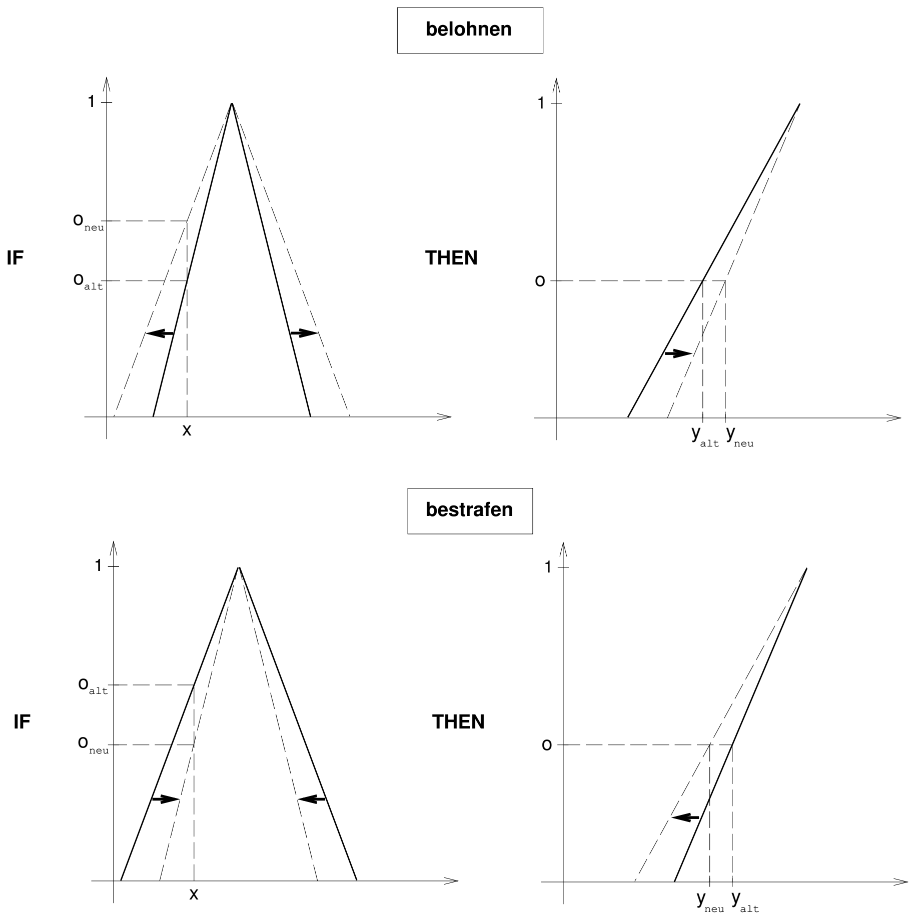


Abbildung 3.2: Belohnen und Bestrafen von Fuzzy-Mengen

Der Erfülltheitsgrad einer Regel wird dadurch erhöht, daß die Fuzzy-Mengen in der Prämisse verbreitert werden. Um ihn herabzusetzen, verringert man ihre Breite. In Abb. 3.2 ist zu sehen, daß so der neue Erfülltheitsgrad o_{neu} beim Belohnen größer und beim Bestrafen kleiner ist als o_{alt}.

Zur Erhöhung der Regelausgabe wird die Breite der Fuzzy-Menge in der Konklusion verringert. Um die Regelausgabe zu senken, wird die Fuzzy-Menge verbreitert. Wie in Abb. 3.2 erkennbar ist, liegt dadurch der neue Stellwert y_{neu} beim Belohnen über und beim Bestrafen unter dem alten Wert y_{alt}.

In die Größe der Veränderung einer Fuzzy-Menge der Regel R_r geht der individuelle Beitrag dieser Regel zum Wert der Stellgröße, also der Erfülltheitsgrad, mit ein. Dieser kann aus der scharfen Ausgabe der Regel berechnet werden, da in der Konklusion nur monotone Fuzzy-Mengen vorkommen. Es wird daher ein *Fuzzy-Regel-Fehler*

$$E_{R_r} \stackrel{\text{Df}}{=} o_{R_r} \cdot E_{ges}$$

definiert, der natürlich auch vom Fuzzy-Fehler des Systems E_{ges} abhängig ist.

Um die Breite einer Fuzzy-Menge μ zu verändern, die durch die beiden Punkte a, b, mit $\mu(a) = 0$ und $\mu(b) = 1$ definiert ist, wird der Punkt a verschoben. Für eine Fuzzy-Menge v_k in der Konklusion einer Regel R_r ergibt sich damit folgende Änderung:

$$a_k^{\text{neu}} = \begin{cases} a_k - \sigma E_{R_r}|a_k - b_k| & \text{falls } a_k < b_k \\ a_k + \sigma E_{R_r}|a_k - b_k| & \text{sonst,} \end{cases}$$

dabei stellt σ die Lernrate oder den Lernfaktor dar.

Eine Fuzzy-Menge $\mu_{k_i}^{(i)},\ k_i \in \{1, \ldots, s_i\}$ der Eingabevariablen ξ_i, wobei s_i die Anzahl der Fuzzy-Mengen dieser Variablen angibt, ändert sich wie folgt:

$$a_{k_i}^{(i)\text{neu}} = \begin{cases} a_{k_i}^{(i)} + \sigma E_{R_r}|a_{k_i}^{(i)} - b_{k_i}^{(i)}| & \text{falls } a_{k_i}^{(i)} < b_{k_i}^{(i)} \\ a_{k_i}^{(i)} - \sigma E_{R_r}|a_{k_i}^{(i)} - b_{k_i}^{(i)}| & \text{sonst} \end{cases}$$

R_r bezeichnet hier die Regel, in deren Prämisse die zu ändernde Fuzzy-Menge benutzt wird. Falls eine Fuzzy-Menge in mehreren Regeln auftaucht, wird sie entsprechend oft geändert.

Das Lernen von Kontrollregeln

Als Erweiterung des Lernalgorithmus ist es ebenfalls möglich, Fuzzy-Regeln vom Neuronalen Fuzzy-Regler erlernen zu lassen. Dazu erzeugt der Regler beim Start eine Regelbasis, die alle Regeln enthält, die sich aus der Kombination aller Fuzzy-Mengen der Ein- und Ausgabevariablen bilden lassen. Während des Lernvorgangs entfernt er dann die Regeln, die er für falsch oder überflüssig hält.

Der Lernvorgang gliedert sich in drei Phasen:

- Phase 1
  In dieser Phase werden alle Regeln entfernt, deren Ausgaben ein vom Fehler E_{ges} verschiedenes Vorzeichen aufweisen. Zusätzlich durchlaufen die Regeleinheiten einen Alterungsprozeß. Regeln, die nur selten oder während eines vorgegebenen Zeitraumes gar nicht benutzt wurden, werden dann ebenfalls entfernt. Nach Abschluß dieser Lernphase enthält die Regelbasis zwar noch mehrere Regeln mit gleicher Prämisse, die Vorzeichen ihrer Ausgabewerte sind jedoch identisch.

- Phase 2
  Hier wird von allen Regeln, die die gleiche Prämisse aufweisen, diejenige ausgesucht, die für die geeignetste gehalten wird. Alle anderen Regeln werden gelöscht. Für jede Regeleinheit wird dazu während dieser Lernphase ein Fehler akkumuliert. Dieser ergibt sich aus der Summe der mit den Erfüllheitsgraden gewichteten Fehler E_{ges}. Am Ende der Lernphase werden dann nur die Regeln mit der kleinsten Fehlersumme beibehalten.

  Damit während dieser Phase möglichst alle Stellgrößen gleich oft „ausprobiert" werden, beteiligt sich jede Regel mit der Wahrscheinlichkeit $\frac{2}{n}$ am aktuellen Zyklus. Dabei gibt n die Anzahl der möglichen Konklusionen an.

- Phase 3
  In dieser Phase werden die Fuzzy-Mengen, wie im letzten Unterabschnitt beschrieben, optimiert.

Kapitel 4

NEFCON-I

NEFCON-I steht für *neuronal fuzzy controller*. Dabei handelt es sich um ein Programmsystem, das den im letzten Kapitel beschriebenen Neuronalen Fuzzy-Regler implementiert. Es wird, aufbauend auf die InterViews-Grafikklassen, eine grafische Benutzeroberfläche zur Verfügung gestellt, die es ermöglicht, die Wissensbasis in Form von linguistischen Variablen und Regeln zu definieren. Der eigentliche Regler kann dann mit verschiedenen Optionen gestartet werden, wobei sein Verhalten während des Regelungsvorgangs grafisch veranschaulicht wird. Es ist so möglich, die Wirkung einzelner Komponenten der Wissensbasis auf das Regelverhalten zu untersuchen. Außerdem werden die Änderungen, die während des Lernvorgangs an den Fuzzy-Mengen oder in der Regelbasis vorgenommen werden, unmittelbar sichtbar gemacht.

Theoretisch lassen sich beliebige Anwendungen durch NEFCON-I regeln, die durch eigenständige Programme repräsentiert werden. Als Beispiel für ein solches Programm wurde im Rahmen dieser Diplomarbeit die in Kapitel 6 beschriebene Simulation eines inversen Pendels implementiert.

Dieses Kapitel dient als Benutzerhandbuch für NEFCON-I. Es wird erklärt, welche Angaben der Regler für die Steuerung einer Anwendung erwartet und wie diese mit Hilfe der grafischen Benutzeroberfläche erstellt werden. Außerdem werden die Anforderungen beschrieben, die das zu regelnde Anwendunsprogramm erfüllen muß.

4.1 Prozeßregelung mit NEFCON-I

Der schematische Aufbau einer Prozeßregelung wird in Abb. 4.1 gezeigt. Man erkennt eine Gliederung in drei Teile :

- Der Prozeß
  Prozesse, die mit NEFCON-I geregelt werden sollen, müssen als eigenstän-

dige Programme vorliegen, die ihre Meßwerte in den Standard-Ausgabekanal schreiben und die Stellwerte aus dem Standard-Eingabekanal lesen.

- Der Neuronale Fuzzy-Regler
  Er ist der eigentliche Kern von NEFCON-I. Er übernimmt die Meßwerte vom zu regelnden Prozeß und liefert als Ausgabe entsprechende Stellwerte. Wie im letzten Kapitel beschrieben, werden die Fuzzy-Mengen der linguistischen Variablen anhand des Fehlers E_{ges} optimiert. Es ist möglich, mit Hilfe des Fehlers eine Menge von Kontrollregeln zu erlernen. Da sowohl die Berechnung der Stellwerte als auch des Fehlers auf linguistischen Variablen und Regeln basiert, bietet es sich an, zwei Fuzzy-Regler auf diese Aufgaben anzusetzen. Der hier implementierte Neuronale Fuzzy-Regler enthält also zwei „einfache" Fuzzy-Regler und eine dritte Komponente. Diese übernimmt die Auswertung des Fehlers und der Steuer-Informationen und veranlaßt entsprechende Änderungen in der Wissensbasis. Die Steuer-Informationen geben dabei Aufschluß über die Beteiligung einzelner Regeln an den Stellwerten.

- Die Wissensbasis
  Da der Neuronale Fuzzy-Regler intern zwei „einfache" Fuzzy-Regler enthält, sind in seiner Wissensbasis zwei Sätze von linguistischen Variablen und Regeln enthalten. Die einen dienen zur Berechnung des Stellwertes und die anderen zu der des Fehlers. Dementsprechend wird zwischen *Steuer-Variablen* und *Fehler-Variablen* bzw. *Steuer-Regeln* und *Fehler-Regeln* unterschieden.

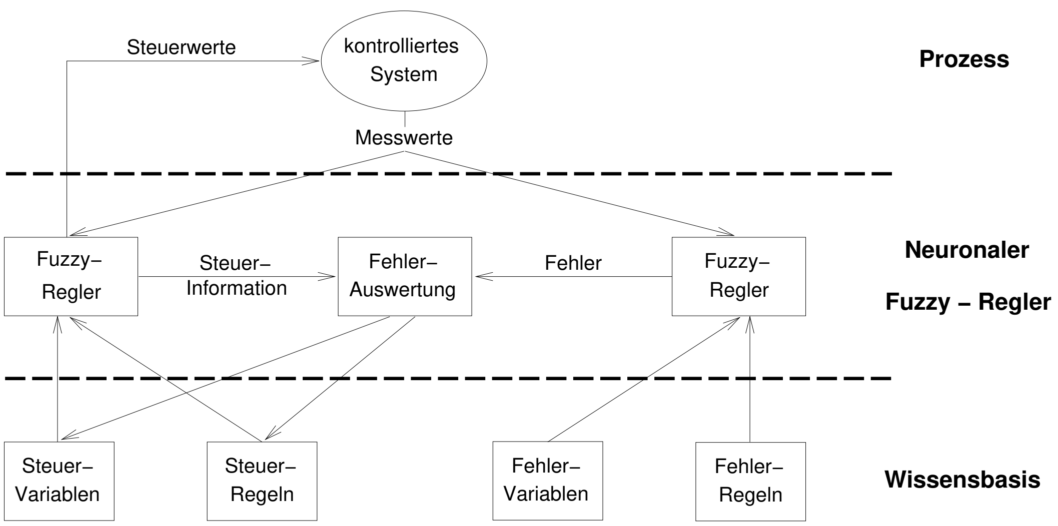


Abbildung 4.1: Aufbau der Prozeßregelung

Der Ablauf der eigentlichen Regelung ist sehr einfach. Das Programm, welches den zu regelnden Prozeß repräsentiert bzw. simuliert, es wird im folgenden als Anwendungsprogramm bezeichnet, wird von NEFCON-I aus gestartet. Dabei wird die Standard-Ein- und Ausgabe auf einen gemeinsamen Kanal „umgebogen". Sobald das Anwendungsprogramm Meßwerte in diesen Kanal schreibt, antwortet der Regler mit entsprechenden Stellwerten. Das ausführliche Protokoll der Prozeßkommunikation ist Anhang C zu entnehmen.

Voraussetzung für die Regelung eines gegebenen Anwendungsprogramms ist jedoch eine geeignete Wissensbasis. NEFCON-I stellt verschiedene Eingabefenster, sogenannte Editoren, zur Verfügung, mit der die verschiedenen Komponenten der Wissensbasis definiert werden können. Im folgenden wird die Bedeutung, die den Komponenten im einzelnen zukommt, näher beschrieben.

Steuer-Variablen und Steuer-Regeln

Die Steuer-Variablen modellieren die Meß- und Stellwerte des zu regelnden Prozesses. Zu ihrer Definition gehört unter anderem die Angabe eines Variablentyps, der angibt, ob es sich um Ein- oder Ausgabevariablen handelt. Über die Anzahl der angegebenen Variablen des jeweiligen Typs wird dem Regler also die Anzahl der Meß- und Stellwerte mitgeteilt. Es muß daher mindestens eine von jedem Typ vorhanden sein.

Jede linguistische Variable enthält mindestens eine Fuzzy-Menge. Die Form der Fuzzy-Mengen unterliegt dabei, wie im letzten Kapitel beschrieben, gewissen Beschränkungen. Und zwar dürfen bei den Eingabevariablen nur Funktionen verwendet werden, die sich aus maximal zwei linear verlaufenden Intervallen zusammensetzen, wie zum Beispiel Dreiecksfunktionen. Die Ausgabevariablen sind dagegen auf monotone Funktionen beschränkt.

Die Steuer-Regeln stellen die Kontrollregeln dar, mit deren Hilfe die Stellwerte berechnet werden. Sie bauen daher auf den Steuer-Variablen auf, das heißt, sie benutzen als linguistische Terme die Namen der Steuer-Variablen bzw. die von deren Fuzzy-Mengen. Die Anzahl der Regeln ist beliebig. Da der Neuronale Fuzzy-Regler in der Lage ist, selbst Regeln zu erlernen, kann die Angabe der Steuer-Regeln auch vollständig entfallen.

Fehler-Variablen und Fehler-Regeln

Der Fehler wird, wie bereits erwähnt, mit Hilfe von linguistischen Termen beschrieben, die durch Fuzzy-Mengen modelliert werden. Diese Fuzzy-Mengen sind in den Fehler-Variablen enthalten. Genau wie bei den Steuer-Variablen ist auch hier eine Aufteilung in Ein- und Ausgabevariablen nötig.

Da die Berechnung des Fehlers auf den Meßwerten des zu regelnden Systems beruht, werden diese auch hier durch Eingabevariablen modelliert. Der Fehler selbst wird durch eine Ausgabevariable modelliert, die den Namen *fehler* trägt.

Die Fehler-Regeln definieren mit Hilfe der Fehler-Variablen die Berechnungsvorschrift für den Fehler. Dabei sollte darauf geachtet werden, daß alle Systemzustände durch Regeln abgedeckt sind, in denen sich das System in einem nicht gewünschten Zustand befindet. Denn für jede Situation, in der sich keine passende Regel anwenden läßt, ist der Fehler gleich Null.

4.2 Programmstart

Der Befehl zum Aufruf von NEFCON-I und seine wichtigsten Optionen lauten:

`nefcon` [-dir=**Verzeichnis**]
[-p=**Datei**]
[-e]
[...]

Das Verzeichnis, das beim Aufruf optional angegeben werden kann, bezieht sich auf das Laden und Speichern von Dateien (siehe Abschnitt 4.8). Der Dateiname spezifiziert die zu verwendende Protokoll-Datei (siehe Abschnitt 4.7.1). Bei Verwendung der Option `-e` erscheinen alle in NEFCON-I verwendeten Texte in englischer Sprache. Eine eingehendere Beschreibung der einzelnen Optionen und der zugehörigen X11-Resourcen findet sich in Anhang A. Beim Aufruf ohne Parameter erscheint das NEFCON-I Hauptfenster, wie es in Abb. 4.2 zu sehen ist.

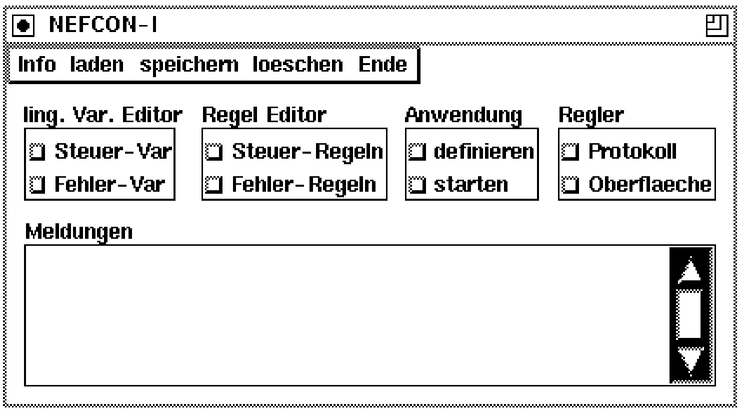


Abbildung 4.2: Das NEFCON-I Hauptfenster

Das Hauptfenster enthält eine Menüleiste, verschiedene Buttons und ein Ausgabefeld für diverse Meldungen. Es dient als Plattform für den Aufruf der einzelnen Editoren und grafischen Oberflächen, die dann jeweils in separaten Fenstern erscheinen. Um den Regelungsvorgang für ein beliebiges Anwendungsprogramm zu starten, sind folgende Schritte durchzuführen:

1. Definition einer Wissensbasis
2. Spezifikation eines Anwendungsprogramms
3. Starten des Reglers
4. Starten des Anwendungsprogramms

Der Aufruf der entsprechenden Eingabefenster erfolgt über verschiedene Buttons, die in vier Gruppen angeordnet sind:

- Steuer-Var und Fehler-Var
  Durch diese beiden Buttons wird der Editor für linguistische Variablen aufgerufen. Mit Hilfe dieses Editors können linguistische Variablen definiert werden. Abhängig davon, welcher Button zum Aufruf benutzt wurde, werden die neu definierten Variablen als Steuer-Variablen oder als Fehler-Variablen in der Wissensbasis abgelegt.

  Der Editor kann auch dazu benutzt werden, vorhandene Variablen zu modifizieren oder näher zu untersuchen. Letzteres ist beispielsweise interessant, um die Auswirkungen, die der Lernalgorithmus während des Regelungsvorgangs auf die Fuzzy-Mengen hatte, deutlicher sichtbar zu machen.

  Wenn der Aufruf des Editors über den Button Steuer-Var erfolgt, werden evtl. in der Wissensbasis vorhandene Steuer-Variablen in den Editor geladen. Bei der Verwendung des Buttons Fehler-Var wird ebenfalls zunächst geprüft, ob sich schon entsprechende Variablen in der Wissensbank befinden. Diese werden jedoch nur dann in den Editor geladen, wenn sie zu den Steuer-Variablen kompatibel sind. Das heißt, sie müssen erstens Eingabevariablen mit den gleichen Namen besitzen. Zweitens darf in den Fehler-Variablen nur eine Ausgabevariable enthalten sein, und diese muß den Namen *fehler* tragen.

  Für den Fall, daß die Wissensbasis keine oder nur ungültige Fehler-Variablen enthält, werden diese automatisch aus den Steuer-Variablen erzeugt. Der Benutzer muß dann nur noch die Partitionierung der Fehler-Variablen durch Fuzzy-Mengen ergänzen.

- Steuer-Regeln und Fehler-Regeln
  Diese beiden Buttons rufen den Editor zur Definition der linguistischen Regeln auf. Je nach Button gilt auch hier der Aufruf entweder den Steuer-Regeln oder den Fehler-Regeln.

  In der Wissensbank vorhandene Regeln werden nur dann in den Editor geladen, wenn sie mit den Steuer-Variablen bzw. den Fehler-Variablen kompatibel sind. In den Regeln dürfen also keine Variablen oder Fuzzy-Mengen verwendet werden, die nicht definiert sind.

- definieren und starten
  Beide Buttons beziehen sich auf das von NEFCON-I zu regelnde Anwendungsprogramm. definieren erzeugt ein Eingabefenster, in dem sowohl der Name des Programms als auch die Reihenfolge der Meß- und Stellgrößen festgelegt werden kann. Durch starten wird dieses Programm dann als eigener Prozeß gestartet.

- Protokoll und Oberfläche
  Durch den Button Oberfläche wird die sogenannte *Regleroberfläche* erzeugt. Von hier aus kann der eigentliche Regler mit verschiedenen Optionen gestartet werden. Die Oberfläche enthält unter anderem das *grafische Ausgabeprotokoll*. Dieses veranschaulicht, welche Regeln und Fuzzy-Mengen während des Regelungsvorgangs benutzt werden. Außerdem werden die Meß- und Stellwerte zu den einzelnen Situationen angezeigt.

  Die Art der grafischen Darstellung kann in gewissen Grenzen variiert werden. Entsprechende Angaben können über den Button Protokoll gemacht werden.

Beim Aufruf der angesprochenen Fenster existieren gewisse Einschränkungen. So kann beispielsweise kein Regel-Editor erzeugt werden, solange die zugehörigen linguistischen Variablen nicht definiert sind. Der Grund liegt darin, daß dann Informationen fehlen, die zum Aufbau der Grafik und zur Überprüfung der Gültigkeit eingegebener Regeln benötigt werden. Bezüglich der Reihenfolge, in der man die unterschiedlichen Eingabefenster aufrufen kann, gelten daher zwei Regeln:

1. Zuerst müssen Steuer-Variablen vorhanden sein.
2. Die Fehler-Variablen müssen vor den Fehler-Regeln definiert werden.

Da die Definition der Wissensbasis relativ arbeitsintensiv ist, besteht die Möglichkeit, die einzelnen Komponenten in Dateien abzulegen. Bei späteren Programmstarts können sie dann wieder eingeladen werden. Dies hat auch den Vorteil, daß für jede Komponente mehrere Versionen gespeichert werden können. Es ist so leicht möglich, unterschiedliche Versionen, sofern diese kompatibel sind, miteinander zu kombinieren und deren Regelungsverhalten zu testen. Das Laden und Speichern erfolgt über die Menüpunkte laden und speichern (siehe Abschnitt 4.8).

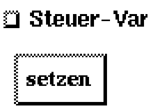
Der Menüpunkt loeschen erlaubt es, einzelne Komponenten der Wissensbasis zu löschen. Dies bezieht sich jedoch nicht auf evtl. vorhandene Dateien. Über Ende wird schließlich das gesamte Programm beendet.

Im nächsten Abschnitt werden zunächst allgemeine Bedienungselemente der grafischen Benutzeroberfläche beschrieben. Danach wird die Bedienung der Editoren für linguistische Variablen und Regeln näher erläutert. Die darauf folgenden Abschnitte beschäftigen sich dann mit dem Start von Regler und Anwendungsprogramm. Eine Beschreibung der Dateiarbeit und ein ausführliches Beispiel sind in den beiden letzten Abschnitten dieses Kapitels enthalten.

4.3 Allgemeine Bedienungselemente

Die grafische Benutzeroberfläche von NEFCON-I setzt sich im wesentlichen aus einfachen immer wieder vorkommenden Bedienungselementen zusammen. In diesem Abschnitt soll sowohl eine Bezeichnung für diese Elemente eingeführt als auch deren Bedeutung erklärt werden.

