Technische Universität Braunschweig
Institut für Rechnerverbund und Betriebssysteme

Diplomarbeit

NEFCON-I: Entwurf und Implementierung einer Erweiterung des Softwaretools NEFCON-I für Neuronale Fuzzy-Regler

von
cand. inform. Roland Stellmach

Aufgabenstellung und Betreuung:

Prof. Dr. R. Kruse und Dr. D. Nauck

Braunschweig, im April 1995

Abstract

Construction and optimization of fuzzy controllers often falls back upon heuristic. Formal construction methods does not exist yet. An approach to supporting the optimization process consists of combination of neural- and fuzzy methods. Within this theses a model of a hybrid neural fuzzy controller called the NEFCON model, will be considered. For this model new learning algorithms were developed and implemented. Additionally, some restrictions existing in the original model were removed and the graphical surface was revised.

Kurzfassung

In der Entwurfs- und Optimierungsphase eines Fuzzy–Reglers muß oft mangels formaler Entwurfsmethoden auf Heuristiken zurückgegriffen werden. Ein Ansatz zur Unterstützung des Optimierungsprozesses besteht in der Kombination Neuro- und Fuzzy–Konzepte. Im Rahmen dieser Arbeit wird das Modell eines hybriden Neuronalen Fuzzy–Reglers, das NEFCON–Modell, vorgestellt. Für dieses Modell werden neue Lernalgorithmen entwickelt und implementiert. Zusätzlich werden einige Beschränkungen des ursprünglichen Modells behoben und die graphische Benutzeroberfläche überarbeitet.

Inhaltsverzeichnis

Abbildungsverzeichnis

Einleitung

Im Rahmen des NEFCON–Projektes (Neural Fuzzy Control) an der TU Braunschweig werden Strategien entwickelt und erprobt, die Fuzzy–Regler um die Lernfähigkeit Neuronaler Netze erweitern (siehe Nauck, Klawonn, Kruse: Neuronale Netze und Fuzzy–Systeme, Vieweg, 1994). Das Ergebnis dieser Vereinigung von Fuzzy–Methoden und neuronalen Konzepten zu einem "hybriden" Neuronalen Fuzzy–Regler ist sowohl als Fuzzy–Regler als auch als Neuronales Netz mit Fuzzy–Mengen als Gewichten interpretierbar. Dieser Ansatz bietet also die Möglichkeit, die Lernfähigkeit der Neuronalen Netze mit den Transparenzeigenschaften der Fuzzy–Regler und ihrer Möglichkeit zur Nutzung von a–priori Wissen zu verbinden.

Das Lernverfahren für das NEFCON–Modell basiert auf einem "Fuzzy–Fehlermaß", das den Zustand des zu regelnden Systems beschreibt. Dieses Fehlermaß wird mit Hilfe von linguistischen Regeln definiert und dient als Grundlage für einen überwachten Lernalgorithmus, der die Methoden des "reinforcement learning" nutzt.

Beim NEFCON–Modell besteht ein hybrider Neuronaler Fuzzy–Regler aus zwei parallel arbeitenden, einfachen Fuzzy–Reglern. Der erste, der sogenannte Fehler–Regler, berechnet aus den Meßwerten ein Fuzzy–Fehlermaß, das von einem Lernalgorithmus zur Veränderung der Regelbasis des zweiten Reglers (Handlungsregler) herangezogen wird.

Diese Arbeit baut auf dem, über ftp [1] bereits erhältlichen, Programmpaket "NEFCON–I" von Hermann Dieckgerdes auf. Meine Aufgabe besteht darin, das System zu erweitern und neue Lernalgorithmen zu entwickeln und zu erproben. Die wichtigste Aufgabe dabei war, einen Lernalgorithmus zu entwickeln, mit dessen Hilfe die Beschränkung, nur lineare Fuzzy–Mengen in den Konklusionen des

[1] `ftp.ibr.cs.tu-bs.de:pub/local/nefcon`

Handlungsreglers benutzen zu dürfen, aufgehoben werden konnte. Dazu habe ich eine Methode entwickelt, die erlaubt, Fuzzy–Mengen durch Verschieben und nicht wie bisher, nur durch Veränderung der Breite zu optimieren.

Außerdem beschäftigte ich mich auch mit der Erzeugung und dem Erlernen neuer Regelbasen, sowohl bei gegebener als auch bei zu erlernender Partitionierung der linguistischen Variablen.

Zusätzlich habe ich die in C++ und InterViews [2] geschriebene Simulationsoberfläche für das NEFCON–Modell –gemäß den Wünschen einiger Benutzer– umgestaltet und erweitert, um ein komfortableres Arbeiten zu ermöglichen.

[2]**InterViews** ist eine C++ Klassenbibliothek zum Aufbau von Bedieneroberflächen unter X–Windows.

Kapitel 1

Das NEFCON–Modell

In diesem Kapitel wird eine hybride Architektur für einen Neuronalen Fuzzy–Regler [1] und dafür entwickelte Lernalgorithmen vorgestellt.

Das hier vorzustellende Modell, das NEFCON–Modell, kann sowohl als gewöhnlicher Fuzzy–Regler als auch als Neuronales–Netzwerk interpretiert werden. Das NEFCON–Modell erfüllt somit die Anforderungen an ein hybrides Modell. Es vereinigt die Lernfähigkeit eines Neuronalen–Netzes mit der Transparenz eines Fuzzy–Reglers und erlaubt die Nutzung von a–priori Wissen. Insbesondere bleibt während des Lernvorgangs die zugrundeliegende Struktur des Systems erhalten. Die vom Lernalgorithmus vorgenommenen Änderungen bleiben sowohl während als auch nach Abschluß des Lernvorgangs interpretierbar [Nauck 1993].

Das Lernverfahren für NEFCON basiert auf dem Fuzzy–Fehlermaß, das den Zustand des zu regelnden Systems beschreibt. Da der korrekte Stellwert zu einem gegebenen Systemzustand nicht angegeben werden kann, scheidet die Verwendung eines überwachten Lernalgorithmus mit fester Lernaufgabe aus. Aus gleichen Gründen ist es nicht möglich, direkt den Fehler zu bestimmen. Dennoch kann das Wissen über den Zustand des Systems in Termen wie "*gut*", "*schlecht*" oder "*optimal*" ausgedrückt werden. Meistens ist das Ziel der Regelung, der optimale Zustand, bekannt. Es liegt daher nahe, die Güte des Systemzustands durch

[1] Neuronale Fuzzy–Regler mit hybrider Architektur verbinden die Architekturen von Fuzzy–Reglern und Neuronalen–Netzen miteinander. Bei dem Ansatz der kooperativen Architektur arbeiten Fuzzy–Regler und Neuronale–Netzwerke zusammen, wobei ihre Aufgaben voneinander getrennt sind.

eine Fuzzy–Relation mit Hilfe linguistischer Regeln zu beschreiben, woraus sich der Fuzzy–Fehler ableiten läßt [Nauck 1993]

Der zu entwickelnde überwachte Lernalgorithmus mit einer freien Lernaufgabe wird also das Prinzip des Verstärkenden Lernens (reinforcement learning) [Nauck et al. 1994] nutzen. Das Verstärkungssignal wird auf dem wissensbasierten Fuzzy–Fehlermaß aufbauen.

1.1 Die Architektur von NEFCON

Im folgenden wird ein technisches System S mit n Meßgrößen ξ_i $(i = 1, \ldots, n)$ und einer Stellgröße η und den zugehörigen Grundmengen $X_1, \ldots, X_n$ und Y betrachtet.

Das Bild 1.1 stellt als Beispiel die Architektur eines NEFCON–Systems mit zwei Eingangsgrößen, fünf Regeln und einer Stellgröße dar [Nauck 1993].

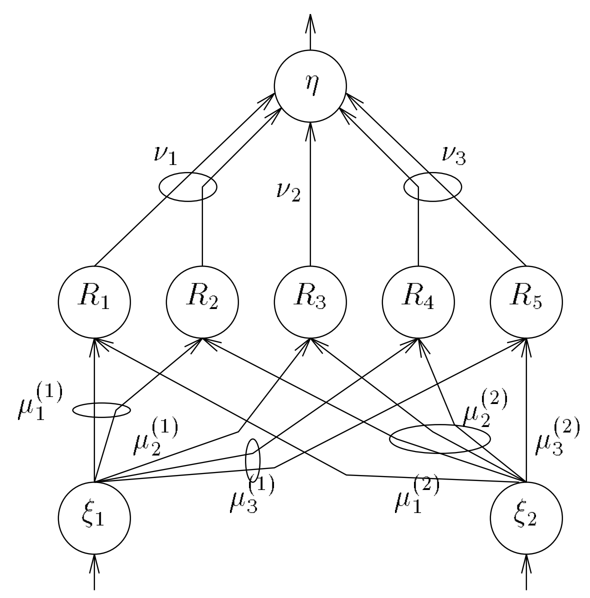

Abbildung 1.1: NEFCON–System mit fünf Regeln, zwei Eingangsgrößen und einer Stellgröße

Das Modell entspricht einem dreischichtigen, vorwärtsbetriebenen Neuronalem Netz. Die Eingabeschicht besteht aus Einheiten, die die externe Eingabe entgegennehmen und ggf. transformieren. Die innere Schicht enthält Einheiten, die linguistische Regeln repräsentieren. Ihre Aktivierungen entsprechen dem Erfüllungsgrad ihrer Prämissen. Die Ausgabeschicht enthält genau eine Ausgabeeinheit, die einen scharfen Stellwert liefert.

Im Unterschied zu gewöhnlichen Neuronalen Netzen gibt es im NEFCON–Modell Verbindungen zwischen den einzelnen Schichten, die Gewichte gemeinsam benutzen (shared weights, coupled links). Im Bild 1.1 haben z.B. die Verbindung von der Einheit ξ_1, zu den Einheiten R_1 und R_2 dasselbe Gewicht $\mu_1^{(1)}$. Ein Lernalgorithmus muß also Änderungen an allen gekoppelten Verbindungen in gleicher Weise durchführen.

Die Verbindungsgewichte sind Fuzzy–Mengen (nicht Zahlenwerte wie bei herkömmlichen Neuronalen–Netzen). Die Gewichte zwischen der Eingabeschicht und der inneren Schicht repräsentieren also die Fuzzy–Mengen der Prämissen der Fuzzy–Regeln, und die Gewichte der Verbindungen zwischen den inneren Schichten und der Ausgabeschicht entsprechen den Fuzzy–Mengen der Regelkonklusionen. Die Regelbasis wird also durch die Verbindungsstruktur kodiert (man sieht dies am besten, wenn die Gewichte wie ihre korrespondierenden Fuzzy–Mengen bezeichnet werden).