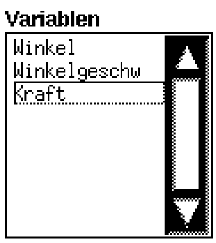
Das Aussehen der abgebildeten Bedienungselemente ist abhängig vom Bildschirm, auf dem das Programm läuft, und den verwendeten Kommandozeilen-Optionen (siehe Anhang A). Die Abbildungen in diesem Abschnitt, wie auch die im restlichen Kapitel, entsprechen dem Erscheinungsbild auf einem monochromen Bildschirm.



Einfache **Buttons** führen nach dem Anklicken mit der Maus bestimmte Aktionen aus. Es existieren zwei Arten, die sich in ihrer Wirkung jedoch nicht unterscheiden. Der obere der beiden hier dargestellten Buttons wird an einigen Stellen von NEFCON-I auch dazu benutzt, durch seine Markierung einen bestimmten Zustand anzuzeigen.

Durch **Auswahl-Buttons** kann genau eine von mehreren Alternativen selektiert werden. Jede Alternative wird dabei durch einen Button repräsentiert und gilt als ausgewählt, wenn dieser markiert ist. Auswahl-Buttons kommen daher immer in Gruppen von mindestens zwei Buttons vor.

Der **Toggle-Button** ändert nach jedem Anklicken mit der Maus den dargestellten Text. Aus einer begrenzten Menge von Möglichkeiten kann so, ähnlich wie bei den Auswahl-Buttons, eine selektiert werden. Dabei gilt diejenige Möglichkeit oder derjenige Text als ausgewählt, der gerade sichtbar ist.



Ein **Auswahlfeld** stellt eine Menge von Texten dar. Falls nicht alle gleichzeitig darstellbar sind, kann mit Hilfe des Scrollbalkens der sichbare Teil der Texte verschoben werden. Die mittlere und rechte Maustaste können innerhalb des Feldes ebenfalls zum scrollen verwendet werden. Durch einen Doppelklick mit der linken Maustaste ist es möglich, einen bestimmten Text zu selektieren. In diesem Beispiel ist der Text `Kraft` ausgewählt.

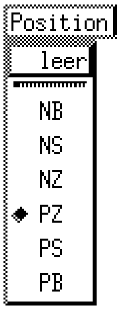
Bezüglich des Aussehens ähneln Auswahlfelder den **Ausgabefeldern**. Ausgabefelder dienen lediglich der Ausgabe von Texten und sind in NEFCON-I immer mit dem Titel `Meldungen` versehen. Innerhalb eines Ausgabefeldes haben die Maustasten keine Bedeutung.

Eingabefelder erlauben die Eingabe beliebiger Zeichenketten. Nach dem Anklicken mit der Maus erscheint der Cursor an der gewünschten Position. Eingaben werden nur angenommen, wenn sich der Mauszeiger in einem Eingabefeld befindet. Mit folgenden Tastenkombinationen kann der Editiervorgang gesteuert werden:

<ctrl>-b :	Cursor ein Zeichen nach links
<ctrl>-f :	Cursor ein Zeichen nach rechts
<ctrl>-a :	an den Anfang der Zeile springen
<ctrl>-e :	an das Ende der Zeile springen
<ctrl>-d :	das Zeichen unter dem Cursor löschen
<ctrl>-h :	das Zeichen links vom Cursor löschen

Der Editiervorgang wird durch Drücken der Returntaste beendet (siehe auch [LINTON et al., 1992]).

Bei den in NEFCON-I verwendeten **Menüs** handelt es sich um herkömmliche Popup-Menüs. Durch die Auswahl von Menüpunkten werden bestimmte Aktionen ausgelöst.



Ein **Auswahlmenü** bietet eine ähnliche Funktionalität wie eine Gruppe von Auswahl-Buttons. Aus einer Menge von angebotenen Alternativen, die durch die einzelnen Menüpunkte repräsentiert werden, kann genau eine ausgewählt werden. Diese wird dann durch eine Markierung gekennzeichnet. In diesem Beispiel ist „PZ“ ausgewählt.

4.4 Der Editor für linguistische Variablen

Der Aufruf des Editors erfolgt im Hauptfenster durch die Buttons Steuer-Var oder Fehler-Var. Abhängig vom Button, bezieht sich der Aufruf des Editors entweder auf die Steuer-Variablen oder auf die Fehler-Variablen.

Falls schon entsprechende Variablen in der Wissensbasis vorhanden sind, wird eine Kopie von ihnen in den Editor geladen. Das heißt, wenn der Editor verlassen wird, ohne eventuelle Änderungen abzuspeichern, bleibt die Wissensbasis unverändert. Dies ist der Fall, wenn der Editor über den Menüpunkt abbrechen verlassen wird. Bei beenden werden dagegen alle Eingaben und Änderungen in die Wissensbasis übernommen.

Der Editor ist in Abb. 4.3 zu sehen. Sein Aufbau gliedert sich in vier Bereiche, die jeweils für sich umrandet sind. Der Bereich links oben im Editorfenster stellt in einem Auswahlfeld die Namen aller bisher definierten Variablen dar. Durch doppeltes Anklicken mit der linken Maustaste kann eine dieser Variablen ausgewählt werden.

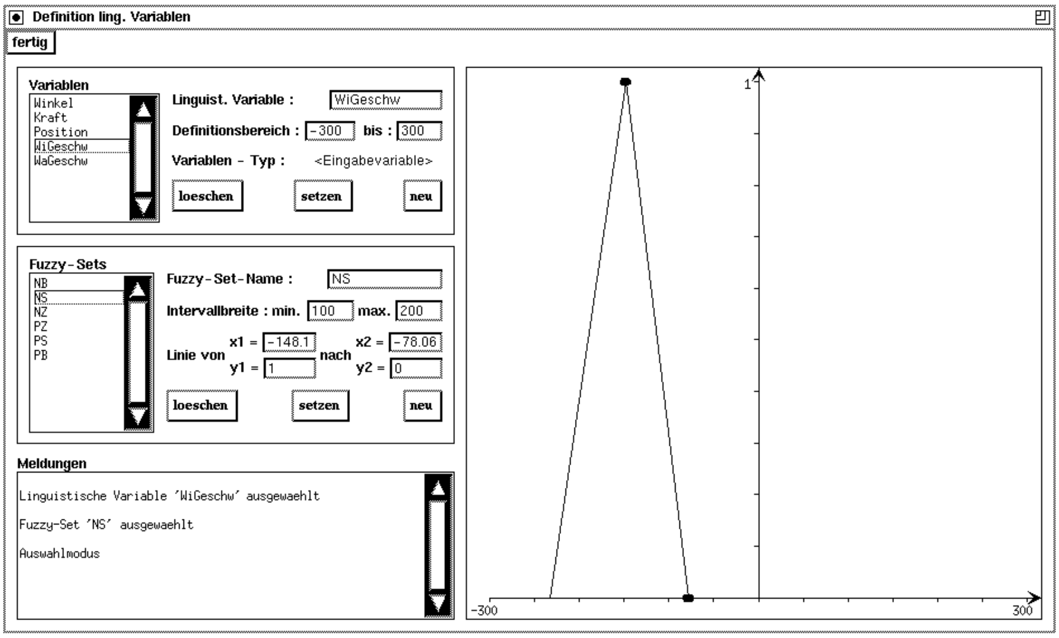


Abbildung 4.3: Der Editor für linguistische Variablen

Sie wird dann durch einen Balken im Auswahlfeld hervorgehoben, wie dies in Abb. 4.3 am Beispiel der Variablen `WiGeschw` zu sehen ist. Außerdem wird rechts neben dem Auswahlfeld der Name, der Definitionsbereich und der Typ dieser Variablen angezeigt.

Der Bereich darunter zeigt in einem Auswahlfeld die Namen aller Fuzzy-Mengen an, die zu der selektierten Variable gehören. Auch hier kann einer der Namen ausgewählt werden. Die Zugehörigkeitsfunktion der entsprechenden Fuzzy-Menge wird dann im Koordinatensystem in der rechten Hälfte des Editorfensters dargestellt. In Abb. 4.3 ist die Fuzzy-Menge `NS` ausgewählt.

Falls der Editor für die Steuer-Variablen aufgerufen wurde, wird als weitere Information die minimale und die maximale Intervallbreite der selektierten Fuzzy-Menge angegeben. Die entsprechenden Felder befinden sich rechts neben dem Auswahlfeld. Diese beiden Werte stecken die Grenzen ab, in denen sich die Intervalle einer Fuzzy-Menge während des Lernalgorithmus verändern dürfen.

Die rechte Maustaste besitzt innerhalb des Koordinatensystems eine besondere Bedeutung. Und zwar wird mit ihr der sogenannte *Auswahlmodus* gestartet bzw. nach einem weiteren Tastendruck beendet. Im Auswahlmodus werden die Anfangs- und Endpunkte der einzelnen Intervalle der Funktion in die vier dafür vorgesehenen Felder ausgegeben. Diese befinden sich rechts neben dem Auswahlfeld für Fuzzy-Mengen. Es wird immer das Intervall dargestellt, das den x-Wert enthält, an dem sich der Mauszeiger in dem Koordinatensystem befindet. Zusätzlich werden die beiden Endpunkte im Koordinatensystem durch zwei Kreise gekennzeichnet. In Abb. 4.3 ist das zweite Teilstück der Zugehörigkeitsfunktion markiert.

Das Ausgabefeld, das sich unten links im Editorfenster befindet, dient zur Ausgabe verschiedener Fehlermeldungen. Außerdem wird für die meisten Aktionen, die der Benutzer im Editor durchführt, eine Bestätigung ausgegeben.

4.4.1 Die Eingabe von Variablen

Um eine neue Variable zu definieren, wird der gewünschte Name in das Eingabefeld eingegeben, das mit `Name der Variable` beschriftet ist. Wenn schon eine Variable mit diesem Namen existiert, wird keine neue Variable erzeugt, sondern die alte Variable selektiert. Dieses Eingabefeld kann also auch, alternativ zur Maus, für die Auswahl von Variablen benutzt werden.

Beim Erzeugen einer neuen Variablen wird diese mit den Werten initialisiert, die in den verschiedenen Eingabefeldern eingetragen sind. Also mit dem angezeigten Definitionsbereich und dem angezeigten Typ. Außerdem werden alle Fuzzy-Mengen übernommen, die in dem entsprechenden Auswahlfeld angezeigt werden. Auf diese Weise ist es leicht möglich, Variablen zu kopieren. Dazu wird eine schon vorhandene Variable selektiert, so daß dessen Werte in den Eingabefeldern erscheinen. Wird dann ein neuer Variablenname eingegeben, übernimmt die neu definierte Variable alle Werte.

Soll eine neu definierte Variable vorerst keine Fuzzy-Mengen enthalten, muß das Auswahlfeld für Fuzzy-Mengen zum Zeitpunkt der Definition leer sein. Erreicht werden kann dies durch Eingabe einer leeren Zeichenkette im Eingabefeld für den Variablennamen. In den Eingabefeldern des Definitionsbereiches werden dann vorgabemäßig die Werte -1 bzw. 1 angezeigt, und der Variablentyp wird auf `Eingabevariable` gesetzt.

In dem Bereich für die Eingabe der Variablen befinden sich drei Buttons, nämlich $\boxed{\text{loeschen}}$, $\boxed{\text{setzen}}$ und $\boxed{\text{neu}}$. Die Wirkung dieser Buttons bezieht sich jeweils auf die gerade ausgewählte Variable. Durch $\boxed{\text{loeschen}}$ wird diese Variable gelöscht und durch $\boxed{\text{setzen}}$ werden eventuelle Änderungen gespeichert. Der Button $\boxed{\text{neu}}$ macht alle Änderungen nach dem letzten Speichern rückgängig. Bei der Auswahl einer neuen Variablen oder bei Beenden des Editors wird die gerade ausgewählte Variable automatisch gespeichert.

Eingabe des Definitionsbereiches

Der Definitionsbereich einer Variablen wird durch den minimalen und den maximalen Wert der Meß- oder Stellgröße festgelegt, den diese Variable modelliert. Die dazugehörigen Eingabefelder befinden sich rechts neben dem Auswahlfeld für die Variablen.

Es ist darauf zu achten, daß der Bereich für eventuell schon vorhandene Fuzzy-Mengen nicht zu klein wird. Wenn eine Fuzzy-Menge existiert, deren Zugehörigkeitsfunktion außerhalb der angegebenen Grenzen definiert ist, ist die Angabe des Definitionsbereiches ungültig. In diesem Fall wird eine entsprechende Fehlermeldung erzeugt, und der alte Bereich wird wieder angezeigt.

Falls ein gültiger Bereich definiert wurde, werden die Achsen des Koordinatensystems entsprechend neu skaliert.

Definition des Variablentyps

Der Variablentyp wird durch einen Toggle-Button bestimmt, der sich unter den Eingabefeldern des Definitionsbereiches befindet. Bei der Angabe des Typs ist zu beachten, daß in der jetzigen Programmversion bei Ausgabevariablen nur monotone Fuzzy-Mengen erlaubt sind. Das heißt, wenn eine Variable Fuzzy-Mengen enthält, die nicht monoton sind, kann sie nicht als Ausgabevariable deklariert werden. Entsprechende Versuche schlagen fehl und führen zu Fehlermeldungen.

4.4.2 Die Eingabe der Fuzzy-Mengen

Vor der Eingabe neuer Fuzzy-Mengen muß eine Variable selektiert werden. Alle eingegebenen Fuzzy-Mengen sind dann Bestandteil dieser Variablen. Sie werden, ähnlich wie bei der Definition von Variablen, durch die Eingabe eines Namens im entsprechenden Eingabefeld definiert. Falls schon eine Fuzzy-Menge mit dem angegebenen Namen existiert, wird diese selektiert. Die drei Buttons $\boxed{\text{loeschen}}$, $\boxed{\text{setzen}}$ und $\boxed{\text{neu}}$ haben ebenfalls die gleiche Bedeutung wie bei der Definition der Variablen.

Bei den Steuer-Variablen kann optional für jede Fuzzy-Menge eine minimale und maximale Breite angegeben werden. Wie bereits erwähnt, legen diese Werte fest, in welchem Rahmen die Intervalle einer Fuzzy-Menge während des Lernvorgangs veränderbar sind. Die entsprechenden Eingabefelder befinden sich unter dem Feld für den Namen der Fuzzy-Menge. Vorgabemäßig wird als Wert für das Maximum der gesamte Definitionsbereich der Variablen festgelegt. Der Wert für das Minimum berechnet sich dann als $\frac{1}{10}$ des Maximums.

Die Zugehörigkeitsfunktion einer Fuzzy-Menge besteht aus einer Menge von Intervallen, in denen die Funktion linear verläuft. Jedes Intervall wird durch den Anfangs- und Endpunkt dieser Linie definiert. Die Eingabe erfolgt entweder grafisch im Koordinatensystem oder numerisch in den entsprechenden Eingabefeldern.

Für die numerische Definition der einzelnen Linien existieren vier Eingabefelder, eines für jede Koordinate der beiden Punkte. Diese Felder befinden sich neben dem Auswahlfeld für Fuzzy-Mengen. Nach dem Drücken der Return-Taste im letzten Eingabefeld, dies ist das Feld für die y-Koordinate des zweiten Punktes, wird die angegebene Linie übernommen.

Alternativ besteht die Möglichkeit, Zugehörigkeitsfunktionen grafisch mit Hilfe der Maus zu definieren. Die Endpunkte der einzelnen Linien werden dazu einfach mit der linken Maustaste im Koordinatensystem markiert. Die Koordinaten, an denen sich der Mauszeiger befindet, werden in den Eingabefeldern angezeigt, sobald sich diese im Koordinatensystem bewegt. So ist es möglich, die Linien auch grafisch sehr genau zu definieren.

Die mittlere Maustaste hat bei der grafischen Eingabe eine besondere Bedeutung. Wird der zweite Punkt mit dieser Taste festgelegt, verlängert sich die Linie automatisch beidseitig bis an die Grenzen des Definitionsbereiches. Das heißt, in diesem Fall legen die beiden mit der Maus markierten Punkte nur Position und Steigung der Linie fest.

Da jede Funktion injektiv sein muß, wird bei der Eingabe jeder Linie geprüft, ob sie sich mit bereits vorher definierten überschneidet. Ist das der Fall, so wird die neu angegebene Linie übernommen, und die alten werden entsprechend gekürzt oder ganz gelöscht.

Um ein bestimmtes Intervall der Zugehörigkeitsfunktion zu löschen, muß es auf Null gesetzt werden. Man definiert dazu eine „gedachte“ Linie auf der Abszisse des Koordinatensystems. Bei der grafischen Eingabe kann dies einfach dadurch erreicht werden, daß die Punkte unterhalb der Abszisse gesetzt werden. Wird dabei die mittlere Maustaste benutzt, verlängert sich die Linie bis an die Grenzen des Definitionsbereiches, was zur Folge hat, daß die gesamte Funktion gelöscht wird.

4.5 Der Regel-Editor

Der Aufruf des Editors erfolgt im Hauptfenster durch die Buttons $\boxed{\text{Steuer-Regeln}}$ oder $\boxed{\text{Fehler-Regeln}}$. Ähnlich wie beim Editor für linguistische Variablen, gilt der Aufruf entweder den Steuer-Regeln oder den Fehler-Regeln. Das Beenden des Editors erfolgt auch hier über die Menüpunkte $\boxed{\text{beenden}}$ oder $\boxed{\text{abbrechen}}$. Dabei werden bei Abbruch des Editors evtl. Änderungen nicht gespeichert. Der Editor ist in Abb. 4.4 abgebildet.

Man erkennt in der Abbildung vier voneinander getrennte Bereiche. Von oben nach unten sind dies:

Die Regeltafel : In diesem Auswahlfeld werden alle Regeln als Text darstellt. Mit Hilfe der Maus kann eine davon selektiert werden.

Der Eingabebereich für die Regeln : Hier können neue Regeln definiert oder alte gelöscht werden. Die Darstellung einer Regel erfolgt durch Toggle-Buttons und Eingabefelder.

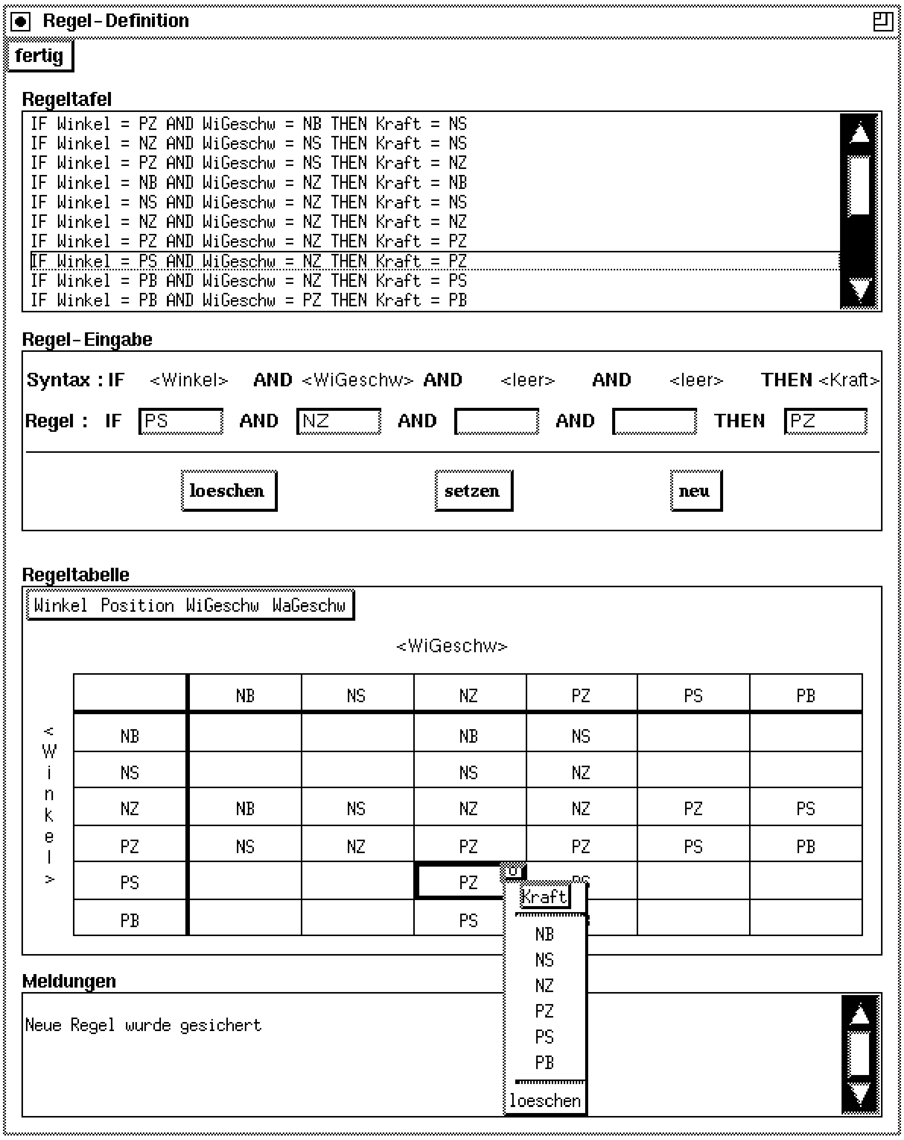


Abbildung 4.4: Der Editor für linguistische Regeln

Die Regeltabelle : Alle Regeln werden hier in einer Tabelle dargestellt. Durch Anklicken mit der Maus kann eine davon selektiert werden. Es ist außerdem möglich, über ein Menü Regeln zu definieren oder zu löschen.

Das Ausgabefeld : Dieses Feld dient der Ausgabe verschiedener Meldungen für den Benutzer.

Grundsätzlich existieren also zwei Alternativen zur Darstellung und Bearbeitung von Regeln. Sowohl die Regeltafel, zusammen mit dem Eingabefeld für Regeln, als auch die Regeltabelle können als eigenständige Regel-Editoren angesehen werden.

Die Regeltabelle hat, zumindest bei Systemen mit zwei Eingabe- und einer Ausgabevariablen, den Vorteil einer sehr übersichtlichen Darstellung von Regeln. Es ist zwar auch möglich, die Regeltabelle einzusetzen, wenn das System mehr Variablen enthält (siehe Abschnitt 4.5.2), jedoch geht dann die Übersichtlichkeit schnell verloren. Regeln, in deren Prämisse nur eine Variable auftaucht, sind dagegen nicht in der Tabelle darstellbar. Das bedeutet, daß Regeln der Form

R: **IF** $var1$ **IS** $fuz1$ **THEN** $var2$ **IS** $fuz2$

nur im *Eingabebereich für Regeln* und nicht in der Tabelle definiert werden können. Dabei sind $var1$ und $var2$ die Namen von zwei Variablen und $fuz1$ und $fuz2$ die Namen zweier Fuzzy-Mengen.

Eine Regelbasis muß immer konsistent sein, sie darf also keine Regeln mit gleicher Prämisse und widersprüchlichen Konklusionen enthalten. Daher wird bei der Eingabe jeder Regel geprüft, ob schon eine Regel mit gleicher Prämisse vorhanden ist. Falls dies der Fall ist, wird die alte Regel aus der Regelbasis entfernt.

Das folgende Beispiel zeigt zwei widersprüchliche Regeln:

R1: **IF** *winkel* **IS** *NE* **AND** *geschw* **IS** *PZ* **THEN** *kraft* **IS** *PS*
R2: **IF** *winkel* **IS** *NE* **AND** *geschw* **IS** *PZ* **THEN** *kraft* **IS** *NZ*

Wenn beide Regeln in dieser Reihenfolge eingegeben werden, wird Regel R1 gelöscht und R2 in die Regelbasis aufgenommen.

In den folgenden Unterabschnitten soll nun die Benutzung der Regeltafel mit dem dazugehörigen Eingabebereich und die der Regeltabelle näher erläutert werden.

4.5.1 Die Regeltafel

Wie bereits erwähnt, entspricht die Regeltafel einem Auswahlfeld, in dem die gesamte Regelbasis dargestellt wird. Die Darstellung von Regeln erfolgt hier, im Gegensatz zur Regeltabelle, als zusammenhängender Text. Die jeweils selektierte Regel wird im Eingabebereich dargestellt und kann dort bearbeitet werden. Sie wird sowohl im Auswahlfeld als auch in der Regeltabelle markiert.

Der Eingabebereich enthält eine Zeile mit Toggle-Buttons und eine Zeile mit Eingabefeldern. Darunter befinden sich die Buttons $\boxed{\text{loeschen}}$, $\boxed{\text{setzen}}$ und $\boxed{\text{neu}}$. Zur Definition einer Regel werden in den Eingabefeldern die Namen der Fuzzy-Mengen eingetragen. Mit Hilfe der Toggle-Buttons, die sich über den Eingabefelder befinden, wird festgelegt, zu welchen Variablen die eingetragenen Fuzzy-Mengen gehören. Ein Paar, bestehend aus Toggle-Button und Eingabefeld, repräsentiert also genau eine Bedingung der Form:

$$\textit{Variable} = \textit{Fuzzy-Menge}$$

Die Regel, die im Eingabebereich von Abb. 4.4 dargestellt wird, lautet also:

IF *Winkel* **IS** *PS* **AND** *WiGeschw* **IS** *NZ* **THEN** *Kraft* **IS** *PZ*

In dieser Regel tauchen in der Prämisse nur zwei der vier Eingabevariablen auf. Um Regeln zu definieren, die nicht alle linguistischen Variablen benutzen, stellt man die Toggle-Buttons, die nicht benötigt werden, so ein, daß sie den Text `leer` anzeigen. Das dazugehörige Eingabefeld darf dann keinen Eintrag enthalten.

Prämisse und Konklusion einer Regel müssen jeweils mindestens eine Bedingung enthalten. Die Toggle-Buttons sind so ausgelegt, daß für die Prämisse nur Eingabe- und für die Konklusion nur Ausgabevariablen ausgewählt werden können. Falls nur eine Variable des entsprechenden Typs existiert, entfällt außerdem die Möglichkeit, den entsprechenden Button auf `leer` zu setzen.

Um eine neue Regel zu erzeugen, muß diese zunächst mit Hilfe der Toggle-Buttons und der Eingabefelder dargestellt werden. Sie wird genau dann in die Regelbasis aufgenommen, wenn die Return-Taste im letzten Eingabefeld betätigt oder der Button $\boxed{\text{setzen}}$ gedrückt wird.

Der Button $\boxed{\text{loeschen}}$ löscht die Regel, die im Eingabebereich dargestellt wird. Um eine bestimmte Regel aus der Regelbasis zu entfernen, wählt man diese zunächst mit der Maus aus und betätigt dann $\boxed{\text{loeschen}}$. Der Button $\boxed{\text{neu}}$ macht alle Änderungen rückgängig, die seit dem letzten Speichern im Eingabebereich gemacht wurden.

4.5.2 Die Regeltabelle

Die Darstellung von Regeln

Die Regeltabelle bietet, im Gegensatz zur Regeltafel, eine eher grafisch orientierte Methode zur Darstellung und Eingabe von Regeln. Wie der Name schon sagt, werden die Regeln hier in einer Tabelle eingetragen. Die Zeilen und Spalten repräsentieren dabei die Fuzzy-Mengen jeweils einer Eingabevariablen. Die Namen dieser beiden Variablen werden durch zwei Toggle-Buttons, die sich über bzw. neben der Tabelle befinden, angezeigt.

Jeder der Einträge in der Tabelle entspricht genau einer Regel. Im einfachsten Fall enthält diese Regel zwei Eingabe- und eine Ausgabevariable. Die Namen für die Fuzzy-Mengen in der Prämisse werden durch Zeile und Spalte des Eintrags festgelegt. Der Name der Fuzzy-Menge in der Konklusion entspricht dem eingetragenen Namen.

Durch das Anklicken einer in der Tabelle dargestellten Regel mit der Maus, wird diese Regel auch in der Regeltafel markiert und im Eingabebereich dargestellt.

Die Tabelle ist für Systeme mit zwei Eingabe- und einer Ausgabevariablen sehr übersichtlich. Falls die Anzahl der Variablen größer wird, ergeben sich jedoch Probleme:

1. Es muß festgelegt werden, welche Variablen in der Tabelle dargestellt werden. Das gilt sowohl für die beiden Eingabevariablen, deren Fuzzy-Mengen zeilen- und spaltenweise am Tabellenrand abgetragen werden, als auch für die Ausgabevariable, deren Fuzzy-Mengen als Einträge innerhalb der Tabelle stehen.

2. Um einen Tabelleneintrag eindeutig einer bestimmten Regel zuordnen zu können, muß für die Variablen, die nicht direkt in der Tabelle dargestellt werden, eine Belegung mit Fuzzy-Mengen vorausgesetzt werden. In der Tabelle werden dann nur Regeln dargestellt, in denen die Variablen, die nicht in der Tabelle erscheinen, mit den vorbestimmten Fuzzy-Mengen „belegt" sind.

Das erste Problem, die Auswahl der darzustellenden Variablen, wird für Eingabe- und Ausgabevariablen auf unterschiedliche Art und Weise gelöst. Die Eingabevariablen, die in der Tabelle darzustellen sind, können mit Hilfe der Toggle-Buttons festgelegt werden, die sich neben der Tabelle befinden. Die Auswahl der Ausgabevariablen erfolgt über ein Auswahlmenü, das in der linken oberen Ecke der Tabelle erscheint, wenn mehr als eine Variable dieses Typs vorhanden ist. In Abb. 4.4 ist das Menü nicht zu sehen, da für dieses Beispiel nur eine Ausgabevariable definiert wurde.

Die Belegung der Variablen, die nicht in der Tabelle dargestellt werden, erfolgt ebenfalls durch Auswahlmenüs. Die Menüleiste befindet sich, wie in Abb. 4.4 zu sehen, über der Tabelle. Wenn mehr als zwei Eingabevariablen definiert sind, enthält die Menüleiste für jede ein Auswahlmenü. Das gleiche gilt für die Ausgabevariablen, wenn davon mehr als eine existiert. Über die Auswahlmenüs können die Variablen nun entweder mit einer ihrer Fuzzy-Mengen belegt oder als `leer` gekennzeichnet werden. Letzteres bedeutet, daß in der Tabelle nur Regeln dargestellt werden, in denen diese Variable nicht auftaucht.

Beispiel

Bei einem System mit den Eingabvariablen *Winkel*, *WiGeschw* und *Position* und der Ausgabevariablen *Kraft* sei folgende Regelbasis definiert:

R1: **IF** *Winkel* **IS** *NE* **AND** *WiGeschw* **IS** *PZ* **THEN** *Kraft* **IS** *PS*
R2: **IF** *Winkel* **IS** *NE* **AND** *WiGeschw* **IS** *PM* **AND** *Position* **IS** *PZ* **THEN** *Kraft* **IS** *NZ*
R3: **IF** *Winkel* **IS** *NE* **AND** *WiGeschw* **IS** *PZ* **AND** *Position* **IS** *PZ* **THEN** *Kraft* **IS** *NZ*

Weiter wird angenommen, daß die Toggle-Buttons so eingestellt sind, daß in der Tabelle die Eingabevariablen *Winkel* und *WiGeschw* dargestellt werden. Wenn nun im Menü die Variable *Position* mit der Fuzzy-Menge *PZ* belegt ist, werden die Regeln R2 und R3 in der Tabelle dargestellt. Falls die Variable mit `leer` belegt ist, erscheint nur die Regel R1. In allen anderen Fällen bleibt die Tabelle leer. Die

Belegung von *Winkel* und *WiGeschw* innerhalb des Menüs ist unerheblich, da beide in der Tabelle enthalten sind.

Die Eingabe von Regeln

Wenn sich die Maus innerhalb der Tabelle bewegt, wird das jeweilige Tabellenfeld durch einen breiteren Rahmen markiert. Außerdem erscheint rechts oben in diesem Feld ein Menü, durch das neue Regeln definiert oder alte gelöscht werden können. In Abb. 4.4 ist dieses Menü im ausgeklappten Zustand zu sehen. Der erste Menüeintrag ist der Name der Ausgabevariablen, die in der Tabelle dargestellt wird. Dieser Eintrag dient lediglich der Information und kann nicht ausgewählt werden. Es folgen die Namen der einzelnen Fuzzy-Mengen und ein Menüeintrag zum Löschen von Regeln.

Eine neue Regel wird definiert, indem mit Hilfe des Menüs der Name der Fuzzy-Menge bestimmt wird, der in dem gewünschten Tabellenfeld erscheinen soll. Die Regelbasis wird dann um die Regel erweitert, die dem so erstellten Tabelleneintrag entspricht.

4.6 Definition und Start der Anwendung

Um NEFCON-I in die Lage zu versetzen, ein Anwendungsprogramm zu starten und mit diesem korrekt zu kommunizieren, sind bestimmte Angaben nötig. Es muß erstens bekannt sein, wie das Programm heißt. Zweitens muß die Reihenfolge festgelegt werden, in der die Meß- und Stellwerte ausgegeben bzw. eingelesen werden. Die entsprechenden Angaben werden in einem Eingabefenster gemacht, das im Hauptfenster durch den Button $\boxed{\text{definieren}}$ aufgerufen wird. Es ist in Abb. 4.2 zu sehen.

Das Eingabefenster enthält ein Eingabefeld für den Namen des Programms. In Abb. 4.2 ist als Beispiel der Name `pendel` eingetragen. Falls sich das Programm nicht in dem Verzeichnis befindet, von dem aus NEFCON-I gestartet wurde, ist der relative Pfadname mit anzugeben.

Rechts und links vom Eingabefeld befindet sich je eine Gruppe von Toggle-Buttons. Die auf der linken Seite geben, von oben nach unten ausgewertet, die Reihenfolge an, in der das Anwendungsprogramm die Stellwerte einliest. Auf der rechten Seite wird auf die gleiche Weise die Reihenfolge definiert, in der die Meßwerte ausgegeben werden. Die Stellwerte werden dabei durch die Namen der Ausgabevariablen repräsentiert und die Meßwerte entsprechend durch die der Eingabevariablen.

Das Eingabefenster wird, genau wie die Editoren für linguistische Variablen und Regeln, über das Menü $\boxed{\text{fertig}}$ verlassen. Vor dem Verlassen des Editors durch $\boxed{\text{beenden}}$, werden alle Angaben auf Vollständigkeit und Korrektheit überprüft. Wird ein Fehler gefunden, erscheint eine Fehlermeldung im Ausgabefeld und das

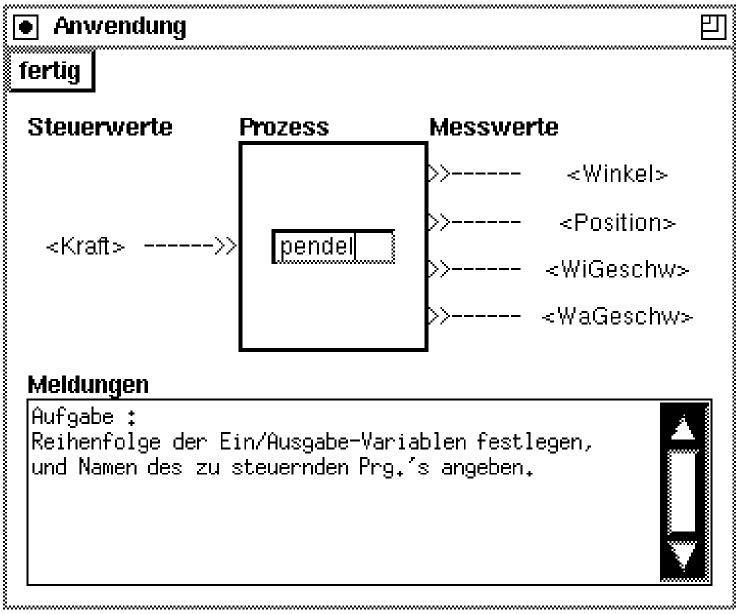


Abbildung 4.5: Fenster zur Definition der Anwendung

Fenster wird nicht geschloßen. Mögliche Gründe sind das Fehlen eines Programmnamens oder oder doppeltes Auftreten eines Variablennamens bei der Angabe der Reihenfolgen.

Ein korrekt definiertes Anwendungsprogramm kann im NEFCON-I-Hauptfenster durch den Button starten gestartet werden.

4.7 Die Regler-Oberfläche

Von der Regler-Oberfläche aus kann der Neuronale Fuzzy-Regler mit verschiedenen Optionen gestartet werden. Die Oberfläche wird im NEFCON-I-Hauptfenster durch den Button Oberflaeche aufgerufen und ist in Abb. 4.6 zu sehen.

Die Regler-Oberfläche kann grob in zwei Teile aufgeteilt werden. Der Bereich für die eigentliche Bedienung des Reglers befindet sich in der linken unteren Hälfte des Fensters. Das Feld rechts daneben dient, wie üblich, der Ausgabe diverser Meldungen.

Der obere Teil des Fensters enthält den sogenannten *grafischen Protolkollbereich*. In diesem Bereich können die Steuer-Variablen, die Steuer-Regeln und die Systemwerte dargestellt werden. Dabei werden die Veränderungen der einzelnen Komponenten während des Regelungsvorgangs sichtbar gemacht. Der nächste Unterabschnitt beschreibt die Darstellung der einzelnen Daten und verschiedene Möglichkeiten, die Darstellungsform zu beeinflußen.

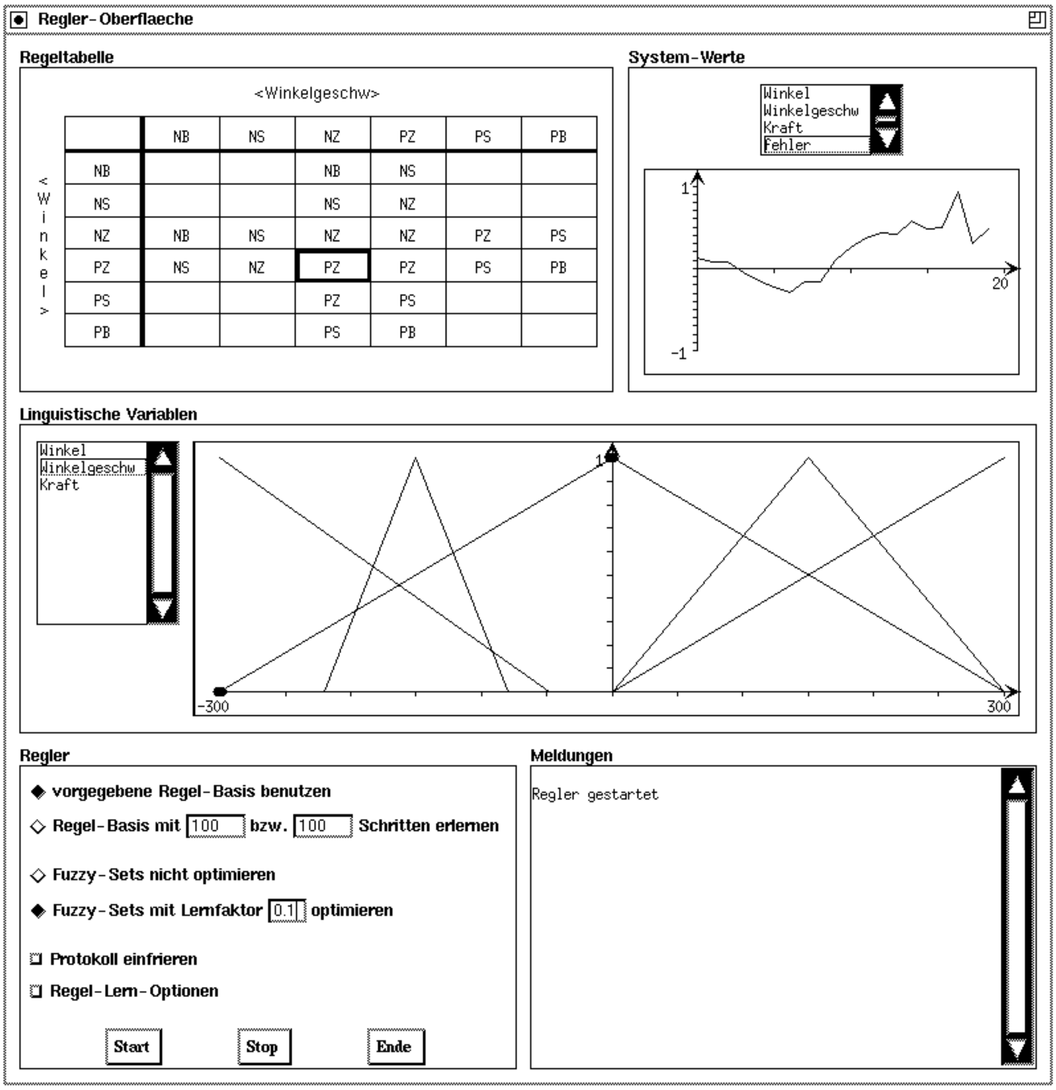


Abbildung 4.6: Die Regler-Oberfläche

4.7.1 Das Protokoll

Während des Regelungsvorgangs können verschiedene Ausgaben sowohl auf den Bildschirm als auch in eine Datei gemacht werden. Diese Ausgaben sollen den Regelungsverlauf und die Vorgänge innerhalb des Reglers veranschaulichen. Welche Daten dabei im grafischen Protokollbereich erscheinen, kann zum Teil vorher ebenso festgelegt werden wie die Art ihrer Darstellung. Auch das Anlegen einer Protokolldatei ist optional und muß vom Benutzer explizit gefordert werden.

Die entsprechenden Angaben werden durch die Definition eines Protokolls gemacht. Bevor näher darauf eingegangen wird, werden die Darstellungsmöglichkeiten des grafischen Protolkollbereiches erläutert. Die Darstellung bezieht sich auf:

Die Regelbasis : Sie kann entweder als Tafel oder als Tabelle ausgegeben werden (siehe Abschnitt 4.5). Dabei wird während des Regelungsvorgangs jeweils die Regel mit dem größten „Treffergrad“ markiert. Es ist aber auch möglich, ganz auf die Darstellung der Regeln zu verzichten.

Die Fuzzy-Mengen : Alle Fuzzy-Mengen genau einer Variablen werden in einem Koordinatensystem dargestellt. Die Auswahl der darzustellenden Variablen erfolgt über ein Auswahlfeld. Die Veränderungen der Fuzzy-Mengen durch den Lernalgorithmus lassen sich während des Regelungsvorgangs direkt beobachten. Außerdem wird die Fuzzy-Menge, die zu der Regel mit dem größten „Treffergrad“ gehört, markiert. Auf Wunsch kann die gesamte Darstellung jedoch auch entfallen.

Die Systemwerte : Unter Systemwerte werden die Meß- und Stellwerte sowie der Systemfehler verstanden. Sie können entweder grafisch oder numerisch ausgegeben werden. Auch hier kann auf die Darstellung verzichtet werden. Bei der grafischen Darstellung kann, wie bei den Variablen, nur ein Wert zur Zeit ausgegeben werden. Dieser wird auch hier über ein Auswahlfeld bestimmt.

Für die Definition des Protokolls existiert ein Eingabefenster, das durch den Button Protokoll im NEFCON-I-Hauptfenster aufgerufen wird. Es ist in Abb. 4.7 zu sehen. Die gewünschte Einstellung für den grafischen Protokollbereich wird über die verschiedenen Auswahl-Buttons festgelegt. Zusätzlich kann angegeben werden, ob eine Protokolldatei erzeugt werden soll. Die beiden Buttons Ende und Abbruch schließen das Fenster. Bei Abbruch der Protokolldefinition werden eventuelle Änderungen nicht übernommen.

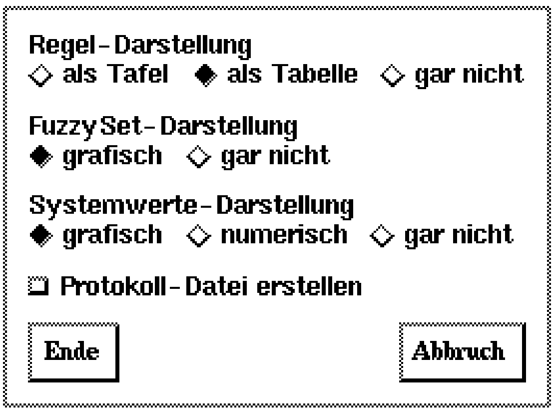


Abbildung 4.7: Das Protokollfenster

Die in Abb. 4.7 gezeigte Einstellung entspricht den Vorgabewerten, das heißt, diese Einstellung für das Protokoll wird benutzt, wenn nicht explizit eine andere definiert wird. In diesem Fall wird also keine Protokolldatei erzeugt, und der grafische Protokollbereich der Regler-Oberfläche erscheint so, wie es in Abb. 4.6 zu sehen ist.

Auf den Inhalt der Protokolldatei kann kein Einfluß genommen werden. Er ist von der grafischen Darstellung unabhängig. Der Name der Datei kann als Option beim Programmstart übergeben werden. Die entsprechende Option lautet:

```
-p="DATEI"
```

Als Vorgabewert wird „`protokoll.nfc`“ für den Namen benutzt. Die Datei beinhaltet im wesentlichen den Regelungsverlauf, also die gesamten Systemwerte und die Daten der Zugehörigkeitsfunktionen zu Beginn und am Ende des Regelungsvorgangs.

4.7.2 Starten des Reglers

Durch die beiden Buttons $\boxed{\text{Start}}$ und $\boxed{\text{Stop}}$ wird der Regler gestartet und gestoppt. Für den Start des Reglers können verschiedene Optionen angegeben werden.

Optionen für den Start

Der Regler kann entweder mit vorgegebenen Steuer-Regeln starten oder versuchen, selbst geeignete Kontrollregeln zu erlernen. Die entsprechende Option wird durch die beiden oberen Auswahl-Buttons im Bedienfeld für den Regler festgelegt. Für den Fall, daß eine Regelbasis erlernt werden soll, kann die Länge der ersten beiden Lernphasen (siehe Abschnitt 3.2.2) über die Anzahl der Lernschritte vorgegeben werden. Die entsprechenden Eingabefelder befinden sich neben dem Auswahl-Button. Ein Lernschritt entspricht dabei einem Regelungs-Zyklus. Die abschließende dritte Lernphase, also das Optimieren der Fuzzy-Mengen, ist bezüglich der Länge nicht begrenzt.

Das Erlernen einer Regelbasis kann noch durch weitere Angaben beeinflußt werden. Im nächsten Unterabschnitt wird näher auf die verschiedenen Möglichkeiten eingegangen.

Als weitere Startoption kann dem Regler mitgeteilt werden, ob die Fuzzy-Mengen der Steuer-Variablen während des Regelungsvorgangs optimiert werden sollen. Diese Option wird ebenfalls durch zwei Auswahl-Buttons repräsentiert. Der Lernfaktor für die Optimierung ist variabel und kann über ein entsprechendes Eingabefeld angegeben werden.

Die Wahl der Optionen in Abb. 4.6 entspricht den Vorgabewerten. Das heißt, es werden die vorgegebenen Steuer-Regeln benutzt und die Steuer-Variablen werden mit dem Lernfaktor 0.1 optimiert.

Die einzelnen Optionen werden nur beim Starten des Reglers ausgewertet. Änderungen, die während des Regelungsvorgangs gemacht werden, wirken sich also erst nach einem erneuten Start des Reglers aus. Eine Ausnahme bildet dabei der Lernfaktor, seine Änderung wird auch nach dem Start noch berücksichtigt.

Die Optionen für das Erlernen von Regeln

Mit dem Button Regel-Lern-Optionen wird ein Eingabefenster erzeugt, in dem die Optionen für das Erlernen der Steuer-Regeln gesetzt werden können. Dieses Fenster ist in Abb. 4.8 dargestellt.

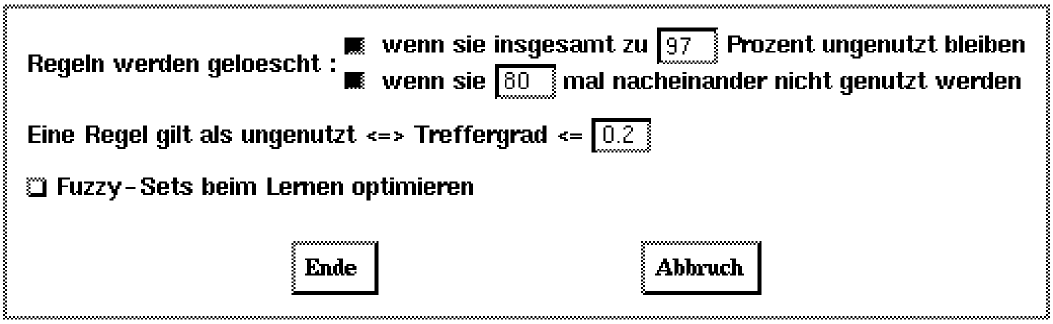


Abbildung 4.8: Die Optionen für das Erlernen von Regeln

Die Optionen, die hier gesetzt werden, beziehen sich ausschließlich auf die erste Lernphase beim Erlernen der Regeln. In dieser Phase geht es darum, unsinnige Regeln und Regeln, die nicht oder nur sehr selten benutzt werden, zu löschen.

Beim Festlegen der Optionen in diesem Fenster wird definiert, wann eine Regel aufgrund ihrer Nutzungsdauer gelöscht werden soll. Es gibt zwei Möglichkeiten, die Nutzungsdauer zu betrachten. Einerseits kann ein bestimmter Prozentsatz festgelegt werden, der angibt, zu wieviel Prozent eine Regel mindestens benutzt werden muß, um nicht gelöscht zu werden. Andererseits kann man das Löschen davon abhängig machen, wieviel Lernschritte hintereinander eine Regel unbenutzt bleiben darf. Darüber hinaus kann definiert werden, ab welchem Treffergrad eine Regel als benutzt angesehen wird.