Im Bild 1.1 steht z.B. die Einheit R_3 für die Regel

$$R_3\text{: } \texttt{IF } \xi_1 \texttt{ is } A_2^{(1)} \texttt{ and } \xi_2 \texttt{ is } A_2^{(2)} \texttt{ THEN } \eta \texttt{ is } B_2$$

mit den linguistischen Termen $A_2^{(1)}, A_2^{(2)}, B_2$ die durch die Fuzzy–Mengen $\mu_2^{(1)}, \mu_2^{(2)}$ und ν_2 repräsentiert werden.

Es folgt das formale Modell von NEFCON als dreischichtiges Fuzzy–Perceptron nach der Dissertationsschrift von Detlef Nauck [Nauck 1993]:

Definition 1.1 *Ein dreischichtiges Fuzzy–Perceptron ist ein dreischichtiges Neuronales Netz* $(U, W, \mathrm{NET}, A, O, \mathrm{ex})$ *mit:*

1. $U = \left(\left(\bigcup_{i \in \{1,2,3\}} U_i \right) \neq \emptyset \right.$ *eine Menge von* Verarbeitungseinheiten.
   Es gilt $(\forall i, j \in \{1, 2, 3\})(U_i \cap U_j = \emptyset$ *für* $i \neq j)$.
   U_1 *heißt* Eingabeschicht, U_2 Regelschicht *und* U_3 Ausgabeschicht.

2. *Die* Netzwerkstruktur *ist* $W : U \times U \to \mathcal{F}(\mathbb{R})$*, wobei ausschließlich* $W(u, v)$ *mit* $u \in U_i$ *und* $v \in U_{i+1}$ *existieren.*

3. A *ordnet jeder Einheit* $u \in U$ *eine Aktivierungsfunktion* A_u *zur Berechnung der Aktivierung* a_u *zu. Es gilt :*

   für $u \in U_1 \cup U_2$*:*

$$A_u : \mathbb{R} \to \mathbb{R}, \quad a_u = A_u(\mathrm{net}_u) = \mathrm{net}_u,$$

   für $u \in U_3$*:*

$$A_u : \mathcal{F}(\mathbb{R}) \to \mathcal{F}(\mathbb{R}), \quad a_u = A_u(\mathrm{net}_u) = \mathrm{net}_u,$$

4. O *ordnet jeder Einheit* $u \in U$ *eine Ausgabefunktion* O_u *zur Berechnung der Ausgabe* o_u *zu. Es gilt :*

   für $u \in U_1 \cup U_2$*:*

$$A_u : \mathbb{R} \to \mathbb{R}, \quad o_u = O_u(o_u) = a_u,$$

   für $u \in U_3$*:*

$$O_u : \mathcal{F}(\mathbb{R}) \to \mathbb{R}, \quad o_u = O_u(a_u) = DEFUZZ_u(\mathrm{net}_u),$$

5. NET *ordnet jeder Einheit* $u \in U$ *eine Netzeingabegabefunktion* NET_u *zur Berechnung der Netzeingabe* o_u *zu. Es gilt :*

   für $u \in U_1$*:*

$$\mathrm{NET}_u : \mathbb{R} \to \mathbb{R}, \quad \mathrm{net}_u = \mathrm{ex}_u,$$

   für $u \in U_2$*:*

$$\mathrm{NET}_u : (\mathbb{R} \times \mathcal{F}(\mathbb{R}))^{U_1} \to [0, 1],$$
$$\mathrm{net}_u = \min_{u' \in U_1}(W(u', u)(o_{u'})),$$

   für $u \in U_3$*:*

$$\mathrm{NET}_u : ([0, 1] \times \mathcal{F}(\mathbb{R}))^{U_2} \to \mathcal{F}(\mathbb{R}),$$
$$\mathrm{net}_u : \mathbb{R} \to [0, 1],$$
$$\mathrm{net}_u(x) = \max_{u' \in U_2}(\min(o_{u'}, W(u', u)(x))),$$

6. $\mathrm{ex} : U_1 \to I\!R$, *ordnet jeder Eingabeeinheit* $u \in U_1$ *ihre externe Eingabe zu* $\mathrm{ex}(u) = \mathrm{ex}_u$. ex *ist für* U_2 *und* U_3 *nicht definiert.*

Wie aus der obigen Definition ersichtlich, entsprechen die Abläufe innerhalb des Neuronalen Fuzzy–Reglers denen eines vorwärtsbetriebenen Neuronalen Netzes. Die Eingabeeinheiten repräsentieren in ihren Aktivierungen die scharfen Werte der Meßgrößen und reichen sie an die mit ihnen verbundenen Regeleinheiten weiter. Die Propagierungsfunktion der Regeleinheiten ermittelt die Zugehörigkeitswerte zu den Fuzzy–Mengen der Verbindungen zwischen Eingabe- und Regelschicht und bestimmt mittels einer t–Norm (min–Operator) den Erfüllungsgrad der jeweiligen Regelprämisse. Dieser Wert repräsentiert die Aktivierung der Regeleinheit.

Die Regeleinheiten geben ihre Aktivierung weiter an die Ausgabeschicht. Deren Propagierungsfunktion verknüpft den Erfüllungsgrad einer Regeleinheit mit der Fuzzy–Menge der jeweiligen Verbindung zur Ausgabeschicht. Dies entspricht der Ermittlung der Ausgabe–Fuzzy–Menge einer Regel. Schließlich werden die Einzelausgaben der Regeln zur Gesamtausgabe zusammengefaßt.

Unter Zuhilfenahme von Eingabewerten und eines sich daraus ergebenden Fuzzy–Fehlermaßes kann das Verhalten von NEFCON optimiert werden indem sowohl Gewichte der Verbindungen zwischen den Schichten (also die Fuzzy–Mengen der Prämissen und der Konklusionen) als auch die Verbindungen (also die Regeln selbst) verändert werden mit dem Ziel, den Optimalzustand des zu regelnden Systems zu erreichen.

Das oben vorgestellte Modell läßt sich erst dann als ein Fuzzy–Regler, also als ein NEFCON–System interpretieren, wenn es zusätzlich im wesentlichen noch zwei Einschränkungen erfüllt (siehe [Nauck 1993]):

- gleiche linguistische Werte einer Variablen werden nur durch eine Fuzzy–Menge repräsentiert (gekoppelte Verbindungen)
- es gibt keine zwei Regeln mit identischen Prämissen.

Diese Überlegungen führen zu folgender formalen Definition (siehe [Nauck 1993]):

Definition 1.2 *Gegeben sei ein dynamisches System* S *mit* n *Meßgrößen und einer Stellgröße. Im Zusammenhang mit der Regelung von* S *seien* k *linguistische*

Regeln bekannt. Ein Fuzzy–Perceptron ist genau dann ein NEFCON–System, wenn es folgende Einschränkungen erfüllt:

1. $U_1 = \{\xi_1, \ldots, \xi_n\}$, $U_2 = \{R_1, \ldots, R_k\}$, $U_3 = \{\eta\}$.
2. *Jede Verbindung zwischen Einheiten* $\xi_i \in U_1$ *und* R_r *wird mit einem linguistischen Term* $A_{j_r}^{(i)}$ $(j_r \in \{1, \ldots, p_i\})$ *benannt.*
3. *Jede Verbindung zwischen Einheiten* $R_r \in U_2$ *und der Ausgabeeinheit* η *wird mit einem linguistischen Term* B_{j_r} $(j_r \in \{1, \ldots, q\})$ *benannt.*
4. *Verbindungen, die von denselben Eingabeeinheiten wegführen und gleiche Benennungen besitzen, tragen zu jeder Zeit gleiche Gewichte.*

   Die zur Ausgabeeinheit führenden Gewichte, die gleiche Benennungen besitzen, tragen auch gleiche Gewichte.
5. $L_{u,v}$ *bezeichne die Benennung zwischen den Einheiten* u *und* v.

$$(\forall v, v' \in U_2)((\forall u \in U_1) L_{u,v} = L_{u,v'}) \Longrightarrow v = v'.$$

Ein NEFCON–System wird also wie folgt konstruiert:

1. Für jede Eingangsgröße ξ_i $(i = 1, \ldots, n)$ wird eine Verarbeitungseinheit mit gleicher Bezeichnung in der Eingabeschicht angelegt.
2. Für die Stellgröße wird die einzige Ausgabeeinheit η angelegt.
3. Für jede Kontrollregel R_r wird eine innere Einheit (Regeleinheit) gleicher Bezeichnung angelegt.
4. Jede Regeleinheit R_r wird gemäß ihrer korrespondierenden Kontrollregel mit den entsprechenden Eingabeeinheiten und der Ausgabeeinheit verbunden. Als Verbindungsgewichte sind die Fuzzy–Mengen zu wählen, die den linguistischen Termen der Prämissen und der Konklusion der Kontrollregel R_r entsprechen.
5. Es sind eine t–Norm, t–Conorm und eine Defuzzifizierungsstrategie geeignet zu wählen (s.u.).

1.2 Fuzzy–Fehler–Propagation

Die Interpretation eines NEFCON–Systems als dreischichtiges Perceptron (siehe Definition 1.1) erlaubt die Anwendung neuronaler Konzepte, vor allem von Lernverfahren aus dem Bereich Neuronaler Netze.

Die Lernalgorithmen für das NEFCON–Modell verwenden die Methoden des reinforcement Learning (siehe [Nauck 1993]) mit einer freien Lernaufgabe unter Verzicht auf das adaptive Kritikerelement, das durch ein Verstärkungssignal ersetzt wird, das auf einem Fuzzy–Fehler–Maß basierend definiert wird.

Beim NEFCON–Modell werden prinzipiell zwei Teile des Lernalgorithmus unterschieden. Der erste, die Fuzzy–Fehler–Propagation befaßt sich mit der Adaption von Fuzzy–Mengen, also der Optimierung der Verbindungsgewichte zwischen den Schichten bei gegebener Regelbasis. Der zweite bewirkt das Erlernen der Regelbasis.

1.3 Fuzzy–Fehler–Maß

Das Ziel der Regelung ist, das zu regelnde System in einen optimalen Zustand zu steuern. Dieser optimale Zustand sei durch den Vektor von Meßgrößen $X_{\mathrm{opt}} = \left(x_1^{(\mathrm{opt})}, \ldots, x_n^{(\mathrm{opt})}\right)$ repräsentiert.

Die Güte des Regelverhaltens mißt sich daran, wie nahe der aktuelle Meßwertevektor $X = (x_1, \ldots, x_n)$ am Optimalzustand liegt.

Der Zustand des Systems wird also als optimal bezeichnet, wenn $X = X_{\mathrm{opt}}$.