Die Angabe des Prozentsatzes und der Anzahl der Lernschritte für das Löschen von Regeln erfolgt in den beiden oberen Eingabefeldern. Über die Auswahl-Buttons kann ebenfalls bestimmt werden, welche der beiden Methoden benutzt werden soll. Es ist natürlich auch möglich, beide gleichzeitig oder gar keine zu benutzen. Das Eingabefeld darunter legt den minimalen Treffergrad fest.

Zusätzlich kann als weitere Option angegeben werden, ob die Fuzzy-Mengen auch während der zweiten Lernphase schon optimiert werden sollen. Dies wird über den unteren Auswahl-Button festgelegt.

Die Einstellung in Abb. 4.8 ist also so zu interpretieren, daß Regeln gelöscht werden, die zu 97 Prozent nicht benutzt werden, wobei eine Regel genau dann als benutzt gilt, wenn ihr Treffergrad größer ist als 0.2. Die Fuzzy-Mengen werden während der zweiten Lernphase noch nicht optimiert.

Durch die Buttons Ende und Abbruch wird das Fenster wieder geschloßen. Genau wie im Fenster zur Definition des Protokolls, werden die Eingaben nicht berücksichtigt, wenn Abbruch betätigt wird.

Der Regelungsvorgang

Nach dem Start des Reglers wartet dieser auf das Einlesen der ersten Meßwerte. Die Initiative liegt in diesem Moment also beim Anwendungsprogramm. Nach jedem vollständigen Satz von Meßwerten, also einem für jede Eingabevariable, antwortet der Regler mit einem Satz von Stellwerten. Aus diesem Austausch von Daten entwickelt sich so der eigentliche Regelungsvorgang.

Der Button Protokoll einfrieren stoppt die Bildschirmausgaben im grafischen Protokollbereich. Da diese einen Großteil der Rechnerleistung verbrauchen, erhöht sich damit die Geschwindigkeit, in der der Regler die Steuerwerte berechnet, um ein Vielfaches. Um die Bildschirmausgaben wieder zu starten, wird dieser Button erneut betätigt. Eventuelle Ausgaben in eine Protokolldatei sind durch das Einfrieren nicht betroffen.

Beim Stoppen des Regelungsvorgangs durch den Button Stop oder Ende werden die vom Regler optimierten Fuzzy-Mengen als neue Steuer-Variablen in die Wissensbasis übernommen. Wenn der Regler während des laufenden Regelungsvorgangs durch Start erneut gestartet wird, gehen diese Änderungen verloren.

Beim Erlernen einer Regelbasis werden evtl. in der Wissensbasis vorhandene Steuer-Regeln vor dem Start des Reglers gelöscht. Die neu erlernten Regeln werden nach Abschluß der zweiten Lernphase in die Wissensbasis aufgenommen.

Der Button Ende stoppt den Regelungsvorgang und schließt die Regler-Oberfläche.

4.8 Dateiarbeit

Folgende Teile der Wissenbasis können in Dateien abgelegt bzw. aus diesen gelesen werden :

- Die Steuer-Variablen
- Die Fehler-Variablen
- Die Steuer-Regeln
- Die Fehler-Regeln

Es existieren zwei Dateiformate, einmal für linguistische Variablen und einmal für Regeln. Die Dateien, die die Variablen enthalten, haben die Endung „`lv`" und die für die Regeln „`rg`". Die Dateiformate sind in Anhang B dargestellt.

Das Laden und Speichern der Dateien erfolgt über die beiden Menüs $\boxed{\text{laden}}$ und $\boxed{\text{speichern}}$ im NEFCON-I-Hauptfenster. Nach Auswahl eines Menüpunktes erscheint das in Abb. 4.9 dargestellte Fenster.

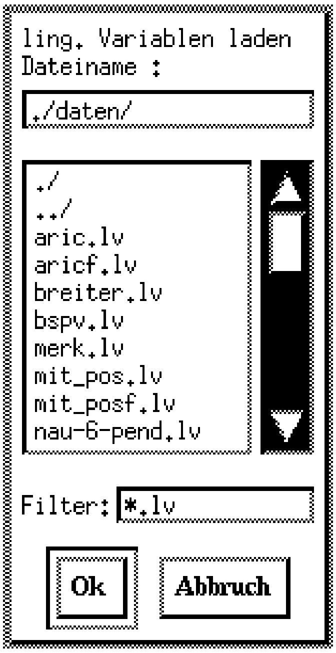


Abbildung 4.9: Das Dateiauswahlfenster

Abhängig vom ausgewählten Menüpunkt, werden entweder alle Dateien mit der Endung „`lv`" für linguistische Variablen oder mit der Endung „`rg`" für Regeln dargestellt. Mit Hilfe der Maus oder des entsprechenden Eingabefeldes wird dann der Name der Datei bestimmt, aus der gelesen oder in die geschrieben wird. Bei der Eingabe im Eingabefeld ist es nicht nötig, die Endung des Dateinamens mit anzugeben.

Als Option kann beim Start von NEFCON-I ein Verzeichnis angegeben werden, in dem nach entsprechenden Dateien gesucht wird und in das die neu erzeugten Dateien abgelegt werden. Die Option lautet:

```
-dir=VERZEICHNIS
```

Der Vorgabewert für dieses Arbeitsverzeichnis ist „`/daten`".

4.9 Ein Beispiel

In diesem Abschnitt wird als Beispiel eine komplette Wissensbasis vorgestellt, die zur Regelung des in Kapitel 6 beschriebenen inversen Pendels herangezogen werden kann. Außerdem wird das Regelungsverhalten vor und nach dem Trainieren der Fuzzy-Mengen verglichen. Der letzte Unterabschnitt zeigt schließlich eine von NEFCON-I erlernte Basis von Kontrollregeln.

4.9.1 Die Wissensbasis

In diesem Beispiel werden für die Regelung nur der Winkel und die Winkelgeschwindigkeit als Meßgrößen verwendet. Als Namen für die entsprechenden Eingabevariablen wurden *Winkel* und *Winkelgeschw* gewählt. Die Ausgabevariable hat den Namen *Kraft*. In Abb. 4.10 sind die Fuzzy-Mengen der Steuer-Variablen und in Abb. 4.11 die der Fehler-Variablen zu sehen.

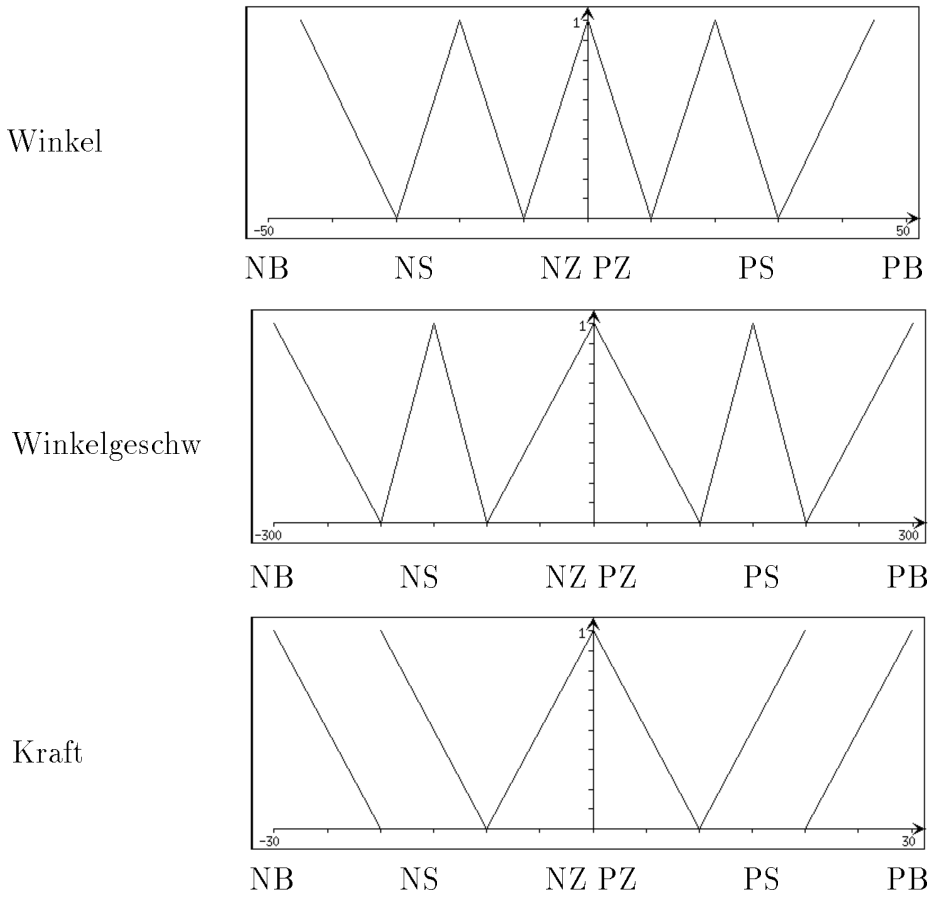


Abbildung 4.10: Steuer-Variablen

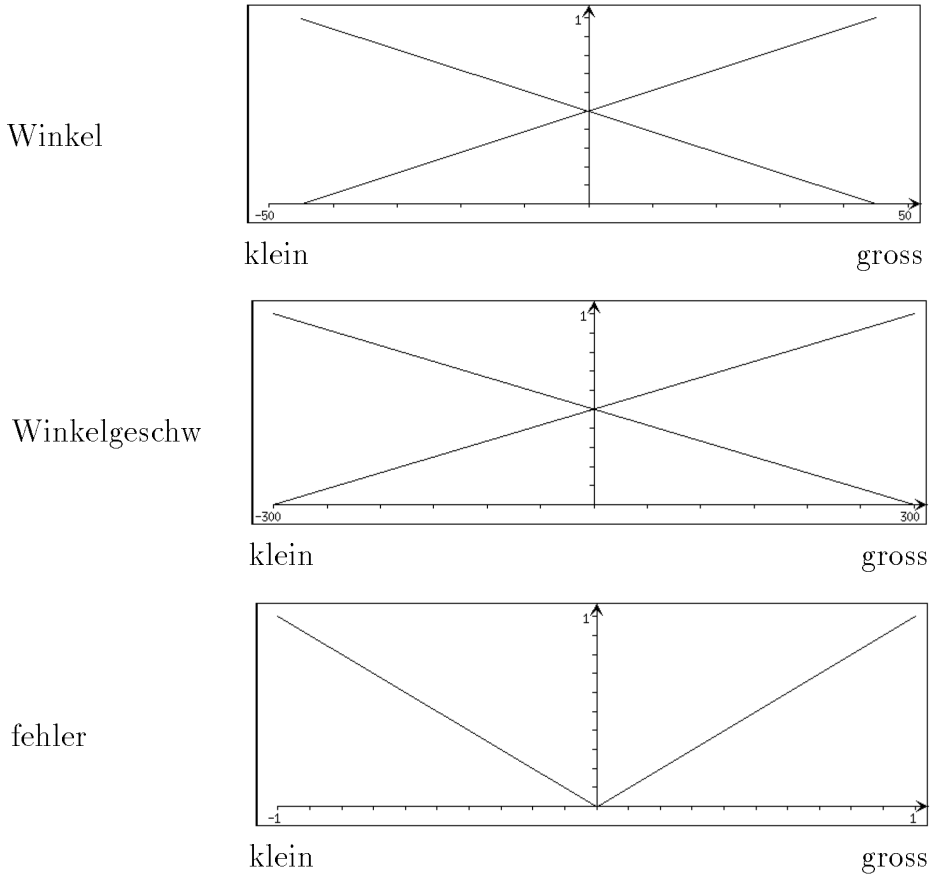


Abbildung 4.11: Fehler-Variablen

Die Steuer-Regeln und die Fehler-Regeln sind in den Abbildungen Abb. 4.12 und Abb. 4.13 dargestellt. Der Fehler wird hier auf sehr einfache Art und Weise berechnet. Für ein System mit nur zwei Eingabevariablen ist dies jedoch ausreichend. Bei einer größeren Anzahl ist eine feinere Partitionierung der einzelnen Fehler-Variablen und eine größere Anzahl von Fehler-Regeln nötig.

In diesem Beispiel werden die kompensatorischen Situationen (siehe Abschnitt 3.2.2) nicht explizit durch Fehler-Regeln modelliert. Sie werden jedoch indirekt dadurch berücksichtigt, daß gleichzeitig mehrere Regeln aktiv sein können und sich dadurch kompensieren. So kann beispielsweise aus einem positivem Winkel und einer negativen Winkelgeschwindigkeit ein Fehler von Null resultieren.

Allgemein ist zu sagen, daß die Definition des Fehlers sehr sorgfältig geschehen sollte, weil von ihm letztlich der Erfolg des Lernvorgangs abhängt. Dies gilt sowohl für das Trainieren der Fuzzy-Mengen als auch für das Erlernen von Steuer-Regeln.

<Winkel>

<Winkelgeschw>	NB	NS	NZ	PZ	PS	PB
NB			NB	NS		
NS			NS	NZ		
NZ	NB	NS	NZ	PZ	PZ	PS
PZ	NS	NZ	NZ	PZ	PS	PB
PS			PZ	PS		
PB			PS	PB		

Abbildung 4.12: Steuer-Regeln

```
IF Winkelgeschw = gross THEN fehler = gross
IF Winkelgeschw = klein THEN fehler = klein
IF Winkel = gross THEN fehler = gross
IF Winkel = klein THEN fehler = klein
```

Abbildung 4.13: Fehler-Regeln

4.9.2 Das Trainieren der Fuzzy-Mengen

Wenn die hier vorgestellte Wissensbasis benutzt wird, ohne die Fuzzy-Mengen zu trainieren, liefert der Regler die in Abb. 4.14 dargestellten Stellwerte. In der x-Achse ist in dieser Abbildung der Winkel und in der y-Achse die Winkelgeschwindigkeit eingetragen. Die z-Achse stellt die Kraft dar, die der Regler für die einzelnen Situationen berechnet. Man erkennt, daß sich die Kraft für Situationen, die sich kaum unterscheiden, teilweise sprunghaft ändert.

Diese Unregelmäßigkeiten spiegeln sich auch im Regelungsverhalten wieder. Mit dieser Wissensbasis, bzw. mit diesen Steuer-Variablen, ist der Regler nicht in der Lage, das Pendel zum Stillstand zu bringen.

Werden die Fuzzy-Mengen dagegen während des Regelungsvorgangs trainiert, zeigt sich ein deutlich besseres Verhalten. Die in Abb. 4.15 gezeigte Funktion beschreibt das Verhalten nach ca. 500 Zyklen. Man erkennt, daß die Funktion deutlich geglättet ist. Außerdem ist hier, im Gegensatz zu Abb. 4.14, für alle Situationen eine Kraft definiert. Die trainierten Fuzzy-Mengen unterscheiden sich von den ursprünglichen vor allem dadurch, daß sie erheblich „breiter“ sind.

Es zeigt sich, daß der Regler mit diesen Steuer-Variablen in der Lage ist, das Pendel um einen Winkel von 0^0 auszubalancieren.

Während des Lernvorgangs können nur die Fuzzy-Mengen verändert werden, die sich an der Berechnung der Stellwerte beteiligen. Es ist daher darauf zu achten, daß möglichst alle Situationen, in denen sich das zu regelnde System befinden kann, mindestens einmal erreicht werden. Bei der Regelung des Pendels kann dies beispielsweise durch Vorgabe verschiedener Winkelgeschwindigkeiten geschehen (siehe Abschnitt 6.1).

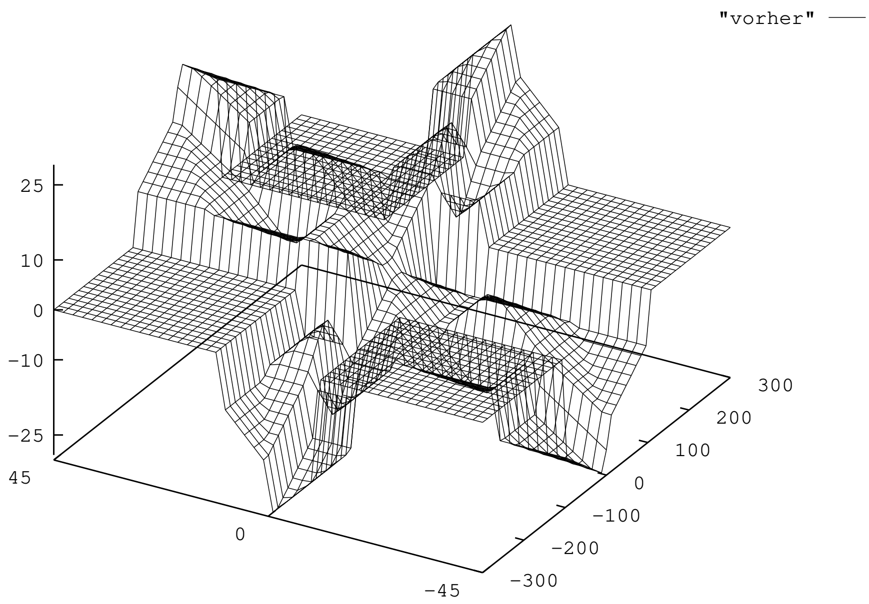


Abbildung 4.14: Vor dem Lernen

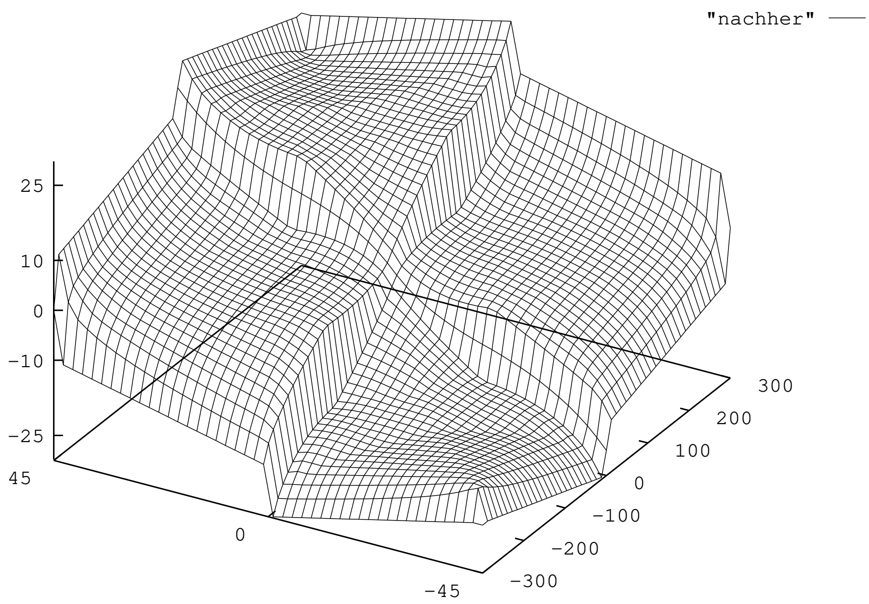


Abbildung 4.15: Nach dem Lernen

4.9.3 Das Erlernen von Steuer-Regeln

Abb. 4.16 zeigt ein Beispiel für eine Regelbasis, die vom Neuronalen Fuzzy-Regler erlernt wurde. Der Lernvorgang wurden dabei mit den nicht trainierten Steuer-Variablen aus Abb. 4.10 gestartet. Die Länge der ersten Lernphase betrug 300 die der zweiten 1000 Zyklen. Die Lern-Optionen wurden so gewählt, daß alle Regeln gelöscht wurden, deren Treffergrad während der ersten Lernphase nie größer war als 0. Auch hier wurde versucht, durch zufällige Startwerte für den Winkel (siehe Abschnitt 6.1) möglichst alle Situationen, in denen sich das Pendel befinden kann, zu erreichen.

<Winkel>

<Winkelgeschw>

	NB	NS	NZ	PZ	PS	PB
NB						
NS		NB	NB	NB	NB	
NZ	NB	NB	NZ	NZ	PS	
PZ	NB	NS	PZ	PZ	PB	PB
PS			PZ	PB	PB	
PB						

Abbildung 4.16: Steuer-Regeln

Diese Regelbasis ist geeignet, das Pendel zum Stillstand zu bringen und auch größere Winkel auszubalancieren. Wie sich gezeigt hat, sind nicht alle Versuch erfolgreich, eine Regelbasis zu erlernen. Ein Grund liegt darin, daß der Zufall eine gewisse Rolle spielt. Sowohl die Berechnung der Stellwerte (siehe Abschnitt 3.2.2) als auch die Zustände des zu regelnden Systems (siehe oben) sind nicht deterministisch. Um das Zufallselement möglichst auszuschalten, sollten daher die Längen der beiden Lernphasen genügend groß gewählt werden.

Zum anderen ist die Wahl der Lern-Optionen sehr wichtig. Die oben beschriebenen Optionen müssen nicht für jede Wissensbasis geeignet sein.

Kapitel 5

Die Implementierung

Die Implementierung des Programmsystems NEFCON-I kann in mehrere Teile gegliedert werden. Den Kern bilden dabei der Neuronale Fuzzy-Regler und die Wissensbasis, die unabhängig von der grafischen Benutzeroberfläche definiert wurden. Die Oberfläche selbst basiert auf den InterViews-Grafikklassen. Diese werden später in diesem Kapitel noch näher vorgestellt. Als Programmiersprache wurde C++ verwendet.

Abb. 5.1 zeigt alle für die Implementierung von NEFCON-I definierten Module. Sie können, entsprechend den Programmteilen, zu Gruppen zusammengefaßt werden. Diese sind hierachisch geordnet, das heißt, die Module aus den oberen Ebenen bauen auf die der unteren Ebenen auf. Folgende vier Gruppen lassen sich unterscheiden:

Allgemeine Module: Diese Module bilden die unterste Ebene. Sie enthalten Klassen und Funktionen, die allgemeine Verwendung finden. Dies sind zum Beispiel Funktionen, die den Umgang mit Zeichenketten oder die Berechnung von Zufallszahlen vereinfachen. Außerdem sind hier alle Texte definiert, die in NEFCON-I benötigt werden. In der aktuellen Version sind sie in den Sprachen Deutsch und Englisch vorhanden.

Definition des Reglers: Hier werden linguistische Variablen und Regeln, die Wissensbasis und der Regler definiert.

Allgemeine Grafikklassen: In diesen Modulen werden allgemeine Grafikelemente definiert, also Elemente, die an verschiedenen Stellen in der Benutzeroberfläche verwendet werden. Beispiele bilden das Ausgabefeld oder das Koordinatensystem.

Die Benutzeroberfläche: Die einzelnen Grafikelemente werden hier zu den unterschiedlichen Editoren und Eingabefenstern zusammengesetzt. Jedes Fenster, das in NEFCON-I auftaucht, wird durch genau eines dieser Module erzeugt.