Gewöhnlich bezeichnet man einen Zustand eines Systems jedoch auch dann als *gut*, wenn der Wert X_{opt} nur ungefähr erreicht worden ist. Es scheint deshalb angemessen, die Güte des Systemzustands als eine unscharfe Größe zu interpretieren und sie mit Hilfe von Fuzzy–Mengen zu beschreiben.

Bei der Festlegung eines als gut zu bezeichnenden Zustandes kann man folgende Fälle unterscheiden:

1. Die Meßwerte X entsprechen in etwa X_{opt}. Für diesen Fall wird die *optimale Fuzzy–Güte* G_{opt} wie folgt definiert:
$$G_{\mathrm{opt}} := \min\left\{\mu_{\mathrm{opt}}^{(1)}(x_1), \ldots, \mu_{\mathrm{opt}}^{(n)}(x_n)\right\},$$

wobei $\mu_{\text{opt}}^{(i)}$ $(i = 1, \ldots, n)$ der unscharfe optimale Wert für die i-te Meßgröße und $\mu_{\text{opt}}^{(i)}(x_i)$ der Zugehörigkeitsgrad der gemessenen Größe x_i zum unscharfen optimalen Wert ist.

2. Die Meßwerte weichen vom Optimum ab, kompensieren sich aber dahingehend, daß das System einem Optimalzustand entgegenstrebt. Bei s solchen kompensatorischen Situationen $\mu_{\text{komp}}^{(i)}$ $(i = 1, \ldots, s)$ wird die kompensatorische Fuzzy–Güte als

$$G_{\text{komp}} := \min\left\{\mu_{\text{komp}}^{(1)}(x_1, \ldots, x_n), \ldots, \mu_{\text{komp}}^{(s)}(x_1, \ldots, x_n)\right\}$$

definiert.

Die System–Fuzzy–Güte G wird aus diesen beiden Maßen definiert als

$$G := g(G_{\text{opt}}, G_{\text{komp}})$$

Die Funktion g ist in Abhängigkeit vom zu regelnden System zu wählen.

Der Fuzzy–Fehler kann nun als

$$E := 1 - G$$

definiert werden.

Zusätzlich zum Fuzzy–Fehler benötigt der Lernalgorithmus für einen gegebenen Systemzustand das Vorzeichen des optimalen Stellwertes $\eta_{\text{opt}} \in Y$ (durch geeignete Transformation des Ausgabewertebereiches kann erreicht werden, daß beim optimalen Systemzustand der Stellwert $o_{\eta_{\text{opt}}} = 0$ ist). Zusammen mit dem Fuzzy–Fehler definiert man den *erweiterten Fuzzy–Fehler* als

$$E^* := E \cdot \text{sgn}(\eta_{\text{opt}}).$$

Mit Hilfe von E^* ist der Lernalgorithmus nun in der Lage, die letzte Regelaktion, die zu den aktuellen Meßwerten geführt hat, zu beurteilen, und er weiß gleichzeitig, welches Vorzeichen der aktuelle optimale Stellwert haben muß.

Also bedeutet z.B. $E^* > 0$, daß bei den aktuellen Meßwerten ein Stellwert größer Null optimale wäre, und daß der aus dem vorangegangenen Meßwerten berechnete Stellwert nicht groß genug war.

Ein Lernalgorithmus, der auf dem oben definierten, erweiterten Fuzzy–Fehler basiert, heißt Fuzzy–Fehler–Propagation und entspricht dem verstärkenden Lernen ohne die Verwendung eines adaptiven Kritikerelementes.

Der Fehler wird als ein Verstärkungssignal angegeben. Er wird rückwärts durch das Netzwerk propagiert, beginnend bei den Ausgabeeinheiten hin zu den Eingabeeinheiten und von den Einheiten lokal zur Adaption der Fuzzy–Mengen benutzt.

In Abhängigkeit vom Fuzzy–Fehler muß für alle an der Regelaktion beteiligten Fuzzy–Mengen ermittelt werden, ob und wie sie verändert werden sollen. Es werden nur solche Fuzzy–Mengen verändert, die mit Regeleinheiten, deren Aktivierung ungleich Null ist, in Verbindung stehen. Also erfahren Regeln mit dem Erfüllungsgrad 0 keine Veränderung.

Es ist klar, daß Regeleinheiten mit hoher Aktivierung einen hohen Anteil an der Ausgabe des Reglers haben. Entsprechend müssen also die mit dieser Regeleinheit in Verbindung stehenden Fuzzy–Mengen verändert werden.

Zusätzlich benötigt der Lernalgorithmus noch den Beitrag jeder einzelnen Regel an der Entstehung des Steuerwertes. Zusammen mit der Kenntnis des Vorzeichens des optimalen Stellwertes η_{opt} kann der Lernalgorithmus entscheiden, ob er eine Regel *belohnen* oder *bestrafen* soll. Offensichtlich werden Regeln belohnt, deren Beitrag zum Steuerwert das Gleiche Vorzeichen wie η_{opt} haben, anderenfalls werden sie bestraft.

Mit der “Belohnung” soll erreicht werden, daß in einer ähnlichen Situation eine Regel einen stärkeren Beitrag zur Regelaktion leistet und sie somit positiv beeinflußt. Dies kann einerseits durch Erhöhung der Regelaktivierung (Erhöhung des Treffergrades der Fuzzy–Menge der Prämisse der Regel) oder auch durch Erhöhung des Ausgabewertes der Regel geschehen. Entsprechend bedeutet “Bestrafung”, daß der Anteil der Regel an der Gesamtausgabe bei einer ähnlichen Situation gesenkt wird.

Neben der Möglichkeit der Angabe des erweiterten Fuzzy–Fehlers auf Grundlage von G_{opt}, G_{komp} und $sgn(\eta_{\text{opt}})$ kann dieser auch in Form von linguistischen Regeln spezifiziert werden. Dazu muß eine Partitionierung des Fehlerintervalls $[-1, 1]$ angegeben werden. Die Auswertung des Fuzzy–Fehlers kann dann durch einen einfachen Fuzzy–Regler (je nach Interpretation Mamdani- oder Gödel–Regler) vorgenommen werden. Das Vorzeichen des Fehlers entspricht dann $sgn(\eta_{\text{opt}})$ und der Betrag dem Betrag von E. Diese Vorgehensweise wird in den Abbildungen 1.3 und

1.2 verdeutlicht, wobei die Terme P,Z,N für Negativ, Positiv und Zero stehen.

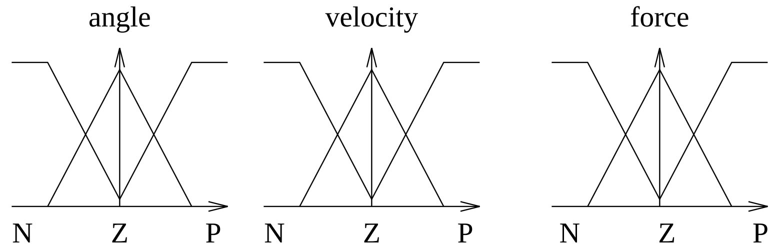

Abbildung 1.2: Fehler–Wissensbasis für das einfache inverse Pendel: Partitionierung der linguistischen Variablen

velocity \ angle	N	Z	P
N	N	P	Z
Z	N	Z	P
P	Z	P	P

Abbildung 1.3: Fehler–Wissensbasis für das einfache inverse Pendel: die Regelbasis

Die folgenden beiden Abschnitte werden sich mit der Konkretisierung des Lernverfahrens befassen. Zuerst wird das ursprüngliche NEFCON Lernverfahren vorgestellt. Weiter werden Möglichkeiten aufgezeigt, seine Einschränkungen (nur lineare Fuzzy–Mengen in den Regelkonklusionen, Optimierung nur durch die Veränderung der Breite der Fuzzy–Mengen) aufzuheben.

1.4 Das ursprüngliche Optimierungsverfahren für Fuzzy–Mengen

Wie schon im vorigen Abschnitt erwähnt, ist ein Lernalgorithmus aufgrund des erweiterten Fuzzy–Fehlers E^* in der Lage, Optimierungen an der Wissensbasis eines Neuronalen Fuzzy–Reglers vorzunehmen. In diesem und dem folgenden Abschnitt wird der Lernalgorithmus zur Adaption von Fuzzy–Mengen behandelt. Hierzu sei vorausgesetzt, daß ein Satz adäquat modellierter Regeln vorhanden ist und das Fehlverhalten des Reglers durch eine nicht–optimale Modellierung der linguistischen Terme verursacht wird.

Für das Lernverfahren ist es wichtig zu wissen, welchen Beitrag jede einzelne Regel am Zustandekommen des Stellwertes hatte. Dieser Wert ist nicht exakt zu bestimmen, wenn die Fuzzy–Mengen der Konklusionen erst zu einer Gesamtausgabe–Fuzzy–Menge akkumuliert werden, und diese schließlich die defuzzifiziert wird. Um dieses Problem zu umgehen, wurde im NEFCON–Modell gefordert, daß die Fuzzy–Mengen der Regelkonklusionen nur monotone Zugehörigkeitsfunktionen haben dürfen. Aufgrund der Existenz der Umkehrfunktion liefert jede Regel R_r in Abhängigkeit ihres Erfüllungsgrades o_r den scharfen Wert $y_r = \nu^{-1}(o_r)$.

Unter der obigen Voraussetzung kann entschieden werden, ob eine bestimmte Regel einen positiven oder einen negativen Einfluß auf die Regelung hatte. Um den Stellwert für einen bestimmten Systemzustand zu erhöhen, werden alle Regeln, deren Beitrag zum Ausgabewert positiv war "belohnt", und diejenigen deren Beitrag negativ war "bestraft". Soll der Stellwert in einem Zustand verringert werden, so muß der umgekehrte Weg beschritten werden. Die Strategie besteht also darin, Regeln, die das System einem optimalen Zustand näherbringen, zu verstärken. Bestrafung von Regeln tritt nur bei Überschwingverhalten ein.

Beim Belohnen wird versucht, eine Regel dahingehend zu ändern, daß sich erstens ihr Erfüllungsgrad o_r bei einem ähnlichen Systemzustand erhöht, und zweitens sich der Betrag ihrer Ausgabe $\nu^{-1}(o_r)$ bei gleichem Erfüllungsgrad steigert. Beim Bestrafen wird umgekehrt vorgegangen. Die Abbildung 1.4 verdeutlicht das Vorgehen.

Aus der Abbildung 1.4 entnimmt man, daß die Erhöhung des Erfüllungsgrades der Prämisse durch die Verbreiterung ihrer Fuzzy–Mengen vorgenommen werden kann. Die Erhöhung der Regelausgabe wird dadurch erreicht, daß man die monotone Fuzzy–Menge der Regelkonklusion steiler macht.

Die Stärke der Veränderung der Fuzzy–Mengen einer Regel R_r hängt von dem individuellen *Fuzzy–Regel–Fehler* dieser Regel ab. In diesen geht der individuelle Beitrag der Regel zum Stellwert mit ein. Durch die Forderung, nur monotone Fuzzy–Mengen in der Regelkonklusionen zu verwenden, ist dieser individuelle Beitrag gerade aus dem Erfüllungsgrad der Regel o_{R_r}. Dies führt zur Definition des Fuzzy–Regel–Fehlers als:

$$E_{R_r} := o_{R_r} \cdot E^*$$

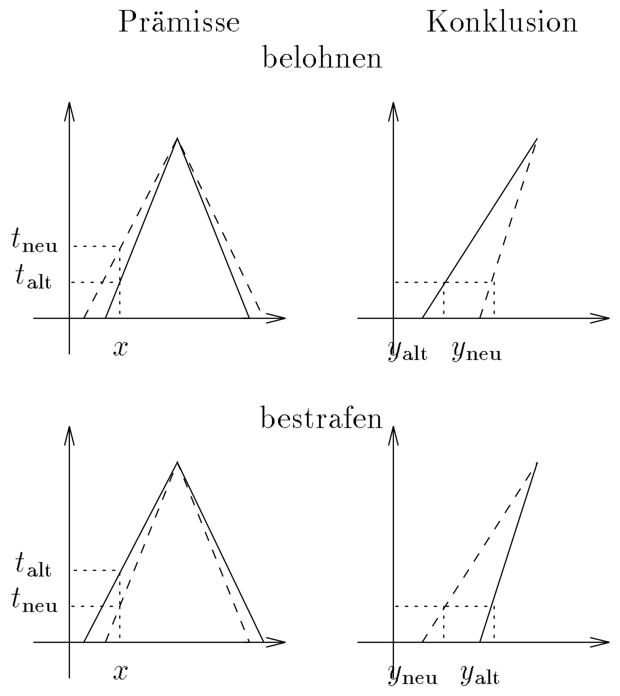

Abbildung 1.4: Belohnen und Bestrafen von Fuzzy–Mengen

Die obigen Überlegungen führen also zu folgendem Verfahren zur Optimierung von Fuzzy–Mengen durch Veränderung ihrer Breite:

Definition 1.3 *Gegeben ein System S mit n Meßgrößen $\xi_i \in X$ und einer Stellgröße $\eta \in Y$ und ein NEFCON–System mit k Regeleinheiten. Der Algorithmus zur Adaption der Fuzzy–Mengen umfaßt folgende Aktionen, die solange wiederholt werden, bis ein Abbruchkriterium erfüllt ist:*

1. *Bestimmung der NEFCON–Ausgabe o_η. Anwendung von o_η auf S und Bestimmung neuer Meßwerte:*

$$o_\eta := \frac{\sum\limits_{i=r}^{k} o_r \cdot \nu_{j_r}^{-1}(o_r)}{\sum\limits_{i=r}^{k} o_r}.$$

2. *Bestimmung des Fuzzy–Fehlers E aus dem neuen Systemzustand.*

3. *Bestimmung von $sgn(\eta_{\text{opt}})$ und des erweiterten Fuzzy–Fehlers $E^* = E \cdot sgn(\eta_{\text{opt}})$ für den neuen Systemzustand.*

4. *Ermittlung des Anteils y_r jeder Regel an der Ausgabe der Ausgabeeinheit:*

$$\begin{aligned} y_r &:= \nu_{j_r}^{-1}(o_r) \\ &= d_{j_r} - o_r \cdot (d_{j_r} - e_{j_r}) \end{aligned}$$

   Bestimmung des Fehlersignals E_{R_r} für jede Regeleinheit R_r ($r \in \{1,\ldots,k\}$) mit

$$\begin{aligned} E_{R_r} &:= \begin{cases} o_{R_r} \cdot E & \text{falls } sgn(y_r) = sgn(\eta_{\text{opt}}) \\ -o_{R_r} \cdot E & \text{sonst} \end{cases} \\ &= sgn(y_r) \cdot o_{R_r} \cdot E^*. \end{aligned}$$

5. *Bestimmung Bestimmung der Parameteränderung für die Zugehörigkeitsfunktionen ν_{j_r} ($j_r \in \{1,\ldots,q\},\ r \in \{1,\ldots,k\}$):*

$$\triangle d_{j_r} := -c \cdot E_{R_r} \cdot |d_{j_r} - e_{j_r}|,$$

   mit einer Lernrate $c > 0$.

   Durchführung der Änderung unter Erfüllung gewisser Randbedingungen.

6. *Bestimmung Bestimmung der Parameteränderung für die Zugehörigkeitsfunktionen $\mu_{j_r}^{(i)}$ ($i \in \{1,\ldots,n\},\ j_r \in \{1,\ldots,p_i\},\ r \in \{1,\ldots,k\}$):*

$$\begin{aligned} \triangle a_{j_r}^{(i)} &:= c \cdot E_{R_r} \cdot (b_{j_r}^{(i)} - a_{j_r}^{(i)}), \\ \triangle c_{j_r}^{(i)} &:= -c \cdot E_{R_r} \cdot (c_{j_r}^{(i)} - b_{j_r}^{(i)}). \end{aligned}$$

   mit einer Lernrate $c > 0$.

   Durchführung der Änderung unter Erfüllung gewisser Randbedingungen.

Dabei dienen die Fuzzy–Mengen $\mu_j^{(i)}$ ($i \in \{1,\ldots,n\}$) ($j \in \{1,\ldots,p_i\}$) und ν_j ($j \in \{1,\ldots,q\}$) zur Repräsentation der linguistischen Terme der Meßgrößen $\xi_i \in X_i \subseteq$

$I\!R$ bzw. der Stellgröße $\eta \in Y \subseteq I\!R$ und sind wie folgt definiert:

$$
\begin{aligned}
\mu_j^{(i)}(x) \quad &:= \quad \begin{cases} \dfrac{x - a_j^{(i)}}{b_j^{(i)} - a_j^{(i)}} & \textit{falls } x \in [a_j^{(i)}, b_j^{(i)}], \\ \dfrac{c_j^{(i)} - x}{c_j^{(i)} - b_j^{(i)}} & \textit{falls } x \in [a_j^{(i)}, b_j^{(i)}], \\ 0 & \text{sonst}, \end{cases} \\
& \quad\ \text{mit } a_j^{(i)}, b_j^{(i)}, c_j^{(i)} \in I\!R, \quad a_j^{(i)} \leq b_j^{(i)} \leq c_j^{(i)}, \\
\nu_j(y) \quad &:= \quad \begin{cases} \dfrac{d_j - y}{d_j - e_j} & \text{falls} \quad \begin{array}{r} (y \in [d_j, e_j] \wedge d_j \leq e_j) \\ \vee \ (y \in [e_j, d_j] \wedge d_j > e_j) \end{array}, \\ 0 & \text{sonst}, \end{cases} \\
& \quad\ \text{mit } d_j, e_j \in I\!R.
\end{aligned}
$$

Damit gelten $\mu_j^{(i)}(a_j^{(i)}) = 0$, $\mu_j^{(i)}(b_j^{(i)}) = 1$, $\mu_j^{(i)}(c_j^{(i)}) = 0$, $\nu_j(d_j) = 0$ und $\nu_j(e_j) = 1$.

Bemerkung

Detlef Nauck benutzt in seiner Dissertationsschrift das Fehlersignal als ein *negatives* Verstärkungssignal. In dieser Arbeit benutze ich ein positives Verstärkungssignal, bei dem sich jeweils nur das Vorzeichen umkehrt.

Die Akkumulation des Steuerwertes entspricht bei diesem Verfahren der Defuzzifizierung nach dem MOM–Verfahren (siehe Abschnitt 2.3). Die Beschränkung auf monotone Fuzzy–Mengen in den Regelkonklusionen läßt es nicht sinnvoll erscheinen, andere Defuzzifikationsstrategien einzusetzen. Dies wäre erst dann von Bedeutung, wenn man auch dreieckförmige Fuzzy–Mengen in den Regelkonklusionen zulassen würde.

Für diesen Fall kommen zusätzliche Probleme auf. Betrachtet man das COA–Verfahren, bei dem die gesamte Ausgabe–Fuzzy–Menge zuerst akkumuliert und danach defuzzifiziert wird, verliert man die wichtige Information über den exakten Beitrag einer Regel an der Entstehung des Steuerwertes. Es ist auch schwer zu bestimmen, was für einen Effekt die Veränderung der Breite der Konklusions–Fuzzy–Menge auf den Gesamtsteuerwert haben wird. Bei symmetrischen Fuzzy–Mengen ist er in vielen Fällen Null. Das zweite Problem besteht, bzw. verstärkt sich sogar, falls man die Defuzzifizierung der Akkumulation vorzieht (das so genannte *average COA*, siehe Abschnitt 2.3).

Ein Ansatz, diese Probleme in Griff zu bekommen wird im folgenden Kapitel beschrieben. Dort wird ein neuer Weg der Optimierung von Fuzzy–Mengen durch Verschiebung vorgestellt.

Kapitel 2

Ein neuer Optimierungsalgorithmus für Fuzzy–Mengen

Das ursprüngliche NEFCON–Modell weist noch wesentliche Einschränkungen bezüglich der Wahl der Formen von Fuzzy–Mengen auf und damit zusammenhängend auch Einschränkungen der Defuzzifizierungsmethoden. In diesem Abschnitt wird ein neues Lernverfahren vorgestellt, das diese Beschränkungen aufhebt und die Benutzung von Dreiecks–Fuzzy–Mengen in den Regelkonklusionen erlaubt. Als Defuzzifizierungsverfahren wird COA oder 'average COA' (siehe Abschnitt 2.3) benutzt. Der Lernalgorithmus optimiert die Fuzzy–Mengen durch Verschiebung.

2.1 Optimierungsalgorithmus druch Veränderung der Lage der Fuzzy–Mengen

Im Unterschied zur ursprünglichen Idee der Fuzzy–Fehler–Propagation wird jetzt nicht nur der erweiterte Fuzzy–Fehler selbst, sondern auch dessen Änderung berücksichtigt. Sei E der erweiterte Fuzzy–Fehler vor einer Regelaktion und E' derjenige, der aus den der Regelaktion folgenden Meßwerten errechnete, erweiterte Fuzzy–Fehler. Aus diesen beiden Werten wird der Parameter $\tau \in \{-1, 0, 1\}$ berechnet, der angibt inwieweit die Regelaktion den Fehler verändert hat, also welche Tendenz der Fehler aufweist.