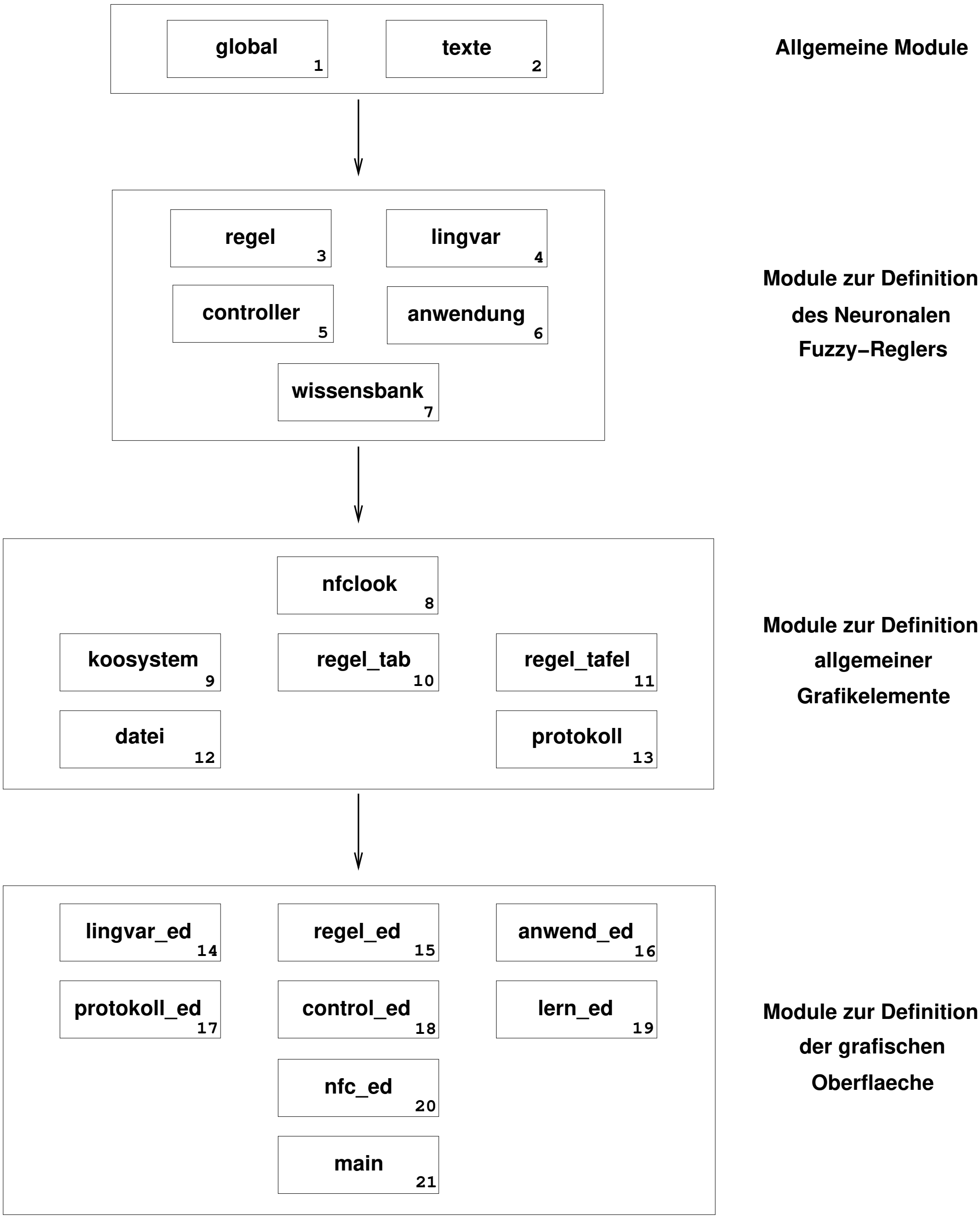


Abbildung 5.1: Modul-Hierachie

Die folgenden Unterabschnitte beschreiben die wichtigsten Klassen und Klassenhierachien soweit, daß dem Leser ein grundlegendes Verständnis der internen Datenstrukturen und Abläufe ermöglicht wird. Grobe Kenntnisse der Programmiersprache C++ werden dabei vorausgesetzt. Den Anfang macht die Implementierung des Reglers. Anschließend werden die zur Erzeugung der Benutzeroberfläche definierten Grafikklassen näher erläutert.

5.1 Wissensbasis und Neuronaler Fuzzy-Regler

Der Regler ist so implementiert worden, daß die jeweiligen Klassen auch unabhängig von der grafischen Oberfläche benutzt werden können. Zusammen mit der Wissensbasis bildet der Regler jedoch die Grundlage für die Benutzeroberfläche. Der Aufbau der meisten Grafikelemente ist beispielsweise abhängig von der Anzahl der definierten Variablen und Fuzzy-Mengen. Daher ist eine Schnittstelle nötig, über die diese Grafikelemente auf die statischen Informationen der Wissensbasis zugreiffen können.

Umgekehrt fallen vor allem während des Regelungsvorgangs beim Regler viele Informationen an, die grafisch darzustellen sind. Für diese dynamischen Daten muß eine Schnittstelle geschaffen werden, über die der Regler die Grafikelemente ansprechen kann.

Eine weitere Schnittstelle ist nötig, um die Kommunikation zwischen Regler und Anwendungsprogramm zu koordinieren.

Die folgenden Unterabschnitte beschreiben zunächst die Klassen zur Implementierung von Wissensbasis und Regler. Anschließend wird näher auf die einzelnen Schnittstellen eingegangen. Aufgrund der elementaren Bedeutung, werden die in diesem Abschnitt vorgestellten Klassen ausführlicher erläutert, als dies in späteren Abschnitten bei der Beschreibung von Grafikelementen der Fall ist.

5.1.1 Die Wissensbasis

Der Regler und viele der Grafikelemente benötigen als Parameter Instanzen der Klassen `LingVarSatz` und `RegelBasis`. Durch sie wird eine Menge von linguistischen Variablen bzw. Regeln repräsentiert. Die Klassendeklaration der Klasse `LingVarSatz` ist in Abb. 5.2 zu sehen.

In Abb. 5.2 wird, wie auch in den folgenden Beispielen, nur der Teil der Klasse dargestellt, der als `public` deklariert ist. Diese Darstellung wird hier auch als *Schnittstelle* einer Klasse bezeichnet.

Die Elementfunktionen `add_lingvar` bzw. `del_lingvar` nehmen neue linguistische Variablen in eine Instanz der Klasse `LingVarSatz` auf bzw. löschen schon vorhandene. Die Variablen werden in der Reihenfolge, in der sie aufgenommen werden, abgespeichert und können dann entweder über diese Nummer oder über ihren Namen angesprochen werden. Verschiedene Versionen der Elementfunktion `hol_lingvar` liefern als Rückgabewert eine der abgespeicherten Variablen. Die Adressierung der Variablen ist dabei auch durch eine Kombination von Variablentyp und Nummer möglich. So kann beispielsweise auf die n-te Eingabevariable zugegriffen werden.

Die restlichen Elementfunktionen liefern Informationen über Anzahl und Reihenfolge der abgepeicherten Variablen.

```
class LingVarSatz
{
  public : LingVarSatz() {};
           LingVarSatz(const LingVarSatz& lingvar_satz);
           ~LingVarSatz();

           LingVar* hol_lingvar(int nr) const;
           LingVar* hol_lingvar(LingVarTyp typ, int nr) const;
           LingVar* hol_lingvar(String name) const;
           void del_lingvar(int nr);
           void del_lingvar(String name);
           void add_lingvar(LingVar* lingvar);

           int var_nr(LingVarTyp, String name) const;
           int var_nr(String name) const;
           int anz_lingvars() const;
           int anz_lingvars(LingVarTyp) const;
           void tausche(int nr1, int nr2);
           LingVarSatz& operator=(const LingVarSatz&);
};
```

Abbildung 5.2: Schnittstelle der Klasse LingVarSatz

Eine Instanz der Klasse `RegelBasis`, deren Schnittstelle in Abb. 5.3 zu sehen ist, repräsentiert eine Regelbasis. Die linguistischen Regeln werden dabei durch die Klasse `Regel` modelliert. Die Elementfunktionen `add_regel` und `del_regel` fügen neue Regeln zur Regelbasis hinzu bzw. entfernen sie wieder. Ähnlich wie bei den linguistischen Variablen, werden die Regeln nach der Reihenfolge, in der sie in die Regelbasis aufgenommen wurden, nummeriert. Über diese Nummer kann dann mit Hilfe der Funktion `hol_regel` auf sie zugegriffen werden.

Als weitere Elelementfunktion steht `anzahl_regeln` zur Verfügung, die die Anzahl aller Regeln ermittelt. Die Funktion `such` stellt fest, ob sich eine bestimmte Regel in der Regelbasis befindet.

Die Variablen und Fuzzy-Mengen, die in Prämisse oder Konklusion einer linguistischen Regel vorkommen, werden innerhalb der Klasse `Regel` durch ihre Namen dargestellt. Dadurch wird ein gewisser Grad an Unabhänigkeit zwischen den Klassen `RegelBasis` und `LingVarSatz` erreicht. Es kann also beispielsweise die gleiche Regelbasis für verschiedene Sätze von linguistischen Variablen verwendet werden. Voraussetzung ist, daß beide Instanzen die gleichen Namen benutzen.

```
class RegelBasis
{
  public : RegelBasis() {};
           RegelBasis(const RegelBasis& regel_basis);
           ~RegelBasis();

           Regel* hol_regel(int nr) const;
           boolean del_regel(Regel* regel);
           boolean del_regel(int nr);
           void add_regel(Regel* regel);

           int anzahl_regeln() const;
           boolean such(Regel*, int& nr);

           RegelBasis& operator=(const RegelBasis&);
};
```

Abbildung 5.3: Schnittstelle der Klasse RegelBasis

5.1.2 Der Regler

Da die Berechnung des Fehlers, ebenso wie die der Steuerwerte, auf linguistischen Variablen und Regeln beruht, wird diese von einem Fuzzy-Regler durchgeführt. Es liegt also nahe, die Struktur des Neuronalen Fuzzy-Reglers auf die eines einfachen Fuzzy-Reglers zurückzuführen. Dieser übernimmt dann die Fehlerberechnung und bildet gleichzeitig den Grundstein für die Implementierung des Neuronalen Fuzzy-Reglers. In Abb. 5.4 ist eine entsprechende Strukturierung als Petri-Netz dargestellt.

Der Neuronale Fuzzy-Regler enthält also einerseits einen Fuzzy-Regler zur Fehlerberechnung. Andererseits erweitert er den Fuzzy-Regler zur Berechnung der Steuerwerte dahingehend, daß er anhand eines Fehlers dessen linguistische Variablen optimiert. Wie in Abb. 5.4 zu sehen, fließen auch andere Informationen in die Optimierung mit ein. Diese geben Aufschluß über die Beteiligung einzelner Regeln und Fuzzy-Mengen am Zustandekommen des vorhergehenden Steuerwertes.

Die in Abb. 5.4 gezeigte Strukturierung spiegelt sich auch in der Programmstruktur bzw. in den Klassendefinitionen zur Implementierung des Neuronalen Fuzzy-Reglers wieder. Die Klasse `FuzController` modelliert einen einfachen Fuzzy-Regler. Die Elementfunktionen zur Berechnung der Ausgabevariablen werden dabei virtuell deklariert, so daß man sie in abgeleiteten Klassen neu definieren oder erweitern kann. Eine solche Ableitung bildet die Klasse `NeuroFuzController`, sie implementiert den Neuronalen Fuzzy-Regler. In ihr ist zusätzlich eine Instanz der Klasse

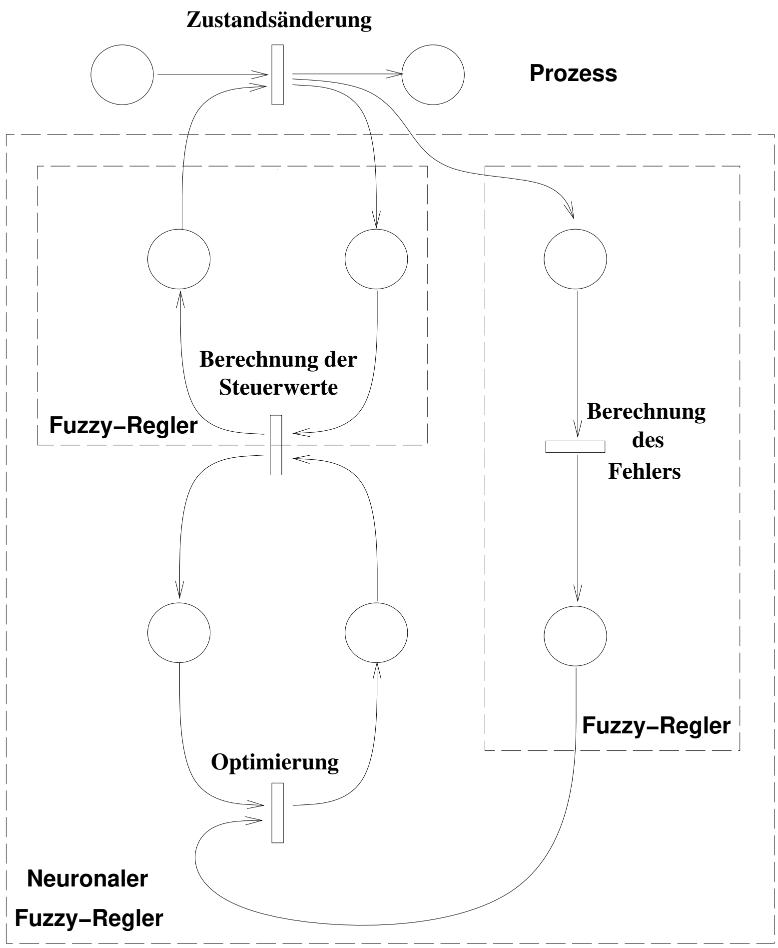


Abbildung 5.4: Der Neuronale Fuzzy-Regler als Petri-Netz

`FuzController` zur Fehlerberechnung enthalten.

Die Schnittstelle von `FuzController` ist in Abb. 5.5 zu sehen. Der Konstruktor benötigt als Parameter eine Variable vom Typ `LingVarSatz*` und eine vom Typ `RegelBasis*`. Die linguistischen Variablen und Regeln bilden für den Regler die Grundlage für die Berechnung der Ausgabewerte. Beide werden als Kopie innerhalb des Reglers gespeichert. Wenn der Parameter für die Regeln den Wert `nil` besitzt, wird eine Regelbasis erzeugt, die alle Regeln enthält, die sich aus den gegebenen Variablen bilden lassen. Dieser Fall tritt, wie später noch beschrieben wird, beim Erlernen einer Regelbasis auf.

Für interne Berechnungen und für die Darstellung von Meß- und Steuerwerten werden Variablen vom Typ `FuzzyTyp` verwendet. In der jetzigen Version ist dieser als `float` definiert.

```
class FuzController
{
  public : FuzController(const LingVarSatz*, const RegelBasis*);
           virtual ~FuzController();

           virtual FuzzyTyp* berechne_steuerwerte(
                               FuzzyTyp* messwerte
                             );

           virtual void reset() {};
           virtual void lern_faktor(float) {};
           virtual void protokoll(Protokoll*) {};

           LingVarSatz* lingvarsatz();

  protected : virtual FuzzyTyp _t_norm(FuzzyTyp, FuzzyTyp);
              virtual FuzzyTyp _defuz(
                                 FuzzyTyp y,
                                 FuzzySet* fuzzy_set,
                                 FuzzyTyp& gewicht
                               );
};
```

Abbildung 5.5: Schnittstelle der Klasse FuzController

Die Elementfunktion `berechne_steuerwerte` berechnet aus gegebenen Meßwerten die entsprechenden Stellwerte. Meß- und Stellwerte werden dabei in Arrays abgelegt, deren Größe sich nach der Anzahl der Ein- und Ausgabevariablen richtet. Zur Berechnung werden die Funktionen `_t_norm` und `_defuz` herangezogen. Als T-Norm wird hier die Minimum-Funktion verwendet. Die Funktion `_defuz` defuzifiziert eine gegebene Fuzzy-Menge und liefert als Ergebnis einen scharfen Wert. Bei monotonen Zugehörigkeitsfunktionen μ wird das Inverse $\mu^{-1}(y)$ berechnet, wobei `y` den Treffergrad der Regel angibt. Bei nicht monotonen Zugehörigkeitsfunktionen wird das Mean-Of-Maxima-Verfahren angewandt. Wie in Abschnitt 3.2.1 beschrieben, wird also die Ausgabe jeder Regel einzeln defuzifiziert. Für jede Ausgabe wird außerdem ein Gewicht berechnet, das in dieser Version dem Treffergrad der Regel entspricht.

Die Elementfunktionen `reset`, `lern_faktor` und `protokoll` sind leer. Da sie *virtuell* deklariert sind, bieten sie abgeleiteten Klassen die Möglichkeit für entsprechende Erweiterungen. Einen Zeiger auf die in der Klasse `FuzController` enthaltene Kopie der linguistischen Variablen liefert die Funktion `lingvarsatz`.

```
class NeuroFuzController : public FuzController
{
  public : NeuroFuzController(
             const LingVarSatz* sv, const LingVarSatz* fv,
             const RegelBasis* sr,  const RegelBasis* fr
           );
           virtual ~NeuroFuzController();

           virtual FuzzyTyp* berechne_steuerwerte(FuzzyTyp* ein);

           virtual void reset();
           virtual void lern_faktor(float);
           virtual void protokoll(Protokoll*);

  protected : void _optimieren(FuzzyTyp fehler);
              FuzController* _fehler_ctrl;
};

class RgLrnNeuroFuzCtrl : public NeuroFuzController
{
  public : RgLrnNeuroFuzCtrl(
             const LingVarSatz* sv, const LingVarSatz* fv,
             const RegelBasis* fr,
             RegelBasis* steuer_regeln,
             RgLernStruktur* lern_strukt
           );
           virtual ~RgLrnNeuroFuzCtrl();

           virtual FuzzyTyp* berechne_steuerwerte(FuzzyTyp* ein);
           virtual void protokoll(Protokoll*);
};
```

Abbildung 5.6: Abgeleitete Klassen von FuzController

Die Schnittstelle der Klasse `NeuroFuzController` ist in Abb. 5.6 zu sehen. Der Konstruktor benötigt als Parameter die Steuer-Variablen und die Steuer-Regeln zum Initialisieren der Elternklasse. Außerdem benötigt er die Fehler-Variablen und die Fehler-Regeln, mit denen er `_fehler_ctrl`, den Regler zur Fehlerberechnung, erzeugt.

Zur Berechnung der Steuerwerte wird auch hier die Elementfunktion `berechne-`

`_steuerwerte` aufgerufen. Sie führt im wesentlichen drei Schritte durch:

1. Aus den Meßwerten wird mit Hilfe von `_fehler_ctrl` der Fehler berechnet.
2. Der Fehler wird an die Elementfunktion `_optimieren` übergeben. Die Informationen über das Zustandekommen des vorhergehenden Steuerwertes sind lokal vorhanden und werden zusammen mit dem Fehler zum Optimieren der Fuzzy-Mengen herangezogen.
3. Durch `FuzController::berechne_steuerwerte`, also die entsprechende Elementfunktion der Elternklasse, werden die neuen Steuerwerte berechnet.

Die Elementfunktion `reset` muß immer dann aufgerufen werden, wenn während des Regelungsvorgangs das zu regelnde System erneut startet. Beim Beispiel der Regelung eines Pendels ist dies immer dann der Fall, wenn das Pendel umgekippt ist. Das Protokoll zur Prozeßkommunikation zwischen NEFCON-I und Anwendungsprogramm ist so ausgelegt, daß das Pendel die entsprechende Information an den Regler übergeben kann. Für den Neuronalen Fuzzy-Regler ist diese Information deshalb wichtig, weil er die nachfolgenden Meßwerte und den daraus resultierenden Fehler dann nicht zum Optimieren der Fuzzy-Mengen heranziehen darf. Der neue Startzustand wurde in diesem Fall nämlich nicht durch die vorhergehenden Steuerwerte beeinflußt.

Über die Elementfunktionen `lern_faktor` und `protokoll` kann der Regler mit einem bestimmten Lernfaktor und einem Protokoll (siehe Unterabschnitt 5.1.4) initialisiert werden. Ohne diese Initialisierung hat der Lernfaktor den Wert 0, und es werden keine Protokollausgaben gemacht.

Von der Klasse `NeuroFuzController` ist `RgLrnNeuroFuzCtrl` abgeleitet. Dieser Name steht für *Regel-lernender Neuronaler Fuzzy-Controller*. Wie der Name schon sagt, ist diese Klasse zusätzlich in der Lage, auch eine Regelbasis zu erlernen. Ihre Schnittstelle ist ebenfalls in Abb. 5.6 zu sehen.

Dem Konstruktor wird anstelle der Steuer-Regeln eine Variable übergeben, die die erlernte Regelbasis aufnimmt, nachdem die ersten beiden Lernphasen beendet sind. Außerdem wird ein Parameter vom Typ `RgLernStruktur*` erwartet, der die verschiedenen Optionen zum Lernen der Regeln enthält.

Die Elementfunktion `berechne_steuerwerte` wird so erweitert, daß sie während der beiden ersten Lernphasen die Regelbasis trainiert. Nach deren Abschluß verhält sie sich genau wie `NeuroFuzController::berechne_steuerwerte`. Die Länge der beiden ersten Lernphasen wird durch die Optionen festgelegt, die dem Konstruktor übergeben werden. In Abhängigkeit der aktuellen Lernphase führt `berechne_steuerwerte` folgende Schritte durch:

- Lernphase 1:
    1. Durch `FuzController::berechne_steuerwerte` werden aus den Meßwerten die Stellwerte berechnet.
    2. `_fehler_ctrl::berechne_steuerwerte` berechnet den aktuellen Systemfehler.
    3. Alle Regeln mit unsinnigem Stellwert werden gelöscht. Zusätzlich werden verschiedene Zähler aktualisiert, die über die Beteiligung der verschiedenen Regeln am Zustandekommen der Stellwerte Aufschluß geben.
    4. Nach dem letzten Lernschritt dieser Phase werden die Zähler ausgewertet. Das heißt, die Regeln, die sich nicht im gewünschten Maß am Regelungsvorgang beteiligt haben, werden gelöscht.
- Lernphase 2:
    1. `_fehler_ctrl::berechne_steuerwerte` berechnet den aktuellen Systemfehler.
    2. Falls die entsprechende Option gesetzt ist, werden die Fuzzy-Mengen trainiert.
    3. Für jede Regel existiert eine Variable, die die Summe aller Systemfehler enthält, an denen die Regel beteiligt war (siehe Abschnitt 3.2.2). Diese Variablen werden hier aktualisiert.
    4. Die Stellwerte werden durch eine abgewandelte Version der Elementfunktion `FuzController::berechne_steuerwerte` berechnet. Und zwar beteiligen sich die einzelnen Regeln dabei nur mit der Wahrscheinlichkeit $\frac{2}{n}$ an der Berechnung der Stellwerte, wobei n die Anzahl der der möglichen Konklusionen angibt.
    5. Nach dem letzten Lernschritt dieser Phase werden die Variablen ausgewertet, in denen der Systemfehler summiert wurde. Das heißt, von allen Regeln mit gleicher Prämisse wird nur die behalten, deren Variable den kleinsten Wert aufweist.
- Lernphase 3:
    1. Aufruf der Funktion `NeuroFuzController::berechne_steuerwerte`.

In Anhang D sind die Quelltexte der verschiedenen Versionen der Elementfunktion `berechne_steuerwerte` angegeben.

5.1.3 Die Schnittstelle zwischen Regler und Anwendung

Die Definition eines Anwendungsprogramms und die gesamte Kommunikation mit diesem wird durch die Klasse `Anwendung` übernommen. Aus der Sicht des Reglers bildet diese Klasse den zu regelnden Prozeß. Ihre Schnittstelle ist in Abb. 5.7 zu sehen.

```
class Anwendung
{
  public : Anwendung(const LingVarSatz*);
           ~Anwendung();
           Anwendung(const Anwendung&);

           void starten(Ausgabe*);
           FuzzyTyp* messwerte();
           void steuerwerte(FuzzyTyp*);

           void link(Action* ok, Action* nook);
           void unlink();

           void name(String);
           String name() const;

           void var(LingVarTyp, String name);
           String var(LingVarTyp, int nr);
           void loesch_var(LingVarTyp);
           int anzahl(LingVarTyp) const;

           Anwendung& operator=(const Anwendung&);
};
```

Abbildung 5.7: Deklaration der Klasse Anwendung

Gestartet wird das Anwendungsprogramm durch die Elementfunktion `starten`. Die Kommunikation mit diesem Programm erfolgt über die beiden Elementfunktionen `messwerte` und `steuerwerte`. Durch diese Funktionen können Meßwerte gelesen oder Steuerwerte abgeschickt werden. Die Verbindung zwischen einer Instanz der Klasse `Anwendung` und dem eigentlichen Anwendungsprogramm wird dabei durch einen gemeinsamen Ein- und Ausgabekanal hergestellt.

Der Elementfunktion `link` können zwei Zeiger auf Funktionen übergeben werden. Die erste dieser Funktionen wird von der Klasse `Anwendung` immer dann aufgerufen, wenn neue Meßwerte vom Anwendungsprogramm gelesen wurden, die zweite, wenn es zu Fehlern beim Einlesen derselben kam oder das Anwendungsprogramm terminierte. Durch diese Funktionen wird eine Verbindung zum Regler hergestellt und dafür gesorgt, daß die Kommunikation während des Regelungsvorgangs nicht abreißt. Da sie ereignisgesteuert ist, wird das Warten auf neue Meßwerte vermieden, und es ist möglich, auch während des Regelungsvorgangs Eingaben in der grafischen Benutzeroberfläche zu bearbeiten. Die Verbindung wird durch `unlink` wieder gelöst.

Die übrigen Elementfunktionen dienen dazu, den Namen des Anwendungsprogramms und die Reihenfolge der Ein- und Ausgabe von Meß- und Steuerwerten festzulegen. Ihre Bedeutung im einzelnen ist hier nicht weiter wichtig, und eine nähere Erläuterung entfällt deshalb.

5.1.4 Die Schnittstelle vom Regler zur grafischen Ausgabe

Um die Änderungen, die sich während des Regelungsvorgangs innerhalb der Wissensbasis ergeben, und um die Aktivitäten des Reglers grafisch zu veranschaulichen, wird der grafische Protokollbereich der Regler-Oberfläche ständig aktualisiert. Dieser beinhaltet beispielsweise Grafikelemente zur Darstellung der Systemwerte oder der Regel mit dem momentan höchsten Treffergrad.