Definition 2.1 *Die* Fehlertendenz $\tau \in \{-1, 0, 1\}$ *ist definiert als*

$$\tau = \begin{cases} 1 & \text{falls} \quad (|E'| \geq |E|) \ \wedge \ (E' \cdot E \geq 0) \\ 0 & \text{falls} \quad (|E'| < |E|) \ \wedge \ (E' \cdot E \geq 0) \\ -1 & \text{falls} \quad E' \cdot E < 0 \end{cases}$$

Also nimmt τ den Wert 1 an, falls der Fehler ansteigt und sein Vorzeichen sich nicht ändert und -1, falls das System übersteuert. Falls der Fehler sinkt und sich sein Vorzeichen nicht ändert, ist $\tau = 0$.

In Abhängigkeit von dem τ–Wert werden die Positionen der Fuzzy–Mengen verändert. Dabei wird bei Systemzuständen, die dem Optimalzustand entgegenstreben, also $\tau = 0$ keine Änderung vorgenommen.

Falls $\tau = 1$ ist, also falls der Fehler absolut wächst und sein Vorzeichen sich nicht ändert, muß der Steuerwert insgesamt größer werden, also die Ausgabewerte der Regelkonklusionen in Richtung des optimalen Stellwertes η_{opt} mit $\text{sgn}(\eta_{\text{opt}}) = \text{sgn}(E)$ erhöht werden (siehe Abbildung 2.1).

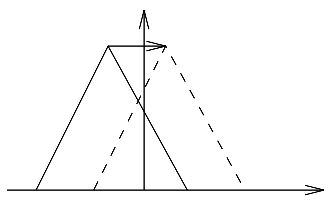

Abbildung 2.1: Verschieben einer Konklusions–Fuzzy–Menge bei $\tau = 1$ und $\text{sgn}(\eta_{\text{opt}}) = 1$

Die Regelprämissen werden insoweit verändert, daß der Erfüllungsgrad der Regeln mit einem Regelausgabewert $y_r = \nu_r^-(o_r)$, der die Gesamtausgabe o_η erhöht, gesteigert (siehe Abbildung 2.2) und derjenigen mit einem Regelausgabewert, der die Gesamtausgabe y senkt, gesenkt wird (siehe Abbildung 2.3). Die Steigerung oder Senkung des Erfüllungsgrades wird durch Verschieben der gesamten Fuzzy–Menge zum Meßwert hin oder vom Meßwert weg bewerkstelligt.

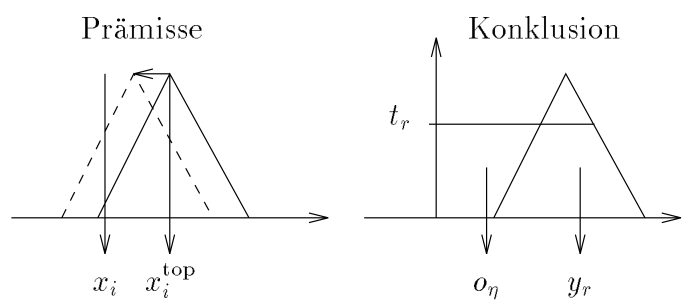

Abbildung 2.2: Verschieben einer Prämissen–Fuzzy–Menge bei $\tau = 1$ und einem verstärkendem Einfluß der Konklusion

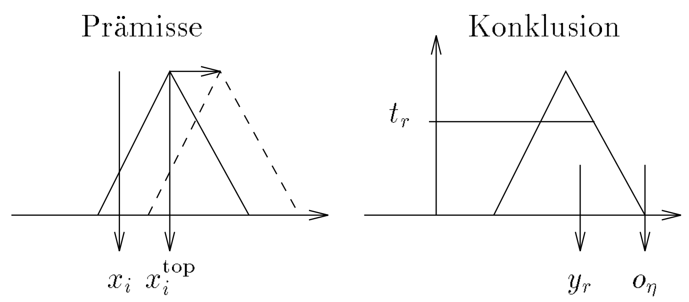

Abbildung 2.3: Verschieben einer Prämissen–Fuzzy–Menge bei $\tau = 1$ und einem abschwächenden Einfluß der Konklusion

$\nu_r^-(o_r)$ ist eine Pseudoinverse der Zugehörigkeitsfunktion ν_r beim Regelerfüllungsgrad o_r. Sie errechnet sich aus der Defuzzifizierung der einzelnen Regelausgaben (siehe Abschnitt 2.3).

Falls $\tau = -1$, d.h. das System übersteuert, muß der Steuerwert zu hoch gewesen sein. In diesem Fall müssen die Regelausgabewerte $y_r = \nu_r^-(o_r)$ gesenkt werden, um damit die Gesamtausgabe abzusenken (siehe Abbildung 2.4).

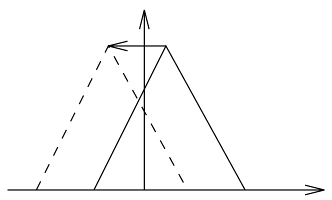

Abbildung 2.4: Verschieben einer Konklusions–Fuzzy–Menge, falls $\tau = -1$ und $\mathrm{sgn}(E) = 1$

Die Erfüllungsgrade der Prämissen müssen in diesem Fall vergrößert werden, falls die zu ihnen gehörende Konklusion einen abschwächenden Einfluß auf den Gesamtausgabewert hatte (siehe Abbildung 2.5).

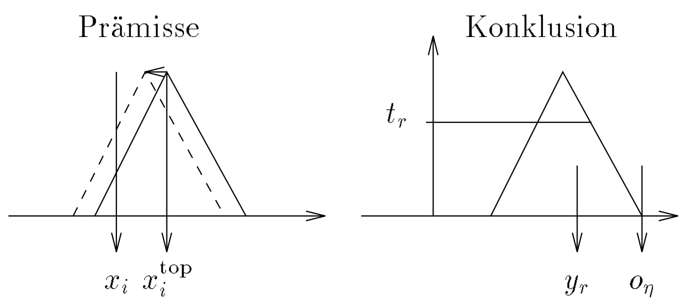

Abbildung 2.5: Verschieben einer Prämissen–Fuzzy–Menge, wenn $\tau = -1$und ihre Konklusions–Fuzzy–Menge hat einen abschwächendenEinfluß

Entsprechend müssen sie verkleinert werden, wenn die entsprechende Konklusions–Fuzzy–Menge einen verstärkenden Einfluß auf die Gesamtausgabe hatte (siehe Abbildung 2.6).

Die folgenden Definitionen verdeutlichen das Vorgehen.

Definition 2.2 *Gegeben ein System S mit n Meßgrößen $\xi_i \in X$ und einer Stellgröße $\eta \in Y$ und ein NEFCON–System mit k Regeleinheiten. Der Algorithmus*

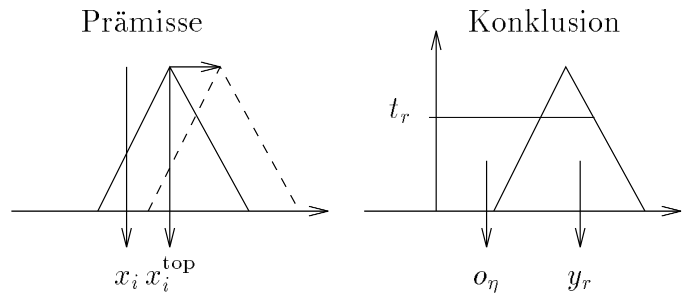

Abbildung 2.6: Verschieben einer Prämissen–Fuzzy–Menge, wenn $\tau = -1$ und ihre Konklusions–Fuzzy–Menge hat einen verstärkenden Einfluß

zur Adaption der Fuzzy–Mengen umfaßt folgende Aktionen, die solange wiederholt werden, bis ein Abbruchkriterium erfüllt ist:

1. *Bestimme den erweiterten Fuzzy–Fehler E^* und $sgn(\eta_{\text{opt}})$ aufgrund der Meßwerte.*

2. *Bestimme den absolute Anteil y_r und den relativen Anteil κ_r an der Ausgabe der Ausgabeeinheit:*

$$y_r := \nu_{j_r}^-(o_r),$$
$$\kappa_r := \tfrac{o_\eta - y_r}{|Y|},$$

   wobei

$$\nu_r^-(o_r) := \frac{\sum\limits_{\substack{y \in Y \\ \nu_{j_r}(y) > 0}} y \cdot \min(o_r, \nu_{j_r}(y))}{\sum\limits_{\substack{y \in Y \\ \nu_{j_r}(y) > 0}} \min(o_r, \nu_{j_r}(y))}.$$

3. *Bestimme den NEFCON–Ausgabewert o_η und wende ihn auf S an. Bestimme neue Meßwerte.*

4. *Ermittle den neuen erweiterten Fuzzy–Fehler $E^{*\prime}$ und die Fehlertendenz τ aufgrund der neuen Meßwerte.*

5. *Bestimme die Parameteränderung für Fuzzy–Mengen ν_{j_r}:*

$$\delta_{\text{conc}}^{j_r} := \tau \cdot sgn(\eta_{\text{opt}}) \cdot |E^{*'}| \cdot o_r \cdot |o_\eta - y_r| \cdot c$$

mit einer Lernrate $c > 0$.

Durchführung der Verschiebung unter Erfüllung gewisser Randbedingungen.

6. *Bestimme die Parameteränderung für Fuzzy–Mengen $\mu(i)_{j_r}$:*

$$\delta_{\text{pre}}^{j_r,i} := \tau \cdot sgn(\eta_{\text{opt}}) \cdot (x_i - x_i^{j_r,\text{top}}) \cdot |E^{*'}| \cdot \kappa_r \cdot c$$

mit einer Lernrate $c > 0$.

Durchführung der Verschiebung unter Erfüllung gewisser Randbedingungen.

Die Simulationsumgebung NEFCON–I (siehe Kapitel 4) bietet drei Verschiebemethoden, die jeweils verschiedene Restriktionen beachten. Die einfachste von ihnen beachtet keine Restriktionen, außer daß Fuzzy–Mengen nicht aus dem Definitionsbereich hinausgeschoben werden. Sie ist so realisiert, daß Fuzzy–Mengen um den Wert δ in Richtung der x–Achse durch einfaches Aufaddieren von δ verschoben werden. Die y–Werte bleiben unverändert. Die zweite Methode erhält die Schnitte der benachbarten Fuzzy–Mengen und ist somit sehr restriktiv. Die dritte Methode verhindert, daß Fuzzy–Mengen einander überholen und funktioniert ansonsten wie die erste.

Der Lernalgorithmus ist vor allem mit der einfachsten Version des Verschiebeverfahrens getestet worden, und verhielt sich dabei am erfolgreichsten. Die restriktiven Verfahren waren vor allem in der Experimentierphase von Bedeutung.