Am einfachsten wäre es, wenn die Steuerung des Protokolls dabei vom Regler ausgeht, da hier alle zu protokollierenden Daten und Informationen über eventuelle Änderungen zusammenfließen. Das Problem dabei ist, daß der Regler in der Modulhierachie weiter unten steht als die Grafikklassen des Protokolls und deshalb nicht direkt auf sie zugreifen kann. Um dieses Problem zu umgehen, wurde die virtuelle Basisklasse `Protokoll`, dargestellt in Abb. 5.8, definiert. Sie kann als Elternklasse für Protokolle dienen, die ihre Ausgaben dann beispielsweise auf den Bildschirm oder in eine Datei machen.

```
class Protokoll
{
  public : virtual void init(const LingVarSatz*) = 0;
           virtual void init(const RegelBasis*) = 0;
           virtual void aktua_lingvarsatz() = 0;
           virtual void zeig_regel(int nr) = 0;
           virtual void zeig_eingabe_var(int nr, FuzzyTyp wert)= 0;
           virtual void zeig_ausgabe_var(int nr, FuzzyTyp wert)= 0;
           virtual void zeig_fehler(FuzzyTyp wert) = 0;
           virtual void fehlschlag() = 0;
};
```

Abbildung 5.8: Deklaration der Klasse Protokoll

Die Klassen `NeuroFuzController` und `RgLrnNeuroFuzCtrl` werden durch die entsprechenden Elementfunktionen mit einer Instanz der Klasse `Protokoll` initialisiert.

Der Regler wickelt nun seine Protokollarbeit ab, indem er die entsprechenden Elementfunktionen des ihm übergebenen Protokolls aufruft. Das heißt, zu Beginn wird das Protokoll vom Regler über die Elementfunktion `init` mit linguistischen Variablen bzw. Regeln initialisiert. Die Funktion `aktua_lingvarsatz` wird aufgerufen, wenn im Rahmen des Lernvorgangs Änderungen an den Fuzzy-Mengen vorgenommen wurden. `zeig_regel` informiert das Protokoll nach jeder Berechnung von Stellwerten über die Regel mit dem größten Aktivierungsgrad. Die Systemwerte, also die Werte der Meß- und Stellwerte sowie der Systemfehler, werden dem Protokoll über die Funktionen `zeig_eingabe_var`, `zeig_ausgabe_var` und `zeig_fehler` mitgeteilt. `fehlschlag` wird nach jedem Fehlschlagen des Regelungsvorgangs aufgerufen.

In NEFCON-I wird der Regler mit einer Instanz der Klasse `NfcProtokoll` initialisiert (siehe Abschnitt 5.2.5).

5.1.5 Die Schnittstelle von der grafischen Ausgabe zur Wissensbasis

In der Klasse `Wissensbank` sind alle Informationen, die zur Regelung eines Anwendungsprogramms nötig sind, vorhanden. Sie enthält also:

- Je eine Instanz der Klasse `LingVarSatz` zur Modellierung der Steuer-Variablen und der Fehler-Variablen.
- Je eine Instanz der Klasse `RegelBasis` zur Modellierung der Steuer-Regeln und der Fehler-Regeln.
- Eine Instanz der Klasse `FuzController`, die den Neuronalen Fuzzy-Regler repräsentiert.
- Eine Instanz der Klasse `Anwendung` zur Darstellung des zu regelnden Systems bzw. des Anwendungsprogramms.

Da die Deklaration dieser Klasse relativ lang ist und nichts Neues mehr beinhaltet, soll ihre Darstellung hier entfallen. Die einzelnen Komponenten der Klasse `Wissensbank` können über die Elementfunktionen sowohl gelesen als auch initialisiert werden. So liest zum Beispiel der Regel-Editor bei seinem Aufruf die alten Regeln aus einer Instanz der Klasse `Wissensbank` und schreibt die neuen Regeln nach seiner Beendigung wieder dorthin zurück.

Diese Klasse, von der natürlich nur eine Instanz in NEFCON-I existiert, stellt also eine Art Plattform zur Verfügung. Alle Komponenten, die für eine beliebige Anwendung mit Hilfe der grafischen Benutzeroberfläche definiert wurden, werden hier abgelegt und von hier wieder ausgelesen. Die Klasse `Wissensbank` enthält außerdem Elementfunktionen, mit der sich die Konsistenz der einzelnen Komponenten untereinander überprüfen läßt.

5.2 Die Benutzeroberfläche

Die Benutzeroberfläche wurde mit Hilfe der InterViews-Klassenbibliothek implementiert. Sie setzt sich aus verschiedenen wiederverwertbaren Grafikelementen zusammen. Diese stammen zum Teil direkt aus InterViews oder sind Erweiterungen von bestehenden InterViews-Grafikelementen. Teilweise wurden aber auch neue Grafikklassen definiert, die sich nach den speziellen Bedürfnissen von NEFCON-I ausrichten. Beispiele bilden die Tabelle zur Regeldarstellung oder das Koordinatensystem zur Darstellung beliebiger Funktionen.

Im nächsten Unterabschnitt soll nun zunächst ein grober Überblick der InterViews-Klassenbibliothek gegeben werden, bevor näher auf die speziell für NEFCON-I definierten Grafikklassen eingegangen wird.

5.2.1 InterViews

InterViews stellt eine C++-Klassen-Bibliothek zur Verfügung, mit deren Hilfe grafische Benutzeroberflächen auf der Basis von X11 erzeugt werden können. Es wurde an der Stanford University entwickelt und setzt keine Kentnisse über das zugrundeliegende X11-System voraus. Im Gegensatz zu anderen Grafik-Bibliotheken, wie zum Beispiel dem *Athena Widget Set* oder *Motif*, wird das System vollständig durch die InterViews-Klassen gekapselt.

Durch die Verwendung der Programmiersprache C++ ist es möglich, Grafikkomponenten auf einer hohen Abstraktionsebene zu behandeln. Einfache Komponenten sind in InterViews bereits vordefiniert, und durch den Vererbungsmechanismus können auf einfache Weise neue definiert werden. Die Kombination von einzelnen Elementen zu größeren Einheiten erfolgt nach einem Modell, das dem, welches in TEX benutzt wird, ähnelt und sehr einfach zu handhaben ist.

Im folgenden werden die wichtigsten Gruppen von InterViews-Klassen kurz dargestellt. Eine eingehendere Beschreibung von InterViews wird in [Linton et al., 1992, Roseman, 1992, Mayer, 1992] gegeben.

Grundlegende Widgets

Die grundlegenden Grafikelemente (Widgets), die in fast allen Benutzeroberflächen vorkommen, sind in InterViews vordefiniert. Dazu gehören:

Buttons: Es werden verschiedene Arten von Buttons zur Verfügung gestellt, wie zum Beispiel `Push Buttons` oder `Radio Buttons`.

Labels: Diese dienen zur Darstellung von Texten innerhalb der Grafik.

Bevels: Sie geben beliebigen Grafikelementen durch Hinzufügen von Umrandungen oder Schattierungen ein dreidimensionales Aussehen.

Menus: Es existieren einfache Klassen, mit denen Menübalken mit beliebigen Untermenüs konstruiert werden können.

Adjusters: Mit Hilfe von Adjustern kann die Darstellung von bestimmten Grafikelementen beeinflußt werden. Zu den Adjustern zählen zum Beispiel die Scroll-Balken.

Diese Grafikelemente werden in der Klasse `WidgetKit` zusammengefaßt. Instanzen der einzelnen Klassen werden durch entsprechende Elementfunktionen zur Verfügung gestellt. Das Erscheinungsbild der Grafikelemente kann über X11-Ressourcen oder eine Option beim Aufruf beeinflußt werden (siehe Anhang A). Es kann sich beispielsweise an *Motif* oder *Open Look* orientieren.

Komposition von Widgets

Das Zusammenfügen der einzelnen Grafikelemete zu größeren Einheiten erfolgt ähnlich wie in TEX. Es existieren folgende Möglichkeiten, eine Gruppe von einzelnen Komponenten relativ zueinander anzuordnen:

- Von links nach rechts nebeneinander (`HBox`).
- Von oben nach unten übereinander (`VBox`).
- Alle auf die gleiche Position (`Overlay`).
- Alle auf die gleiche Position, wobei nur eine zur Zeit sichtbar ist (`Deck`).

Für jede dieser Möglichkeiten werden in der Klasse `LayoutKit` Elementfunktionen zur Verfügung gestellt. Diese benötigen als Parameter die einzelnen Komponenten und liefern als Rückgabewert ein Grafikelemet, das diese in der gewünschten Anordnung enthält. Außerdem existieren Grafikelemente, die dazu benutzt werden können, horizontal oder vertikal Platz zwischen den einzelnen Komponenten zu lassen (`HGlue` oder `VGlue`). Auch auf diese kann man über die Klasse `LayoutKit` zugreiffen. Ebenso wie noch auf verschiedene andere Funktionen, mit denen sich die Anordnug einzelner Grafikelemente beeinflussen lässt. Ihre Beschreibung im einzelnen soll hier jedoch entfallen.

Komplexere Widgets

Über die Klasse `DialogKit` erhält man komplexere Grafikelemente. In der aktuellen InterViews-Version sind jedoch nur zwei realisiert. Eines davon ist die Klasse `FieldEditor` zur Eingabe von Zeichenketten. Das andere ist `FileChooser`, das ein Fenster zur Auswahl von Dateinamen erzeugt.

Definition eigener Widgets

Alle Grafikelemente sind von der abstrakten Basisklasse `Glyph` abgeleitet. Aussehen und Verhalten eines jeden Grafikelementes wird hauptsächlich durch folgende Elementfunktionen festgelegt:

Glyph::request(...): Durch den Aufruf dieser Funktion wird ein `Glyph` dazu aufgefordert, seine gewünschte Größe anzugeben.

Glyph::allocate(...): Diese Funktion informiert die Klasse `Glyph` über den ihm zugeteilten Bildschirmbereich. Es muß hier den Bereich angeben, den es davon benutzen will.

Glyph::draw(...): In dieser Funktion wird das `Glyph` in dem ihm zugeteilten Bereich gezeichnet.

Um ein eigenes Grafikelement zu definieren, bildet man also eine abgeleitete Klasse von `Glyph` und sollte mindestens diese drei Elementfunktionen neu definieren. Eine eingehendere Beschreibung dieser und noch weiterer Elementfunktionen von `Glyph` kann [Linton et al., 1992] entnommen werden.

5.2.2 Grundlegende Grafikelemente von NEFCON-I

Es existieren eine Reihe von einfachen selbstdefinierten Grafikelementen, die in NEFCON-I sehr häufig benutzt werden. In der Regel stellen sie Erweiterungen von schon vorhandenen InterViews-Elementen dar. Sie werden in dem Modul `nfclook` implementiert. Im folgenden werden die Klassen, die diese Grafikelemente modellieren, kurz vorgestellt.

EingabeFeld: Dieses Eingabefeld basiert auf der InterViews-Klasse `FieldEditor`. Es dient zur Eingabe und Darstellung von Zahlen oder Zeichenketten.

AusgabeFeld: In diesem Feld können mehrzeilige Texte dargestellt werden. Es baut auf die InterViews-Klasse `TBScrollBox` auf.

NfcAusgabe: Diese Klasse dient der Ausgabe von Benutzermeldungen in einem festen Format. Die Art der Meldung wird dabei durch den Typ `AusgabeTyp` festgelegt, der die Werte `Fehler`, `Warnung`, `Meldung` oder `Info` annehmen kann. Für die Ausgabe wird dabei eine Instanz der Klasse `AusgabeFeld` benutzt.

NfcStrBrowser: Aufbauend auf die InterViews-Klasse `FileBrowser`, wird eine Reihe von Zeichenketten dargestellt, von denen eine per Maus ausgewählt werden kann. Instanzen dieser Klasse wurden im vorangehenden Kapitel als Auswahlfeld bezeichnet.

AuswahlButton: Ähnlich wie in der Klasse `NfcStrBrowser`, kann mit Hilfe der Maus aus einer Anzahl von Texten einer selektiert werden. Hier wird jedoch immer nur ein Text zur Zeit dargestellt, der nach jedem Anklicken mit der Maus wechselt. Der jeweils dargestellte Text gilt als ausgewählt. Im vorangehenden Kapitel wurden Instanzen dieser Klasse als Toggle-Buttons bezeichnet.

NfcLabel: Diese Klasse wird zur Darstellung von Texten, vor allem in Tabellen, benutzt. Der Unterschied zur InterViews-Klasse `Label` besteht darin, daß sich die Art der Darstellung nach dem zur Verfügung stehenden Platz richtet. Das heißt, wenn der Text zu lang ist, wird nicht der gesamte Text dargestellt, und wenn er kürzer ist, wird er bei der Ausgabe in der Mitte zentriert.

NfcAction: Durch die InterViews-Klasse `Action` können einfache Callback-Funktionen definiert werden, um diese dann beispielsweise zur Konstruktion von Buttons oder Menüs einzusetzen. Der Nachteil dabei ist, daß, wenn man zum Beispiel eine Reihe von Menüpunkten mit der gleichen Callback-Funktion ausstattet, man in dieser Callback-Funktion nicht entscheiden kann, durch welchen Menüpunkt der Aufruf getätigt wurde. Um diesen Nachteil zu umgehen, ohne für jeden der Menüpunkte eine eigene Callback-Funktion schreiben zu müssen, wurde die Klasse `NfcAction` definiert. Sie baut auf die Klasse `Action` auf und weist einer globalen Variable vor dem Ausführen des Callbacks einen bestimmten Wert zu. Anhand dieses Wertes kann dann in der Callback-Funktion der Aufrufer identifiziert werden.

NfcDialogFenster: Diese Klasse erzeugt ein einfaches Fenster auf der Grundlage der InterViews-Klasse `TopLevelWindow`.

NfcKit: In dieser Klasse befinden sich Funktionen, die die Erzeugung oft verwendeter Grafikkonstruktionen vereinfachen und vereinheitlichen sollen (vgl. `LayoutKit` und `WidgetKit`).

5.2.3 Die Darstellung von Zugehörigkeitsfunktionen

Zur grafischen Darstellung von Fuzzy-Mengen und anderen stückweise linearen Funktionen wurde die Klasse `KooSystem` definiert. Sie zeichnet ein Koordinatenkreuz, dessen Abszisse und Ordinate beliebig skaliert werden können. Innerhalb dieses Koordinatenkreuzes lassen sich dann veschiedene Linien darstellen.

Da InterViews kein geeignetes Grafikelement zur Verfügung stellt, das es erlaubt, beliebig auf dem Bildschirm zu zeichnen, wurde als Grundlage bzw. Elternklasse für `KooSystem` die Klasse `KooKreuz` definiert. Diese wiederum ist eine Ableitung von `Glyph`. Sie zeichnet zwei Koordinatenachsen mit einer bestimmten Skalierung und bildet damit den Hintergrund zur Darstellung von Funktionen. Abgeleitet von `KooSystem` ist die Klasse `FuzzySetGrafik`, in der genau eine Instanz der Klasse `FuzzySet` grafisch dargestellt wird.

Die Klassenhierachie für die Implementierung des Koordinatensystems ist in Abb. 5.9 zu sehen. Diese Art der Darstellung wird im folgenden des öfterern benutzt und soll nun kurz erklärt werden.

Die Namen der Klassen werden in Kästchen eingetragen, wobei reine InterViews-Klassen durch kursive Schrift hervorgehoben werden. Bei den übrigen Klassen ist zusätzlich eine Nummer angegeben, die angibt, in welchem Modul sie definiert sind. Die Nummern beziehen sich dabei auf Abb. 5.1.

In den Abbildungen werden zwei Beziehungen zwischen verschiedenen Klassen unterschieden. Zum einen kann eine Klasse A Elternklasse einer Klasse B sein. Man sagt in diesem Fall auch, daß B von A abgeleitet ist. Diese Beziehung wird grafisch mit einer durchgezogenen Linie veranschaulicht. Die Elternklasse steht dabei immer oben. In Abb. 5.9 ist zum Beispiel `FuzzySetGrafik` von `KooSystem` abgeleitet. Die zweite Beziehung ist die, daß eine Instanz der Klasse A in einer Klasse B benutzt wird. Diese Tatsache wird grafisch mit einem „gestrichelten“ Pfeil von A nach B angedeutet. So wird eine Instanz der Klasse `FuzzySetGrafik` beispielsweise zur Realisierung des „grafischen Funktions-Editors“ `GrafFktEd` benutzt.

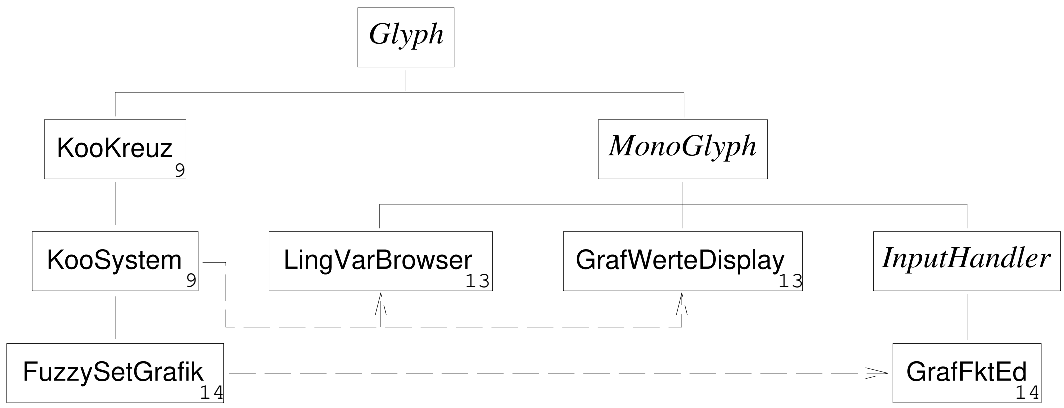


Abbildung 5.9: Klassenhierachie um die Klasse KooSystem

Wie in der Abbildung zu erkennen ist, werden Instanzen der Klasse `KooSystem` noch in zwei anderen Klassen benutzt. In der Klasse `LingVarBrowser` dient das Koordinatensystem der Darstellung aller Fuzzy-Mengen einer ausgewählten linguistischen Variablen. In `GrafWerteBrowser` wird es dazu benutzt, ein Reihe von numerischen Werten grafisch als Kurvenverlauf darzustellen. Beide sind Ableitungen der Klasse `MonoGlyph`, die in den meisten Fällen dazu benutzt wird, bestehende Grafikelemente, also `Glyphs`, zu erweitern. Die Instanzen von `LingVarBrowser` und `GrafWerteBrowser` werden im grafischen Protokollbereich verwendet.

Die Klasse `GrafFktEd` benutzt eine Instanz von `FuzzySetGrafik`, einer Ableitung von `KooSystem`, und bietet die Möglichkeit, Fuzzy-Mengen mit der Maus eingeben zu lassen. Das Koordinatensystem dient in diesem Fall als „Hintergrund“ für den `InputHandler`. Diese Klasse erlaubt die Abfrage von Mausposition und Tastendrücken. Die Klasse `GrafFktEd` wird zur Realisierung des Editors für linguistische Variablen benötigt.

Der Editor wird durch die Klasse `LingVarEd`, eine Ableitung von `NfcDialogFenster`, implementiert. `LingVarEd` enthält ein Ausgabefeld und eine Instanz der Klasse `FuzzySetAuswahl` und bringt beide in einem Fenster unter. Der eigentliche Editor wird dabei durch die Klasse `FuzzySetAuswahl` implementiert. In Abb. 5.10 sind die wichtigsten Klassen zur Implementierung des Editors dargestellt.

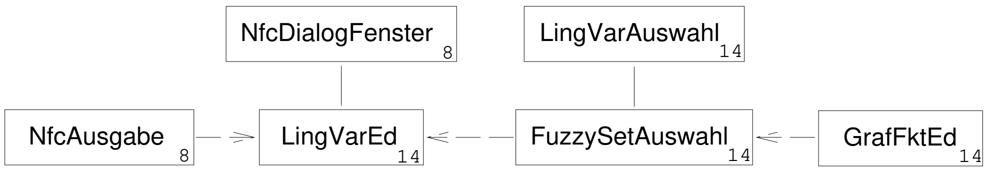


Abbildung 5.10: Klassen zur Implementierung des Variablen-Editors

Wie man in der Abbildung erkennt, ist `FuzzySetAuswahl` von der Klasse `LingVarAuswahl` abgeleitet. Letztere bildet die Grundlage des Editors und implementiert den Teil, der sich mit der Auswahl schon vorhandener oder der Eingabe neuer Variablen beschäftigt. Dies bezieht sich nur auf den Namen, den Typ und den Definitionsbereich der Variablen.

Die Klasse `FuzzySetAuswahl` bildet eine Erweiterung in der Hinsicht, daß zusätzlich Fuzzy-Mengen für die Variablen definiert werden können. Zur grafischen Eingabe derselben wird eine Instanz der Klasse `GrafFktEd` benutzt. Ansonsten setzt sich die Benutzeroberfläche, die von diesen beiden Klassen gebildet wird, aus einfachen Grafikelementen, wie Ein- und Ausgabefeldern, Toggle-Buttons oder Auswahlfeldern zusammen. In Abb. 5.10 sind diese Klassen nicht mehr einzeln aufgeführt.

5.2.4 Die Darstellung von Regeln

Die Darstellung von linguistischen Regeln ist, im Gegensatz zur Darstellung von Fuzzy-Mengen, wesentlich komplexer. Das liegt daran, daß sie in NEFCON-I sowohl in Tabellenform als auch in Form einer Tafel (siehe Abschnitt 4.5) dargestellt werden. Ein anderer Grund liegt darin, daß in den Fällen, in denen mehr als zwei Ein- und mehr als eine Ausgabevariable definiert wurden, durch die Regeltabelle nur eine ganz bestimmte „Sicht“ auf die Regeln geboten werden kann. Neben der eigentlichen Darstellung ist dann auch einiger Aufwand nötig, um zu entscheiden welche Daten dargestellt werden und auf welche Weise dies geschieht.

Die Klassen, die für die Regeldarstellung definiert wurden, können grob in zwei Gruppen aufgeteilt werden. Eine Gruppe implementiert die eigentlichen Grafikelemente, während sich die andere mit deren Verwaltung bzw. Steuerung beschäftigt. In Abb. 5.11 sind beide Gruppen mit ihren Abhängigkeiten dargestellt.

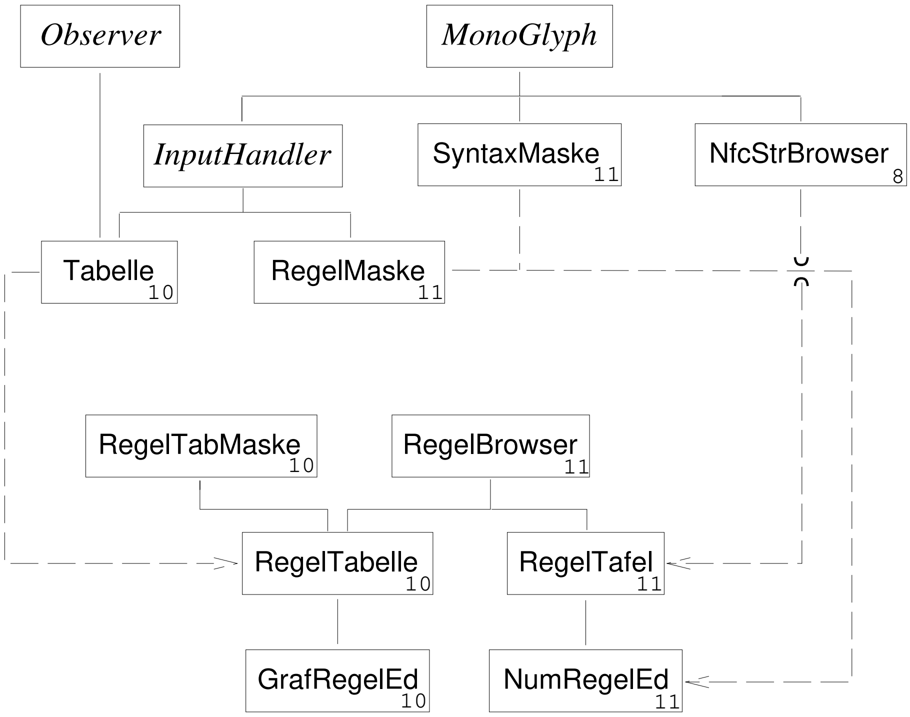


Abbildung 5.11: Klassenhierachie zur Regeldarstellung

Das grundlegende Grafikelement zur Darstellung von Regeln in Tabellenform ist die Klasse `Tabelle`. Sie definiert eine Tabelle mit einer gegebenen Anzahl von Zeilen und Spalten und bietet die Möglichkeit, Texte in den einzelnen Tabellenfeldern einzublenden. Außerdem wird jeweils das Tabellenfeld, in dem sich der Mauszeiger befindet, durch einen breiteren Rand markiert. Um die Position und evtl. Tastendrücke der Maus abfragen zu können, wurde die Klasse `Tabelle` von `InputHandler` abgeleitet. Als weitere Elternklasse fungiert die InterViews-Klasse `Observer`, die es ermöglicht, das Aussehen einzelner Grafikbestandteile, wie zum Beispiel das der Tabellenfelder, auch während der Laufzeit des Programms zu ändern. Eine Instanz von `Tabelle` wird in der Klasse `RegelTabelle` benutzt.