Ein noch ungelöstes Problem ist ein mögliches Zusammenspiel der beiden vorgestellten Lernalgorithmen. Bei dem jetzigen Stand sollten sie nicht gleichzeitig verwendet werden (auch wenn die Simulationsumgebung NEFCON–I dies zuläßt). Jeder Algorithmus liefert für sich bessere Ergebnisse als in Zusammenarbeit mit dem anderen. Experimente haben gezeigt, daß vor allem dann, wenn die beiden Algorithmen gleichzeitig auf die Konklusionen angewendet werden, sie sich einander eher behindern.

2.2 Das Lernverfahren am Beispiel

Nachfolgend wird ein Beispiel zur Demonstration des Lernverfahrens anhand des Stabbalanceproblems durchgeführt. Die notwendigen Dateien für dieses Beispiel liegen dem Programmpaket bei.

Als Fehler–Wissensbasis wird in diesem Beispiel die in den Abbildungen 1.2 und 1.3 vorgestellte benutzt. Als Lernfaktor, wird $c = 0.1$ genommen. Das Pendel wird immer dann mit einem zufälligen Winkel angestoßen, wenn es sich 20 Lernschritte lang in der Nähe des Optimalzustandes befunden hat (Dieses Verhalten läßt sich mit Hilfe der Triggeroption steuern). Das Lernverfahren beginnt mit der Steuer–Wissensbasis, die in den Abbildungen 2.7, 2.8, 2.9 und 2.10 dargestellt werden.

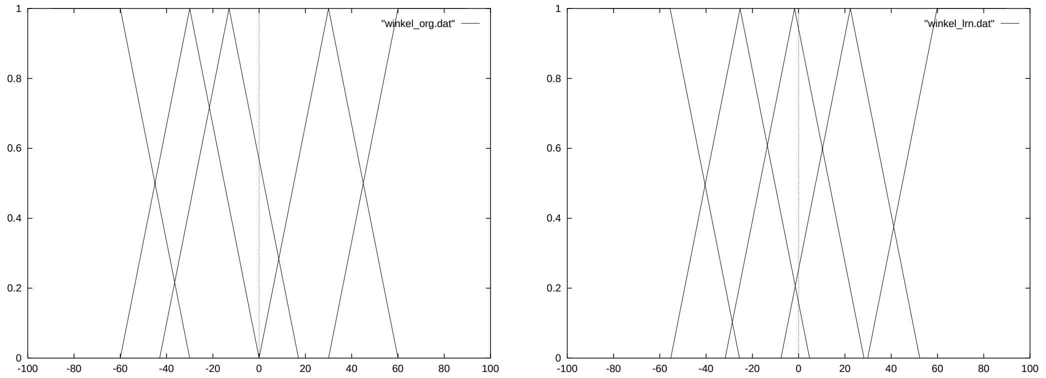

Abbildung 2.7: Partitionierung der linguistischen Eingabe–Variable *winkel* vor dem Start des Lernvorgangs und nach 5000 Lernschritten

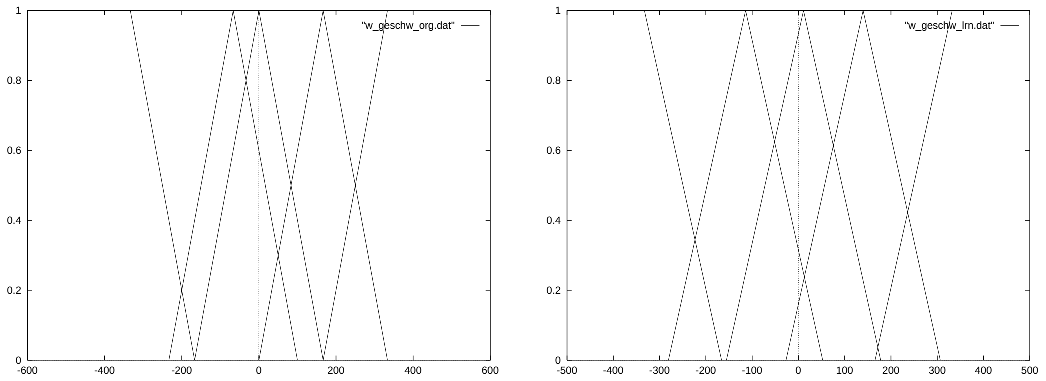

Abbildung 2.8: Partitionierung der linguistischen Eingabe–Variable *w-geschw* vor dem Start des Lernvorgangs und nach 5000 Schritten

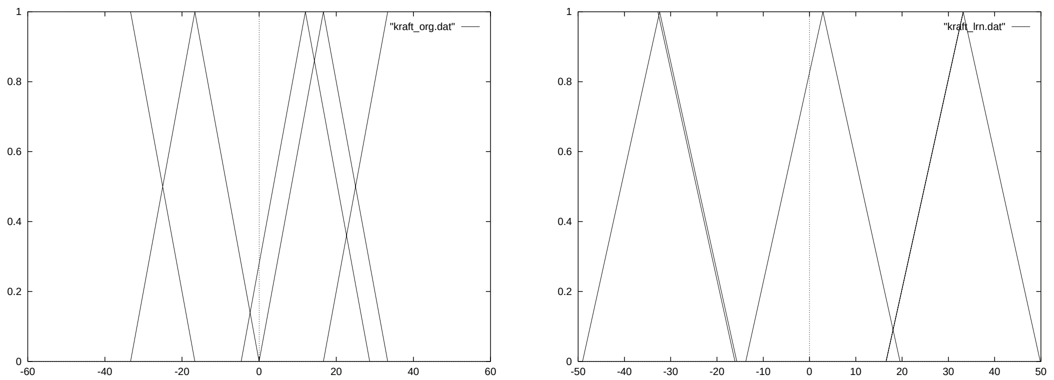

Abbildung 2.9: Partitionierung der linguistischen ausgabe–Variable *kraft* vor dem Start des Lernvorgangs und nach 5000 Schritten

rule table

<winkel>

<w_geschw>	nb	n	z	p	pb
nb		nb	nb		
n	nb		n		
z	nb	n	z	p	pb
p			p		pb
pb			pb	pb	

Abbildung 2.10: Die Regelbasis

Nach 5000 Lernzyklen wurde das Optimierungsverfahren abgebrochen. Die Partitionierung der linguistischen Variablen gibt annähernd die optimale Partitionierung wieder. Der Regler schafft es mit dieser Partitionierung, Winkel im Bereich $[-40°, 40°]$ auszusteuern, obwohl vor dem Start des Lernvorgangs die Regelung

fehlschlug. Man sieht dies recht genau in der Abbildung 2.11, die den Verlauf der Regelfunktion und des Fehlers vor (links) und nach (rechts) dem Lernen darstellt. Bemerkenswert ist, daß die Partitionierung des Ausgabebereichs nur noch de facto drei Fuzzy–Mengen aufweist.

2.3 Defuzzifizierung

Das ursprünglich im NEFCON–Modell verwendete Defuzzifizierungsverfahren war MOM, das sich bei den geforderten monotonen Fuzzy–Mengen in den Regel–Konklusionen sich vereinfacht zu

$$\eta = \frac{\sum\limits_{r=1}^{m} o_r \cdot \nu_r^{-1}(o_r)}{\sum\limits_{r=1}^{m} o_r}.$$

Dabei ist o_r der Erfüllungsgrad der Regel R_r und m die Anzahl der Regeln. Dieses Verfahren läßt sich auch so auffassen, als würde erst defuzzifiziert werden (die Berechnung der $\nu^{-1}(o_r)$ Werte, die wegen der Monotonie existieren), und dann durch Bildung der Gewichteten Summe der Steuerwert akkumuliert werden. Die Vorteile eines solchen Vorgehens sind, daß man den Einfluß einer Regel auf den Steuerwert direkt mit $\nu^{-1}(o_r)$ erhält. Für den Lernalgorithmus ist vor allem das Vorzeichen von $\nu^{-1}(o_r)$ bedeutend.

Zusammen mit der Einführung der dreieckförmigen Fuzzy–Mengen in den Regelkonklusionen mußten neue Defuzzifikationsverfahren implementiert werden. In der Implementierung der Simulationsumgebung NEFCON–I habe ich drei zusätzliche Defuzzifikationsmthoden vorgesehen, die in unterschiedlichen Bereichen Vor- und Nachteile bieten. Diese Verfahren gehen alle von der Idee des COA–Verfahrens aus.