Die Klasse `NfcStrBrowser`, die schon in einem der vorangegangenen Abschnitte beschrieben wurde, wird für die Darstellung der Regeln in Form einer Regeltafel benötigt. Als Erweiterung der Klasse `RegelTafel` existiert die Klasse `NumRegelEd`, dies steht für „numerischer Regel-Editor“. Wie der Name schon sagt, können über diese Klasse auch Regeln eingegeben werden. Dazu werden die Grafikelemente `RegelMaske` und `SyntaxMaske` benutzt. Durch sie wird der in Abschnitt 4.5.1 beschriebene Eingabebereich für Regeln implementiert.

Es ist sinnvoll, für beide Darstellungsformen eine geeignete Basisklasse zu definieren, um diese in das grafische Protokoll einbinden zu können. Die gewählte Form zur Darstellung der Regeln bleibt damit für das Protokoll transparent, es kann dann wahlweise mit einer Instanz der `RegelTabelle` oder `RegelTafel` initialisiert werden.

Die Rolle einer solchen Basisklasse wird hier durch `RegelBrowser` übernommen.

Die Klasse `RegelTabMaske` ist eine weitere Basisklasse von `RegelTabelle`. Sie ist legt fest, welche Regeln in der Regeltabelle dargestellt werden. Dies ist jedoch nur für die Fälle interessant, in der mehr als zwei Ein- oder mehr als eine Ausgabevariable benutzt werden. Diese Klasse implementiert also im wesenlichen die in Abschnitt 4.5.2 beschriebene Menüleiste. Abgeleitet von `RegelTabelle` ist die Klasse `GrafRegelEd`, dieser Name steht für „grafischer Regel-Editor". Diese Klasse nutzt eine Option der Klasse `Tabelle` aus, die es ermöglicht, in einzelnen Tabellenfeldern ein Menü erscheinen zu lassen. Dieses wird dann zur Eingabe von Regeln verwendet (siehe Abschnitt 4.5.2).

Vollständigkeitshalber ist zu erwähnen, daß die einzelnen Menüs, die natürlich auch Grafikelemente sind und von den Klassen `RegelTabMaske` und `GrafRegelEd` benutzt werden, nicht in Abb. 5.11 zu sehen sind. Aus Gründen der Übersichtlichkeit werden reine InterViews-Grafikelemente, die in den einzelnen Klassen benutzt werden, in den Abbildungen weggelassen.

Ähnlich wie bei der Implementierung des Variablen-Editors, wird auch der Editor für linguistische Regeln durch eine Ableitung der Klasse `NfcDialogFenster`, in diesem Fall `RegelEd`, repräsentiert. Für die Eingabe und die Darstellung der Regeln wird je eine Instanz der Klassen `NumRegelEd` und `GrafRegelEd` benutzt. Die entsprechenden Klassen sind in Abb. 5.12 dargestellt.

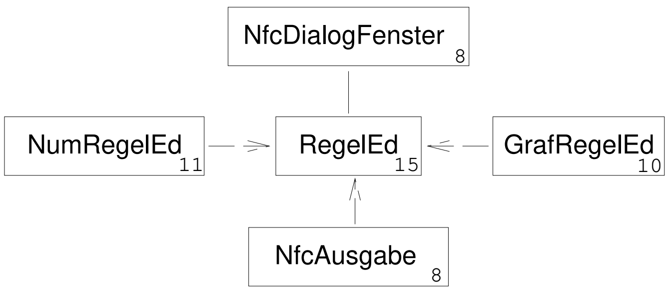


Abbildung 5.12: Klassen zur Implementierung des Regel-Editors

5.2.5 Die Repräsentation des Protokolls

Unter Protokoll wird hier die Darstellung derjenigen Daten verstanden, die den Regelungsvorgang dokumentieren. Dies kann sowohl grafisch auf den Bildschirm als auch in eine Datei geschehen. Welche Daten dabei im einzelnen betroffen sind, ist Abschnitt 4.7.1 zu entnehmen.

Zur grafischen Darstellung von linguistischen Variablen und Regeln werden die schon angesprochenen Klassen `LingVarBrowser` und `RegelBrowser` verwendet. Die Darstellung der Systemwerte erfolgt durch die Klasse `WerteBrowser`. Es existieren zwei

Ableitungen dieser Klasse, nämlich `GrafWerteBrowser` und `NumWerteBrowser`. Erstere benutzt zur Darstellung eine Instanz von `KooSystem` (siehe Abschnitt 5.2.3). In der zweiten Klasse werden die einzelnen Systemwerte numerisch durch Instanzen der Klasse `NfcLabel` dargestellt. Das Erzeugen einer Protokolldatei erfolgt durch die Klasse `DateiProtokoll`.

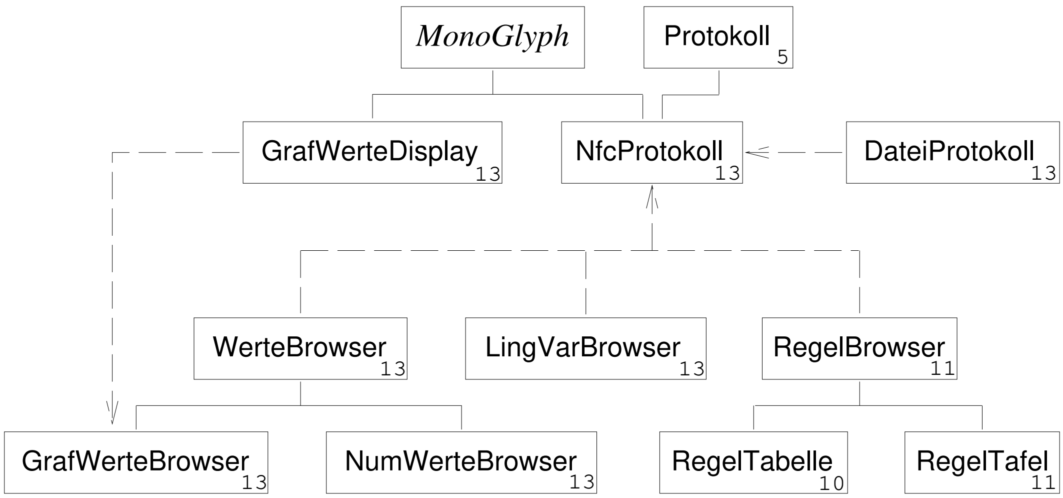


Abbildung 5.13: Klassen zur Darstellung des Protokolls

In Abb. 5.13 sind die einzelnen Klassen dargestellt. Wie man erkennt, wird das Protokoll durch `NfcProtokoll`, einer Ableitung der in Abschnitt 5.1.4 beschriebenen Klasse `Protokoll`, implementiert. Dazu werden Instanzen der vorhin erwähnten Klassen `WerteBrowser`, `LingVarBrowser`, `RegelBrowser` und `DateiProtokoll` benutzt. Dem Konstruktor von `NfcProtokoll` kann mitgeteilt werden, welche dieser vier Instanzen nun tatsächlich verwendet werden sollen und welche nicht, so daß das Aussehen des Protokolls in gewissen Grenzen vom außen beeinflußbar ist. Es kann außerdem vorgegeben werden, welche der beiden möglichen Unterklassen von `WerteBrowser` und `RegelBrowser` benutzt werden sollen. Die verschiedenen Möglichkeiten zum Aufbau des Protokolls werden durch den Typ `ProtokollStruktur` festgelegt.

5.2.6 Die Regler-Oberfläche

Die Regler-Oberfläche `ControllerEd` implementiert ein Fenster, in dem zum einen das grafische Protokoll dargestellt wird und das zum anderen die Benutzeroberfläche zur Bedienung des Reglers enthält. Zur Darstellung des Protokolls wird eine Instanz der Klasse `NfcProtokoll` benutzt, und die Benutzeroberfläche wird aus verschiedenen Buttons und Eingabefeldern zusammengesetzt. Die einzelnen Klassen werden in Abb. 5.14 gezeigt.

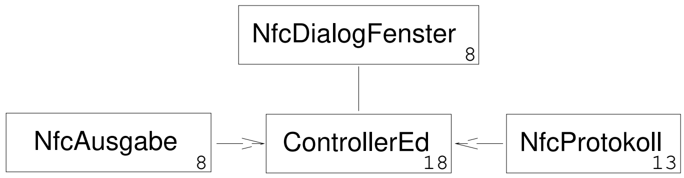


Abbildung 5.14: Klassen zur Darstellung der Regler-Oberfläche

Über die Regler-Oberfläche wird außerdem der Regler erzeugt und mit dem Anwendungsprogramm verbunden. Dazu wird in drei Schritten vorgegangen:

1. Eine Instanz des Reglers erzeugen
   Zuerst wird geprüft, ob alle Komponenten der Klasse `Wissensbank` untereinander konsistent sind. Falls dies der Fall ist, werden die Optionen, die in der Benutzeroberfläche angegeben wurden, untersucht. Abhängig davon wird dann der Regler erzeugt. Das heißt, wenn Regeln erlernt werden sollen, wird eine Instanz von `RgLrnNeuroFuzCtrl` verwendet. Im anderen Fall wird der Regler durch eine Instanz von `NeuroFuzController` gebildet. Dies gilt auch dann, wenn die Fuzzy-Mengen nicht optimiert werden sollen. Die Instanz des Reglers wird in der `Wissensbank` abgelegt.

2. Den Regler initialisieren
   Der Regler wird mit einem Protokoll und einem Lernfaktor initialisiert. Als Protokoll wird ein Zeiger auf die oben beschriebene Instanz der Klasse `NfcProtokoll` verwendet. Der Lernfaktor wird aus dem Eingabefeld in der Benutzeroberfläche gelesen. Falls die Fuzzy-Mengen nicht optimiert werden sollen, wird der Lernfaktor auf Null gesetzt.

3. Die Verbindung zum Anwendungsprogramm herstellen
   Über die Funktion `Anwendung::link()` (siehe Abschnitt 5.1.3) wird eine Verbindung zwischen den beiden Instanzen der Klassen `FuzController` und `Anwendung` hergestellt, die beide in der `Wissensbank` enthalten sind.

5.2.7 Sonstige Eingabefenster

Das Hauptfenster wird durch ein `ApplicationWindow` erzeugt, in dem eine Instanz der Klasse `NfcEditor` dargestellt wird. Diese Klasse organisiert sowohl den Aufruf der verschiedenen Editoren als auch die Bearbeitung der Menüpunkte. Sie enthält genau eine Instanz der Klasse `Wissensbank`, in der, wie in Abschnitt 5.1.5 beschrieben, alle zur Regelung eines Anwendungsprogramms relevanten Daten abgelegt werden.

Zur Organisation der Editoraufrufe gehört, daß diese beim Starten mit den entsprechenden Daten initialisiert werden. Nach dem Beenden eines Editors wird die Komponente der Wissensbasis, die im Editor bearbeitet wurde, aktualisiert.

In Abb. 5.15 ist die Klasse `NfcEd` und alle Grafikklassen, die von ihr benutzt werden, dargestellt. Dabei handelt es sich in erster Linie um die, in den vorangegangenen Abschnitten beschriebenen Editoren. Zum Lesen und Schreiben aus bzw. in Dateien wird eine Instanz der Klasse `DateiArbeit` benutzt.

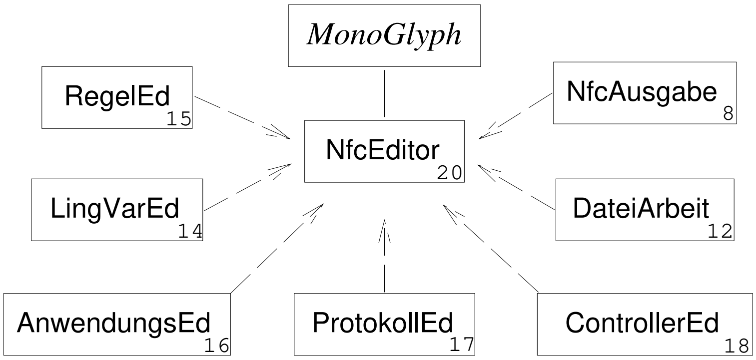


Abbildung 5.15: Klassen zur Darstellung des Hauptfensters

Es existieren noch weitere Eingabefenster, die aufgrund ihrer Einfachheit hier nur kurz erwähnt werden sollen:

AnwendungsEd: Dies ist eine Ableitung von `NfcDialogFenster`. Es bildet aus einem Eingabefeld und mehreren Toggle-Buttons eine Benutzeroberfläche zur Definition der Anwendung.

ProtokollEd: Hierbei handelt es sich um eine Ableitung von `Dialog` und damit um ein Popup-Fenster. Es besteht aus einigen Auswahl-Buttons, über die das Aussehen des Protokolls bestimmt werden kann.

LernEd: Dies ist, genau wie `ProtokollEd`, eine Ableitung von `Dialog`. Es besteht ebenfalls aus einigen einfachen Grafikelementen und dient dazu, die verschiedene Optionen für das Erlernen von Regeln festzulegen.

Kapitel 6

Das Pendel

Als Beispiel für ein dynamisches System wurde die Simulation eines inversen Pendels (Stabbalance-Problem) implementiert. Dieses Programm mit dem Namen `pendel` kann von NEFCON-I aus gestartet und geregelt werden. Mit seiner Hilfe kann das Lern- und Regelverhalten des Neuronalen Fuzzy-Reglers mit einer vorher definierten Wissensbasis näher untersucht werden. Der zu regelnde Prozeß wird dabei grafisch dargestellt, und sein Verhalten kann über die Benutzeroberfläche beeinflußt werden.

Beim Stabbalance-Problem wird versucht einen Stab, der auf einem Wagen befestigt ist, im Schwerefeld durch Bewegung des Wagens vertikal zu balancieren. Den schematischen Versuchsaufbau zeigt Abb. 6.1. Der Schaft des Stabes, an dessen Ende sich die punktförmige Masse m befindet, wird als masselos angenommen. Als Stellwert in diesem System dient die Kraft F, die den Wagen entlang einer horizontalen Achse bewegt. Der Winkel θ des Stabes zur vertikalen Achse, die Position x des Wagens auf der horizontalen Achse sowie die Winkelgeschwindigkeit $\dot{\theta}$ und die Wagengeschwindigkeit $\dot{x}$ können als Meßwerte verwendet werden.

Die Einheit für die Kraft ist Newton. Winkel und Winkelgeschwindigkeit messen sich in Grad bzw. Grad pro Sekunde. Entsprechend werden Position und Wagengeschwindigkeit in Meter bzw. Meter pro Sekunde angegeben.

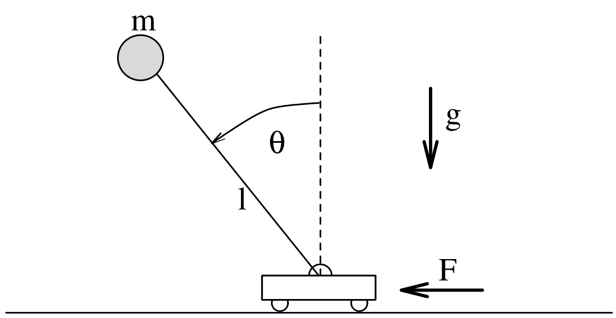


Abbildung 6.1: Das Stabbalance-Problem

Als Grundlage für die Simulation dienen die beiden folgenden Differentialgleichungen [Berenji, 1992]:

$$\ddot{\theta} = \frac{gsin\theta + cos\theta\,(-f - ml\dot{\theta}^2 sin\theta + \mu_c sgn(\dot{x}))/(m_c + m) - \mu_p\dot{\theta}/ml}{l(\frac{4}{3} - (mcos^2\theta)/m_c + m))}$$

$$\ddot{x} = \frac{f + ml\,(\dot{\theta}^2 sin\theta - \ddot{\theta}cos\theta) - \mu_c sgn(\dot{x})}{m_c + m}$$

Hierbei ist l die Länge des Stabes und m_c die Masse des Wagens. Durch μ_v bzw. μ_p werden die Reibungskoeffizienten des Wagens auf der Strecke bzw. des Pendels auf dem Wagen beschrieben.

Dieses Differentialgleichungssystem zweiter Ordnung wurde in [Foerster, 1993] mit dem Runge-Kutta-Verfahren vierter Ordnung gelöst. Die daraus resultierende Lösung wurde für die Implementierung der Pendelsimulation benutzt. Die einzelnen Konstanten sind dabei auf folgende Werte gesetzt worden:

$$\begin{aligned} m &= 1\,kg \\ m_c &= 1\,kg \\ l &= 0.5\,m \\ \mu_p &= 0.33 \\ \mu_c &= 0.33 \end{aligned}$$

Die nächsten Abschnitte beschäftigen sich mit Bedienung und Implementierung des Programms `pendel`.

6.1 Das Simulationsprogramm

Die Syntax für den Programmaufruf lautet:

`pendel` [-laenge **wert**]
[-e]
[-mWi]
[-mWiG]
[-mPo]
[-mWaG]

Die einzelnen Optionen können, genau wie bei NEFCON-I, auch über X11-Resourcen gesetzt werden. Beim Aufruf des Programms durch NEFCON-I, ist diese Methode zu wählen, da dort keine Angabe von Optionen möglich ist. Die einzelnen Optionen haben folgende Bedeutung:

- -laenge **wert**
  Durch **wert** wird die Länge der Strecke in Metern angegeben, auf der sich der Wagen bewegt. Bei einer Länge von 0 m wird die Strecke als unendlich angenommen. Die dazugehörige X11-Resource lautet `*LaengeStrecke` und ist vorgabemäßig mit dem Wert 0 belegt.

- -e
  Alle Texte erscheinen in englischer Sprache. Die X11-Resource hat laut Vorgabe den Wert „off" und heißt `*englisch`.

- -mWi, -mWiG, -mPo, -mWaG
  Mit Hilfe dieser Optionen kann bestimmt werden, welche Meßwerte vom Simulationsprogramm ausgegeben werden. Diese Angaben können jedoch auch nach dem Starten des Programms noch durch die Benutzeroberfläche geändert werden. Die X11-Resourcen dazu lauten `*mitWinkel`, `*mitWiGeschw`, `*mitPosition` und `*mitWaGeschw`. Vorgabemäßig werden Winkel und Winkelgeschwindigkeit als Meßwerte verwendet.

Die grafische Benutzeroberfläche der Pendelsimulation ist in Abb. 6.2 zu sehen. Im unteren Bereich des Fensters wird das Pendel grafisch dargestellt. Die Größe der Darstellung hängt von der Fenstergröße ab und kann somit beliebig eingestellt werden. Die Strecke, auf der sich das Pendel bewegt, wird durch eine gestrichelte Linie markiert. Die Strecke kann dabei prinzipiell als endlos angesehen werden, das heißt, sie wird nicht durch den linken oder rechten Fensterrand begrenzt. Wenn das Pendel sich dem Rand nähert, wird der grafisch dargestellte Ausschnitt der Strecke in die entsprechende Richtung verschoben. Es entsteht dabei der Eindruck, daß sich die Strecke, also die gestrichelte Linie, in die entgegengesetzte Richtung des Pendels bewegt.

Es stehen vier Meßgrößen zur Verfügung, die dazu benutzt werden können, den Zustand des Pendels zu beschreiben. Es ist jedoch nicht zwingend, alle zu benutzen. Für jede existiert eine Eingabefeld, in das entweder der Startwert oder der Text „*ohne*" eingetragen wird. Die mit „*ohne*" gekennzeichneten Meßgrößen werden vom Simulationsprogramm nicht ausgegebem und sind damit für den Regler nicht sichtbar. Ihr Startwert wird auf Null gesetzt. In Abb. 6.2 werden nur der Winkel und die Winkelgeschwindigkeit als Meßwerte ausgegeben. Diese Einstellung wird vorgabemäßig benutzt, wenn keine Parameter beim Programmaufruf angegeben werden.

Die Startwerte geben den Zustand des Pendels zu Beginn des Regelungsvorgangs an. In Abb. 6.2 beträgt der für den Winkel 23 und der für die Winkelgeschwindigkeit 0.

Das Pendel wird durch den Button Start gestartet und durch Stop gestoppt. Beim Starten werden die Meßwerte in den Standard-Ausgabekanal geschrieben, und das Pendel ist nun bereit, einen Stellwert einzulesen. Wird aus dem Standard-Eingabekanal ein Wert für die Kraft gelesen, berechnet sich daraus ein neuer Zustand. Durch die Ausgabe der neuen Meßwerte beginnt der Zyklus dann von vorn.

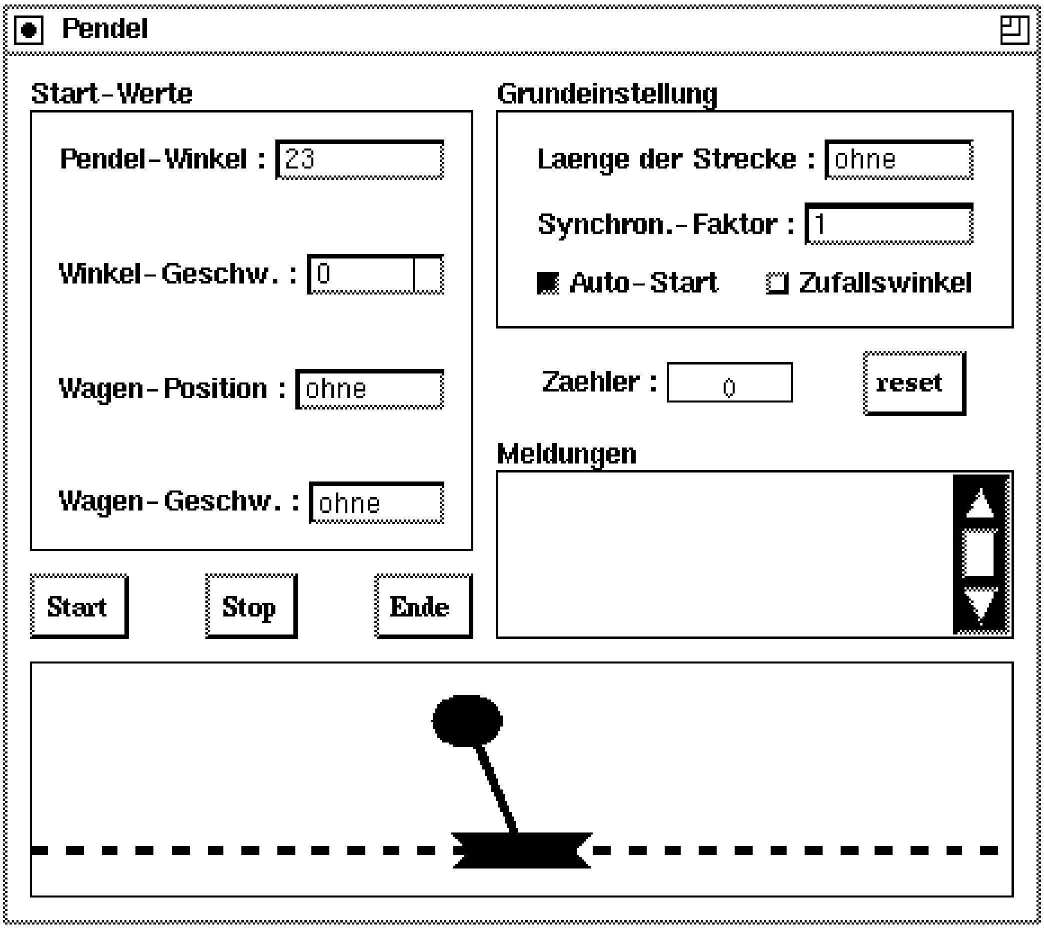


Abbildung 6.2: Die Pendelsimulation

Er wird unterbrochen, wenn das Pendel gestoppt wird oder der Regelungsvorgang fehlschlägt. Letzteres ist genau dann der Fall, wenn das Pendel umkippt oder das Streckenende erreicht.

Wenn der Regelungsvorgang vom Regler aus unterbrochen wird, wenn also keine Stellwerte mehr kommen, muß das Pendel, bevor es erneut gestartet werden kann, gestoppt werden.

Der Bereich der Benutzeroberfläche, der mit *Grundeinstellung* betitelt ist, enthält zwei Eingabefelder, durch die die Länge der Strecke und der sogenannte *Synchronisations-Faktor* festgelegt werden können. Wird für die Länge, wie in Abb. 6.2 zu sehen, der Text „*ohne*“ eingetragen, wird die Strecke als unendlich lang angenommen.

Der Synchronisations-Faktor gibt an, wie oft der Zustand des Pendels zwischen zwei Stellwerten neu berechnet werden soll. Beim erstenmal wird dabei die eingelesene Kraft benutzt, alle nachfolgenden Berechnungen legen für die Kraft den Wert Null zugrunde. Durch eine Erhöhung des Synchronisations-Faktor erhöht sich also die Geschwindigkeit des simulierten Pendels relativ zu der des Reglers.