1. Diskrete COA–Methode (Center Of Area)

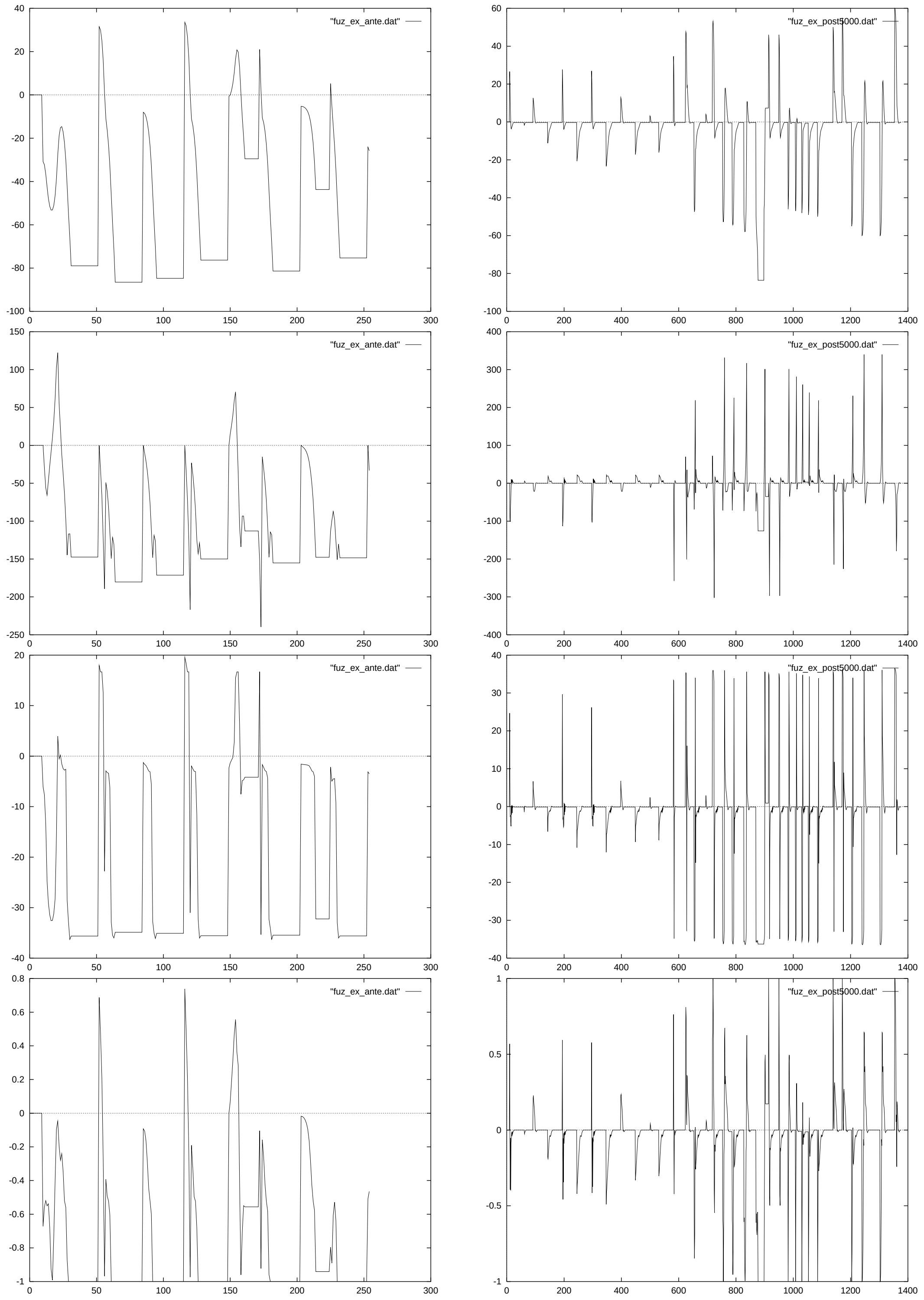

Abbildung 2.11: Verlauf der Meßwerte, des Stellwertes und des erweiterten Fuzzy–Fehlers vor dem Lernen (links) und nach dem Lernen (rechts)

Bei dieser Methode wird als erstes die Ausgabe–Fuzzy–Menge $\nu^* : Y \to [0,1]$ aus den einzelnen Konklusions–Fuzzy–Mengen durch den max–Operator akkumuliert. Der Steuerwert wird dann als

$$\eta = \frac{\sum\limits_{y \in Y} y \cdot \nu^*(y)}{\sum\limits_{y \in Y} \nu^*(y)}$$

berechnet.

Als heuristischer Anteil der Regel R_r am Gesamtsteuerwert wird der Wert

$$\nu_r^-(o_r) := \frac{\sum\limits_{\substack{y \in Y \\ \nu_r(y) > 0}} y \cdot \min(o_r, \nu_r(y))}{\sum\limits_{\substack{y \in Y \\ \nu_r(y) > 0}} \min(o_r, \nu_r(y))}$$

gewählt.

Offensichtlich entspricht dieser Wert dem COA–Wert der Ausgabe–Fuzzy–Menge der Regel R_r. Dieser Wert wird vom Lernalgorithmus als Regelausgabewert y_r verwendet.

Diese Defuzzifizierungsmethode ist immer verwendbar. Trotz der Heuristik bei der Berechnung des Beitrags zum Gesamtsteuerwert ist es auch für die Lernverfahren geeignet. Leider ist die diskrete Berechnung ziemlich langsam. Diese Methode ist vor allem für Optimierung von Fuzzy–Mengen durch Verschieben geeignet. Das Optimierungsverfahren durch Veränderung der Breite müßte zusätzlich noch angepaßt werden.

2. Average COA

   Bei dieser Methode wird die Defuzzifizierung der Akkumulation vorgezogen. Also es werden zunächst für alle Regeln die Werte $\nu_r^-(o_r)$ (siehe oben) berechnet, um aus diesen den Gesamtausgabewert zu akkumulieren:

$$\eta = \frac{\sum\limits_{r=1}^{m} o_r \cdot \nu_r^-(o_r)}{\sum\limits_{r=1}^{m} o_r}.$$

   Diese Methode ist recht schnell und eignet sich gut für Lernverfahren. Die von ihr errechneten Ausgabewerte liegen nahe an denen der diskreten COA. Sie eignet sich nicht für Optimierungsverfahren durch Veränderung der Breite der Konklusions–Fuzzy–Mengen. Vor allem bei Verwendung symmetrischer Fuzzy–Mengen hat die Verbreiterung keinen Effekt. Vielmehr ist dann nur die Lage der Spitze bedeutend, die Fuzzy–Mengen können dann zu der Form eines Impulses vereinfacht werden.

3. Diskrete COA mit Gödel–Inferenzmechanismus

   Speziell um die Berechnung des Fehlers auf die Grundlage der Fuzzy–Relationen zu stellen, wurde die Möglichkeit implementiert, den Fehler auf dem Gödel–Inferenzverfahren und der COA basierend zu berechnen. Bei diesem Verfahren wird zuerst die Ausgabe–Fuzzy–Menge akkumuliert und dann defuzzifiziert. Die Akkumulation geschieht durch konjunktive Verknüpfung (mit dem min–Operator) der Ausgabe–Fuzzy–Mengen der einzelnen Regeln, wobei diese mit Hilfe der Gödel–Implikation (siehe [Kruse et al. 1993]) berechnet werden.

   Wie schon erwähnt, steht diese Methode nur der Berechnung des Fehlers zur Verfügung.

Zusammenfassend läßt sich anmerken, daß ich die besten Erfahrungen mit der diskreten COA gemacht habe, auch wenn der Beitrag zum Steuerwert nicht exakt, sondern nur durch eine Heuristik ermittelt wird. Auf langsameren Rechnerarchitekturen verwende man am besten die “average COA”, die auch gute Ergebnisse liefert und dabei viel schneller ist. Zu beachten ist, daß alle vorgestellten Defuzzifizierungsverfahren bis auf MOM nicht für die Verwendung zusammen mit dem ursprünglichen NEFCON–Optimierungsalgorithmus gedacht sind.

Alle oben vorgestellten Defuzzifizierungsverfahren lassen sich auch zur Fehlerberechnung heranziehen. Aus Gründen der Abarbeitungsgeschwindigkeit bevorzuge ich dazu die MOM bzw. die ’average COA’ Methode.

Kapitel 3

Das Lernen von Kontrollregeln

In diesem Abschnitt werden drei in der Simulationsumgebung NEFCON–I zur Verfügung stehende Lernmethoden für Kontrollregeln vorgestellt. Die dabei verfolgten Ansätze sind sehr unterschiedlich, und obwohl sich bei ihnen der Lernvorgang in drei Phasen gliedert, haben diese Phasen unterschiedliche Bedeutungen.

3.1 Das ursprüngliche Verfahren zum Erlernen von Kontrollregeln

Dieses Verfahren ist von Detlef Nauck im Rahmen seiner Dissertationsarbeit [Nauck 1993] entwickelt worden. Bei dieser Methode 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. In den Phasen des Lernvorgangs werden Regeln, die für falsch oder überflüssig gehalten werden, entfernt.

- Phase 1
  In dieser Phase werden alle Regeln entfernt, deren Ausgaben ein vom Fehler E^* verschiedenes Vorzeichen aufweisen. Zusätzlich durchlaufen die Regeln einen Alterungsprozeß. Regeln, die nur selten oder gar nicht während eines vorgegebenen Zeitraums benutzt wurden, werden entfernt. Nach Abschluß dieser Lernphase enthält die Regelbasis nur noch Regeln deren Vorzeichen der Ausgabewerte für die gleichen Prämissen identisch sind.

- Phase 2
  In dieser Phase wird unter Regeln mit gleicher Prämisse die geeigneteste gesucht, und alle anderen werden gelöscht. Als Maß für die Richtigkeit einer Regel wird die Summe der mit Erfüllungsgrad o_r der Regeln gewichteten Fehler E^* über einen bestimmten Zeitraum genommen. Die Regel, deren Fehlersumme nach Abschluß der Lernphase am kleinsten ist, wird beibehalten. Um sicherzustellen, daß während dieser Lernphase alle Regeln mit gleicher Prämisse gleich oft "ausprobiert" werden, beteiligt sich eine Regel mit der Wahrscheinlichkeit $\frac{2}{n}$ am aktuellen Lernzyklus, wobei n die Anzahl der vorhandenen Konklusionen zur gegebenen Prämisse ist.

- Phase 3
  In dieser Phase werden die Fuzzy–Mengen nach einem der im letzten Abschnitt beschriebenen Verfahren optimiert. Diese Lernphase ist allen Regellernverfahren gleich.

3.2 Das "Bottom-Up" Verfahren zum Erlernen von Regelbasen

Aufgrund der Annahme, daß eine adäquate Partitionierung der linguistischen Variablen schon vorliegt, also die Menge der möglichen Gewichte der noch herzustellenden Verbindungen zwischen den Schichten des Neuronalen–Fuzzy–Reglers bekannt ist, habe ich ein Lernverfahren entwickelt, das die Verbindungen zwischen den Schichten herstellt und sie mit passenden Gewichten belegt.

Das Lernverfahren gliedert sich in zwei Phasen. In der ersten Phase werden die Meßwerte mit Hilfe der vorhandenen Partitionierung klassifiziert. Die Annahme die diesem Verfahren zugrunde liegt ist, daß ähnliche Meßwerte durch eine Fuzzy–Menge und ähnliche Meßwertetupel durch ein entsprechendes Tupel von Fuzzy–Mengen repräsentiert werden kann.

Zusätzlich dazu wird angenommen, daß Meßtupel, die einen ähnlichen Fehlerwert zufolge haben, auch einen ähnlichen Steuerwert benötigen, um das zu regelnde System in einen optimalen Zustand zu steuern. Diese Annahme führt zur Bildung von Verbindungen zwischen der Regel und der Ausgabeschicht. Sie werden mit einer Fuzzy–Menge als Gewicht vorbelegt, die durch ein heuristisches Verfahren (s.u.) aus dem Fehlerwert berechnet wird. Es ist allerdings auch möglich, an dieser Stelle eine Fuzzy–Menge zufällig aus der Partitionierung der linguistischen Ausgabevariable auszuwählen. Die Regelkonklusionen werden in der zweiten Lernphase optimiert. Die Heuristik verkürzt und verbessert das Lernverfahren jedoch erheblich.

In der zweiten Lernphase wird davon ausgegangen, daß die Regelprämissen schon korrekt modelliert und die Struktur des Neuronalen–Fuzzy–Reglers, repräsentiert durch vorhandene Regeln, korrekt ist. Aus der Menge der möglichen Gewichte, also der Fuzzy–Mengen der Partitionierung der linguistischen Ausgabevariablen, wird jeweils diejenige ausgewählt, die das System einem Optimalzustand näherbringt.