Über die beiden Auswahl-Buttons Auto-Start und Zufallswinkel kann das Verhalten beim Starten und Stoppen der Pendelsimulation beeinflußt werden. Ist der Button Auto-Start aktiv, wird das Pendel automatisch neu gestartet, nachdem es durch Umkippen oder Erreichen des Streckenendes gestoppt wurde. Falls Zufallswinkel aktiv ist, wird bei jedem Start ein zufälliger Winkel zwischen -40 und 40 Grad als Startwert erzeugt. Das Aktivieren beider Buttons ist dann sinnvoll, wenn der Regler versucht, das Verhalten des Pendels in möglichst allen Situationen zu testen. Dies ist beispielsweise der Fall, wenn durch NEFCON-I eine Regelbasis erlernt wird.

Außerdem enthält die Benutzeroberfläche einen Zähler für die Anzahl der abgearbeiteten Zyklen. Dieser kann durch den Button reset jederzeit auf Null zurückgesetzt werden.

Beendet wird das Simulationsprogramm durch den Button Ende. Es ist jedoch nur möglich das Programm zu beenden, wenn die Pendelsimulation zuvor gestoppt wurde.

6.2 Die Implementierung

Das Pendel wird durch die drei Klassen `PendelSim`, `PendelGrafik` und `PendelFenster` implementiert. Diese Klassen und ihre Abhängigkeiten sind in Abb. 6.3 zu sehen.

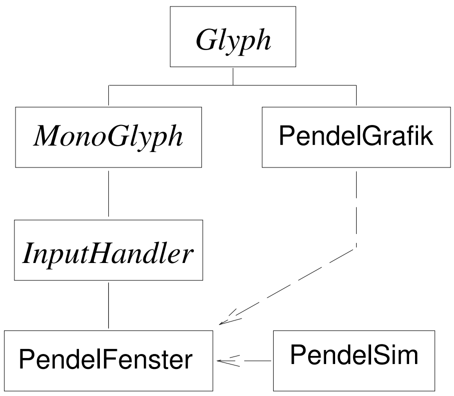


Abbildung 6.3: Klassen zur Implementierung des Pendels

Die mathematische Simulation des inversen Pendels übernimmt die Klasse `PendelSim`. Ihre Schnittstelle ist in Abb. 5.2 zu sehen.

Die Berechnung eines neuen Systemzustands erfolgt durch die Elementfunktion `simulation`, der als Parameter eine Kraft übergeben wird. Der neue Zustand kann durch die Elementfunktionen `position`, `wa_geschw`, `winkel` und `wi_geschw` jederzeit abgefragt oder durch Setzen bestimmter Werte beliebig verändert werden.

```
class PendelSim
{
  public : PendelSim();
           void simulation(float kraft);

           float position();          // Position des Wagens
           void position(float);

           float wa_geschw();         // Geschwindigkeit des Wagens
           void wa_geschw(float);

           float winkel();            // Winkel des Pendels
           void winkel(float);

           float wi_geschw();         // Winkelgeschwindigkeit
           void wi_geschw(float);
};
```

Abbildung 6.4: Schnittstelle der Klasse PendelSim

Falls für die Simulation andere mathematische Formeln benutzt werden sollen, sind die Elementfunktionen der Klasse `PendelSim` einfach auszutauschen.

Um das Pendel grafisch darzustellen, wurde die Klasse `PendelGrafik` implementiert. Diese Klasse zeichnet ein Pendel, dessen Winkel beliebig verändert und das auf einer Strecke beliebig weit nach links oder rechts verschoben werden kann. Falls die neue Position außerhalb des dargestellten Bereiches liegt, wird statt des Pendels die Strecke verschoben.

Das gesamte Fenster, das in Abb. 6.4 zu sehen ist, wird durch die Klasse `PendelFenster` implementiert. Diese Klasse benutzt je eine Instanz von `PendelSim` und `PendelGrafik`. Die Benutzeroberfläche wird aus denselben Grafikelementen zusammengesetzt, wie sie in NEFCON-I benutzt werden. Das heißt, auch für die Implemetierung des Simualtionsprogrammes wurde das Modul `nfclook` benutzt.

Die Überwachung des Eingabekanals erfolgt hier, genau wie bei NEFCON-I, durch eine Instanz der Klasse `Dispatcher`. Dadurch wird es ermöglicht , daß auch während des Regelungsvorgangs auf Eingaben in der Benutzeroberfläche reagiert werden kann.

Anhang A

Aufruf-Parameter

In den beiden folgenden Tabellen werden die Aufruf-Parameter der Programme `nefcon` und `pendel` dargestellt. Die Tabelleneinträge in der Spalte *Option* geben die Aufrufoptionen an, die beim Starten benutzt werden können. In der Spalte *Resource* sind die Namen der dazugehörigen X11-Resourcen eingetragen. Der Vorgabewert, mit dem diese Resourcen belegt sind, ist in der dritten Spalte enthalten.

Die Aufruf-Parameter von `nefcon` lauten:

Option	Resource	Vorgabe	Zweck
-dir=**Verzeichnis**	*DatenVerzeichnis	/daten	Zum Laden und Speichern der Dateien „`*.rg`“ und „`*.lv`“
-p=**Name**	*protokoll	protokoll.nfc	Name der Protokoll-datei
-e	*englisch	off	gibt an, ob englische Texte zu verwenden sind
-editorfont=**Font**	*NfcEditorFont*font	*helvetica-medium-r-normal–12*	Font für Ein- Ausgabefelder und Toggle-Buttons
-infofont=**Font**	*NfcInfoFont*font	*helvetica-bold-r-normal–12*	Font für Beschriftungen
(-)	*PushButton*font	*times-bold-r-normal–12*	Font für Buttons

Tabelle A.1: Aufruf-Parameter für NEFCON-I

Die Aufruf-Parameter von `pendel` lauten:

Option	Resource	Vorgabe	Zweck
-laenge **wert**	*LaengeStrecke	0	Länge der Pendelstrecke in Metern. Bei einer Länge von 0 m wird Strecke als unendlich angesehen.
-e	*englisch	off	Gibt an, ob englische Texte zu verwenden sind.
-mWi	*mitWinkel	on	Winkel als Meßwert ?
-mWiG	*mitWiGeschw	on	Winkelgeschwindigkeit als Meßwert ?
-mPo	*mitPosition	off	Wagenposition als Meßwert ?
-mWaG	*mitWaGeschw	off	Wagengeschwindigkeit als Meßwert ?
-editorfont=**Font**	*NfcEditorFont*font	*helvetica-medium-r-normal–12*	Font für Ein-Ausgabefelder.
-infofont=**Font**	*NfcInfoFont*font	*helvetica-bold-r-normal–12*	Font für Beschriftungen.
(-)	*PushButton*font	*times-bold-r-normal–12*	Font für Buttons.

Tabelle A.2: Aufruf-Parameter für die Pendelsimulation

In der folgenden Tabelle sind ausgewählte InterViews-Optionen und Resourcen dargestellt. Sie können für beliebige InterViews-Programme, also auch für `nefcon` und `pendel`, benutzt werden.

Option	Resource	Zweck
-monochrome	*gui	Das Erscheinungsbild der Grafikelemente ist einem monochromem Bildschirm angepaßt
-motif	*gui	Das Erscheinungsbild ähnelt dem von Motif
-smotif	*gui	Das Erscheinungsbild ähnelt dem von SGIMotif
-openlook	*gui	Das Erscheinungsbild ähnelt dem von OpenLook
-bg	*background	Hintergrundfarbe
-fg	*foreground	Vordergrundfarbe
(-)	*FieldEditor.background	Hintergrundfarbe für Eingabefelder
(-)	*cursorFlashRate	Frequenz, in der Cursor im Eingabefeld blinkt

Tabelle A.3: Ausgewählte InterViews-Optionen

Anhang B

Dateiformate

Die Dateiformate sind hier in einer Backus-Naur ähnlichen Notation angegeben. Es ist dabei anzumerken, daß alle Zahlen durch ein Leerzeichen abgeschlossen werden. Zeichenketten werden durch ein '\n' begrenzt.

Dateiformat für linguistische Variablen:

*.lv	=	**Anz-LV**, (**LingVar**)$^{Anz-LV}$
Anz-LV	=	„Anzahl der linguistischen Variablen“
LingVar	=	**LV-Name**, **Min**, **Max**, **Typ**, **Anz-FM**, (**FuzzyMenge**)$^{Anz-FM}$
LV-Name	=	„Name der linguistischen Variable“
Min	=	„untere Grenze des Definitionsbereiches“
Max	=	„obere Grenze des Definitionsbereiches“
Typ	=	„0 für Eingabevariablen, 1 für Ausgabevariablen“
Anz-FM	=	„Anzahl der Fuzzy-Mengen“
FuzzyMenge	=	**FM-Name**, [**Grenzen**], **Anz-IV**, (**Intervall**)$^{Anz-IV}$
FM-Name	=	„Name der Fuzzy-Menge“
Grenzen	=	'%', **untere-Gr**, **obere-Gr**
untere-Gr	=	„minimale Intervallbreite“
obere-Gr	=	„maximale Intervallbreite“
Anz-IV	=	„Anzahl der Intervalle“
Intervall	=	**x1**, **y1**, **drin1**, **x2**, **y2**, **drin2**
x1	=	„x-Koordinate des linken Begrenzungspunktes“
y1	=	„y-Koordinate des linken Begrenzungspunktes“
drin1	=	„'true' falls linker Punkt im Intervall enthalten“
x2	=	„x-Koordinate des rechten Begrenzungspunktes“
y2	=	„y-Koordinate des rechten Begrenzungspunktes“
drin2	=	„'true' falls rechter Punkt im Intervall enthalten“

Dateiformat für linguistische Regeln:

*.rg	=	**Anz-Rg**, (**Regel**)$^{Anz-Rg}$
Anz-Rg	=	„Anzahl der linguistischen Regeln“
Regel	=	**Anz-Pra**, **Anz-Kon**, (**Bedingung**)$^{Anz-Pra}$, (**Bedingung**)$^{Anz-Kon}$
Anz-Pra	=	„Anzahl der Bedingungen in der Prämisse der Regel“
Anz-Kon	=	„Anzahl der Bedingungen in der Konklusion der Regel“
Bedingung	=	**LV-Name**, **FM-Name**
LV-Name	=	„Name der linguistischen Variable“
FM-Name	=	„Name der Fuzzy-Menge“

Anhang C

Die Prozeßkommunikation

Mit Prozeßkommunikation ist hier die Kommunikation des Reglers mit dem Anwendungsprogramm gemeint. Das in diesem Anhang dargestellte Protokoll muß von Programmen eingehalten werden, die einen Prozeß simulieren, der von NEFCON-I geregelt werden soll.

Aus der Sicht des Anwendungsprogramms erfolgt die gesamte Kommunikation über die Standard-Ein- und Ausgabekanäle. Der Regelungsvorgang wird einfach durch die Ausgabe der ersten Meßwerte gestartet. Sobald alle Meßwerte ausgegeben wurden, antwortet der Regler, vorausgesetzt er wurde bereits gestartet, mit einem ober mehreren Stellwerten. Die Anzahl der Meß- und Stellwerte ergibt sich aus der Wissensbasis des Reglers. Folgende Punkte sind nun zu beachten:

1. Alle Ausgaben sind sind durch ein '\n' abzuschließen. Ebenso werden von NEFCON-I alle Stellwerte durch dieses Zeichen getrennt.

2. Nachdem der Regler gestartet wurde, werden alle Meßwerte von ihm bearbeitet. Das heißt, er antwortet auf jeden Satz von Meßwerten mit einer entsprechenden Anzahl von Stellwerten. Wenn also während des Regelungsvorgangs öfter Meßwerte ausgegeben als Stellwerte gelesen werden, führt das zu Problemen.

3. Bei einem Fehlschlag der Regelung, bezogen auf das Beispiel der Pendelsimulation wäre dies beispielsweise das Umfallen des Pendels, wird dem Regler dies durch die Ausgabe des Zeichens 'r' (reset) mitgeteilt. Das Zeichen wird vor den entsprechenden Meßwerten ausgegeben, die diese Situation beschreiben.

   Der Regler benötigt diese Information zum einem für das Protokoll, und zum anderen weiß er, daß der Regelungsvorgang danach von vorn beginnt.

4. Das Lesen des Zeichens EOF bedeutet, daß das Programm NEFCON-I beendet wurde.

Anhang D

Berechnung der Stellwerte

In diesem Anhang werden die drei Versionen der virtuellen Elementfunktion `berechne_steuerwerte` dargestellt, wie sie von den Klassen `FuzController`, `NeuroFuzController` und `RgLrnNeuroFuzCtrl` definiert werden.

FuzController::berechne_steuerwerte(...)

```
FuzzyTyp* FuzController::berechne_steuerwerte(FuzzyTyp* eingaben)
{
  _eingabe→messwerte(eingaben);

  for(int i = 0; i < _anz_ausgabe_var; i++ ) {
    _treffer_summe[i] = 0;
    _ausgabe[i] = 0;
  }

  int fuz_nr;
  FuzzyTyp wert, gewicht;
  for(int regel = 0; regel < _anz_regeln; regel++) {

    // Treffergrade fuer die Regeln berechnen

    FuzzyTyp treffer = 1;
    for(int var = 0; var < _anz_eingabe_var; var++)
      if((fuz_nr = _regeln[regel]→pra(var)) ≠ -1)
        treffer = _t_norm(treffer, _eingabe→fuzzy_messwert(var, fuz_nr));
    _regeln[regel]→treffer_grad(treffer);

    // Steuerwerte fuer jede Ausgabevariable berechnen
```

```
    for(var = 0; var < _anz_ausgabe_var; var++)
      if((fuz_nr = _regeln[regel]→kon(var)) ≠ -1) {
        if(treffer ≤ 0) {
          wert = 0;
          gewicht = 0;
        } else
          wert = _defuz(
                    treffer,
                    _lingvarsatz→hol_lingvar(
                      AusgabeVar, var
                    )→hol_fuzzyset(fuz_nr),
                    gewicht
                  );
        _regeln[regel]→steuer_wert(var, wert); // Wert merken

        _ausgabe[var] += wert * gewicht; // Werte aufaddieren

        _treffer_summe[var] += gewicht; // Gesamtgewicht bilden

      }
  }
  for(int var = 0; var < _anz_ausgabe_var; var++)
    if(_treffer_summe[var] ≠ 0)
      _ausgabe[var] /= _treffer_summe[var]; // Mittelwert bilden

    else
      _ausgabe[var] = 0; // so llte nicht sein


  return _ausgabe;
}
```

NeuroFuzController::berechne_steuerwerte(...)

```
FuzzyTyp* NeuroFuzController::berechne_steuerwerte(FuzzyTyp* eingaben )
{
  FuzzyTyp system_fehler = *_fehler_ctrl→berechne_steuerwerte(eingaben);

  // Fuzzy-Sets optimieren

  if(!_erster_durchlauf)
```

```
    _optimieren(system_fehler);
  else
    _erster_durchlauf = false;

  // Steuerwerte berechnen

  FuzzyTyp* ausgaben = FuzController::berechne_steuerwerte(eingaben);

  // Protokoll erstellen

  _protokollieren(eingaben, ausgaben, system_fehler);

  // Steuerwerte zurueckliefern

  return ausgaben;
}
```

RgLrnNeuroFuzCtrl::berechne_steuerwerte(...)

```
FuzzyTyp* RgLrnNeuroFuzCtrl::berechne_steuerwerte(FuzzyTyp* eingaben)
{
  //——————————————-

  if(_lern_strukt.schritte_phase1 > 0) {
    _lern_strukt.schritte_phase1--;

    // Lernphase 1

    FuzzyTyp* ausgaben = FuzController::berechne_steuerwerte(eingaben);
    _regeln_lern_phase1(*_fehler_ctrl→berechne_steuerwerte(eingaben));

    // letzter Lernschritt ?

    if(_lern_strukt.schritte_phase1 == 0) {
      _erster_durchlauf = true;

      // Zaehler werden in der 2. Lernphase erneut benoetigt

      for(int i = 0; i < _anz_regeln; i++) {
        _regeln[i]→zaehler1() = 0;
        _regeln[i]→zaehler2() = 0;
```

```
        }
    }

    return ausgaben;
}

//———————————————-

if(_lern_strukt.schritte_phase2 > 0) {
    _lern_strukt.schritte_phase2--;

    // Lernphase 2

    FuzzyTyp fehler = *_fehler_ctrl→berechne_steuerwerte(eingaben);
    if(!_erster_durchlauf) {

        if(_lern_strukt.fuz_trainieren) {
            _optimieren(fehler);
            if(_protokoll ≠ nil)
                _protokoll→aktua_lingvarsatz();
        }

        _regeln_lern_phase2(fehler);
    } else
        _erster_durchlauf = false;

    // beim letzten Lernschritt RegelBasis erzeugen

    if(_lern_strukt.schritte_phase2 == 0) {
        for(int i = 0; i < _anz_regeln; i++)
            if(_regeln[i]→zaehler2() > 0)
                _regeln[i]→fehler() /= _regeln[i]→zaehler2();

        _erzeuge_regelbasis(_regelbasis);
        if(_protokoll ≠ nil)
            _protokoll→init(_regelbasis);
    }

    return _rnd_steuerwerte(eingaben);
}

//———————————————-

// Lernphasen beendet, ab jetzt Fuzzy-Sets optimieren
```

```
  if(_anz_regeln == 0) {
    for(int i = 0; i < _anz_ausgabe_var; _ausgabe[i++] = 0);
    return _ausgabe;
  } else
    return NeuroFuzController::berechne_steuerwerte(eingaben);
}
```

Anhang E

Die Programminstallation

Um das Programm zu installieren müssen in der Regel die NEFCON-I-Quelldateien auf dem jeweiligen System neu übersetzt werden. Voraussetzung dafür ist das Vorhandensein der Grafikbibliothek *InterViews 3.1*.

Das Übersetzen der Quelldateien erfolgt mit Hilfe eines sogenannten *Imakefiles*. Aus diesem wird durch das Kommando `ivmkmf` das eigentliche *Makefile* erzeugt. Vor dem Aufruf von `ivmkmf` muß evtl. die *Environment*-Variable „CPU" auf den Wert „SUN4" gesetzt werden. Das so erzeugte Makefile wird dann einfach durch den Befehl `make` aufgerufen. Nach erfolgreicher Übersetzung liegt das ausführbare Programm unter dem Namen `nefcon` im Unterverzeichnis `SUN4`.

Zur Installation von NEFCON-I kann folgendes Imakefile verwendet werden:

```
AOUT = nefcon

#ifdef InObjectCodeDir

Use_libInterViews()
ComplexProgramTarget(nefcon-I)

#define Obj(file) MakeObjectFromSrc(file)

Obj(texte)
Obj(global)

Obj(lingvar)
Obj(regel)
Obj(controller)
Obj(anwendung)
```

```
Obj(wissensbank)

Obj(nfclook)
Obj(koosystem)
Obj(datei)
Obj(lingvar_ed)
Obj(regel_tafel)
Obj(regel_tab)
Obj(regel_ed)
Obj(anwend_ed)
Obj(protokoll_ed)
Obj(lern_ed)
Obj(protokoll)
Obj(control_ed)
Obj(nfc_ed)
Obj(main)

#else

MakeInObjectCodeDir()

#endif
```

Literaturverzeichnis

[Berenji, 1992] H. R. Berenji (1992). *A Reinforcement Learning-Based Architecture for Fuzzy Logic Control.* Int. J. Approximate Reasoning, 6:267 –292.

[Eklund et al., 1992] P. Eklund, F. Klawonn und D. Nauck (1992). *Distributing Errors in Neural Fuzzy Control.* In *Proc. 2nd Int. Conf. on Fuzzy Logic and Neural Networks (IIZUKA'92)*, Seiten 1139–1142.

[Foerster, 1993] S. Foerster (1993). *Zu Kombinationen Neuronaler Netze und Fuzzy Systeme.* Diplomarbeit, Technische Universität Braunschweig.

[Hayashi et al., 1992] I. Hayashi, H. Nomura, H. Yamasaki und N. Wakami (1992). *Construction of Fuzzy Inference Rules by Neural Network Driven Fuzzy Reasoning and Neural Network Driven Fuzzy Reasoning With Learning Functions.* Int. J. Approximate Reasoning, 6:241–266.

[Hebb, 1949] D. O. Hebb (1949). *The Organization of Behavior.* Wiley, New York.

[Ichihashi, 1991] H. Ichihashi (1991). *Iterative Fuzzy Modelling and a Hierarchical Network.* In R. Lowen und M. Roubens, Hrsg.: *Proc. 4th IFSA Congress*, Seiten 49–52, Brussels.

[Kemke, 1988] C. Kemke (1988). *Der neuere Konnektionismus.* Informatik Spektrum, Seiten 143–162.

[Kruse et al., 1993] R. Kruse, F. Klawonn und J. Gebhardt (1993). *Fuzzy Systeme.* Teubner, Stuttgart.

[Linton et al., 1992] M. A. Linton, P. R. Calder, J. A. Interrante, S. Tang und J. M. Vlissides (1992). *InterViews Reference Manual Version 3.1.* The Board of Trustees of the Leland Stanford Junior University.

[Mayer, 1992] S. Mayer (1992). *Einführung in InterViews.* Technischer Bericht, TU München, Institut für Informatik.

[Nauck, 1993] D. Nauck (1993). *Modellierung Neuronaler Fuzzy-Regler.* Dissertation, Technische Universität Braunschweig.

[NAUCK et al., 1993] D. NAUCK, F. KLAWONN und R. KRUSE (1993). *Combining Neural Networks and Fuzzy Controllers.* In E. P. KLEMENT und W. SLANY, Hrsg.: *Fuzzy Logic in Artificial Intelligence (FLAI93)*, Nummer 695 in *Lecture Notes in Artificial Intelligence*, Seiten 35–46, Berlin. Springer.

[NAUCK und KRUSE, 1992] D. NAUCK und R. KRUSE (1992). *A Neural Fuzzy Controller Learning by Fuzzy Error Propagation.* In *Proc. Workshop of North American Fuzzy Information Processing Society (NAFIPS92)*, Seiten 388–397, Puerto Vallarta.

[NOMURA et al., 1992] H. NOMURA, I. HAYASHI und N. WAKAMI (1992). *A Learning Method of Fuzzy Inference Rules by Descent Method.* In *Proc. IEEE Int. Conf. on Fuzzy Systems 1992*, Seiten 203–210, San Diego.

[PEDRYCZ und CARD, 1992] W. PEDRYCZ und W. C. CARD (1992). *Linguistic Interpretation of Self-Organizing Maps.* In *Proc. IEEE Int. Conf. on Fuzzy Systems 1992*, Seiten 371–378, San Diego.

[RITTER, 1992] S. RITTER, MARTINEZ (1992). *Neuronale Netze.* Addison-Wesley.

[ROJAS, 1993] R. ROJAS (1993). *Theorie der neuronalen Netze: Eine systematische Einführung.* Springer.

[ROSEMAN, 1992] M. ROSEMAN (1992). *A Not-Entirely Gentle Introduction to InterViews.* Technischer Bericht, University of Calgary, Department of Computer Science.

[RUMELHART et al., 1986a] D. E. RUMELHART, G. E. HINTON und R. J. WILLIAMS (1986a). *Learning Internal Representations by Error Propagation.* In [RUMELHART und MCCLELLAND, 1986], Seiten 318–362. MIT Press, Cambridge.

[RUMELHART et al., 1986b] D. E. RUMELHART, G. E. HINTON und R. J. WILLIAMS (1986b). *Learning Representations by Back-Propagating Errors.* Nature, 323:533–536.

[RUMELHART und MCCLELLAND, 1986] D. E. RUMELHART und J. L. MCCLELLAND, Hrsg. (1986). *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, Band 1. MIT Press, Cambridge.

[TAKAGI und HAYASHI, 1991] H. TAKAGI und I. HAYASHI (1991). *NN-Driven Fuzzy Reasoning.* Int. J. Approximate Reasoning, 5:191–212.

[TAKAGI et al., 1992] H. TAKAGI, N. SUZUKI, T. KODA und Y. KOJIMA (1992). *Neural Networks Designed on Approximate Reasoning Architecture and Their Applications.* IEEE Trans. Neural Networks, 3:752–760.

[WALOSZEK, 1988] G. WALOSZEK (1988). *Einführung in parallele Gedächtnismodelle.* Technischer Bericht, Universität Trier.

[WERBOS, 1992] P. J. WERBOS (1992). *Neurocontrol: Where it is Going and Why it is Crucial.* Artificial Neural Networks, 2:61–68.

[WHITE und SOFGE, 1992] D. A. WHITE und D. A. SOFGE, Hrsg. (1992). *Handbook of Intelligent Control. Neural, Fuzzy, and Adaptive Approaches.* Van Nostrand Reinhold, New York.