Im folgenden werden die oben beschriebenen Vorgänge präzisiert:

Sei $\bar{X} = (x_1, \ldots, x_n)$ ein Tupel von Eingabewerten, E^* der aus $\bar{X}$ berechnete Fehlerwert. Dann wird in der **ersten Lernphase** die Regel

$$\texttt{IF } \xi_1 = \hat{A}_1 \texttt{ AND } \cdots \texttt{ AND } \xi_n = \hat{A}_n \texttt{ THEN } \eta = \hat{B}$$

eingefügt, wobei

- $\hat{A}_i$, $\hat{B}$ linguistische Terme
- $\hat{\mu}^i$ die dem linguistischen Term $\hat{A}_i$ zugeordnete Fuzzy–Menge ist, die den höchsten Treffergrad für den Meßwert x_i, wobei gilt:
  $\hat{\mu}^i = \mu^i_j, j = 1, \ldots, m_i$
  $\iff (\forall k = 1, \ldots, m_i, k \neq j)(\mu^i_j(x_i) \geq \mu^i_k(x_i))$
- $\hat{\nu}$ die dem linguistischen Term $\hat{B}$ zugeordnete Fuzzy–Menge ist, die den höchsten Treffergrad für den heuristischen Regelausgabewert o hat , wobei gilt:
  $\hat{\nu} = \nu_j, j = 1, \ldots, m$
  $\iff (\forall k = 1, \ldots, m, k \neq j)(\nu_j(o) \geq \nu_k(o))$
- o der heuristische Ausgabewert beim Fehler E, wobei

$$o = \begin{cases} |E^*| \cdot y^{\max} & \text{falls } E^* \geq 0 \\ |E^*| \cdot y^{\min} & \text{falls } E^* < 0, \end{cases}$$

  wobei $y^{\max}$ und $y^{\min}$ die Grenzen de Definitionsbereiches der linguistischen Ausgabevariablen sind.

Sei zum Beispiel ein System mit einer Eingabevariable und einer Ausgabevariable mit Paritionierungen wie in Bild 3.1 gegeben.

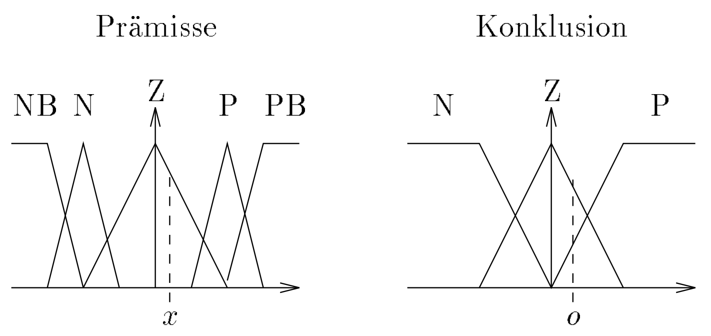

Abbildung 3.1: Beispiel für das Erlernen einer Regel mit dem "Bottom-Up" Verfahren in der ersten Lernphase

Die Fuzzy–Menge `Z` hat im Bild 3.1 den höchsten Treffergrad für den Meßwert x also gilt $\hat{\mu} = \mathtt{Z}$. Der aus x berechnete Fehler E^* führt nach dem obigen heuristischen Verfahren zum Regelausgabewert o. für o ist `Z` die Fuzzy–Menge der Asugabevariablen, die den höchsten Treffergrad hat. Es gilt also $\hat{\nu} = \mathtt{Z}$. Also muß, falls noch nicht vorhanden, die Regel

`IF` Prämisse `is` `Z` `THEN` Konklusion `is` `Z`

eingefügt werden.

In der **zweiten Lernphase** wird versucht die Fuzzy–Mengen der Regelkonklusionen zu optimieren. Dazu wird wie in der Definition 2.2 der Verschiebewert δ^r_{conc} für jede Regel R_r berechnet. Weiter wird für jede Regel R_r der Wert $t_r = \nu_r^-(o_r) + \delta^r_{\text{conc}}$ mit dem Erfüllungsgrad der Regelprämisse o_r und $\nu_r^-(o_r)$ dem Regelausgabewert bzw. der Anteil der Regel R_r an der Gesamtausgabe. δ^r_{conc} ist der mit Hilfe des Algorithmus aus der Definition 2.2 berechntete Parameteränderungswert. Somit stellt der Wert t_r den von dem Lernalgrithmus "gewünschten Anteil" der Regel R_r an der Gesamtausgabe. Die Regel R_r bekommt dann als Konklusion die Fuzzy–Menge $\hat{\nu}_r$ für die gilt:

$$\hat{\nu} = \nu_j, j = 1, \ldots, m \Longleftrightarrow (\forall k = 1, \ldots, m, k \neq j)(\nu_j(t_r) \geq \nu_k(t_r))$$

Die Fuzzy–Menge $\hat{\nu}_r$ ist diejenige mit dem höchsten Zugehörigkeitsgrad für den "gewünschten Anteil" t_r.

Das Vorgehen wird im Bild 3.2 verdeutlicht.

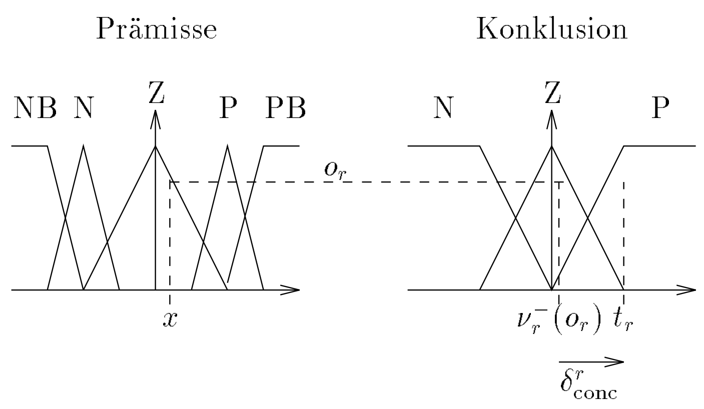

Abbildung 3.2: Beispiel für das Erlernen einer Regel mit dem "Bottom-Up" Verfahren in der zweiten Lernphase

Betrachtet wird die Regel

`IF` Prämisse `is` `Z` `THEN` Konklusion `is` `Z`.

Die Berechnung des Algorithmus ergibt, daß für den Meßwert x nicht mehr die Fuzzy–Menge `Z` sondern nun die Fuzzy–Menge `P` für die geeigneteste gehalten wird. Die Regel wird als geändert zu

`IF` Prämisse `is` `Z` `THEN` Konklusion `is` `P`.

Beim Übernehmen der Regeln in die Regelbasis nach abgeschlossenem Lernen wird darauf geachtet, daß Regeln, die nur selten aktiviert wurden, vernachlässigt werden.

Nachfolgend wird ein Beispiel für die Arbeit des Algorithmus vorgestellt. Als Anwendung wurde hier wieder das inverse Pendel gewählt. Dieses Beispiel wurde mit der Simulationsumgebung NEFCON–I (siehe Kapitel 4) erstellt.

Zur Erstellung des Beispiels wurde die Fehler–Wissensbasis aus der Abbildung 1.3 und die aus dem Bild 3.3 zu entnehmende Partitionierungen der linguistischen

Variablen verwendet. Diese Partitionierung gilt für alle benötigten linguistischen Variablen. Sie wurde mit der Option zur Erstellung von standardisierten Partitionierungen automatisch generiert.

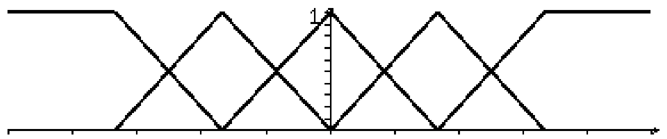

Abbildung 3.3: Partitionierung der linguistischen Variablen, die Fuzzy–Mengen haben die Namen `NB, N, Z, P, PB`

Das Verfahren startet mit einer leeren Regelbasis (ausgeschaltete preinstall–Option). Die erste Lernphase dauert 100 und die zweite 300 Schritte. Nach jeder Lernphase werden Regeln, deren durchschnittlicher, absoluter Regelfehler > 0.8 war, und Regeln, die in 99% aller Fälle nicht benutzt wurden, aus der Regelbasis gestrichen. Während des Lernvorgangs werden die Fuzzy–Mengen nicht optimiert.

Nach Abschluß der ersten Lernphase hat die Regelbasis die im Bild 3.4 sichtbare Gestalt.

rule table

<winkel>

<w_geschw>	nb	n	z	p	pb
nb					
n			n	n	
z		n	pb	p	
p		p	pb		
pb					

Abbildung 3.4: Regelbasis nach der ersten Lernphase

Man erkennt, daß schon nach der ersten Lernphase eine recht gute Regelbasis entstanden ist, bis auf die falsch gewählte Fuzzy–Menge `pb` in der Regel mit der Prämisse *Winkel* `is z AND` *Winkelgeschwindigkeit* `is z`. Die zum Teil recht gute Belegung der Regelprämissen liegt unter anderem auch daran, daß der implementierte Algorithmus schon in der ersten Lernphase die Optimierungen der zweiten Lernphase in geeigneten Fällen vornimmt.

Nach der zweiten Lernphase ist eine sehr gute Regelbasis entstanden.

rule table

<winkel>

<w_geschw>	nb	n	z	p	pb
nb					
n			n	n	
z		n	z	p	
p		p	p		
pb					

Abbildung 3.5: Regelbasis nach der zweiten Lernphase

Das Nichtvorhandensein von Regeln für große Winkel und Winkelgeschwindigkeiten ist aus der Tatsache zu erklären, daß das Pendel nur mit zufälligen Winkeln aus dem Intervall $[-35°, 35°]$ angestoßen worden ist.

3.3 Das Partitionslernverfahren

Dieses Lernverfahren ist aus dem Versuch entstanden, die komplette Steuer–Wissensbasis zu erlernen. Es benötigt keine vorgegebene Partitionierung der linguistischen Variablen, sondern es versucht, diese selbständig zu erlernen. Das Verfahren startet also mit einer leeren Regelbasis, und von den linguistischen Variablen ist nur der Name und ihre Funktion als Ein- oder Ausgabevariable bekannt. In der Implementierung wird aus praktischen Gründen zusätzlich noch die Angabe der maximalen Partitionierungsgröße und ein minimaler Treffergrad $\tau_{\min}$ für die Erzeugung einer neuen Fuzzy–Menge als Eingabe verlangt.

Der Algorithmus gliedert sich wie üblich in zwei Teile und einen dritten, der aus der im vorigen Abschnitt beschriebenen Optimierung der Fuzzy–Mengen besteht. Im folgenden werden die entscheidenden zwei Phasen des Lernalgorithmus beschrieben:

- Phase 1
  In dieser Phase wird versucht, die Eingabewerte mit Hilfe von Fuzzy–Mengen zu klassifizieren. Neue Fuzzy–Mengen werden in die Partitionierung einer linguistischen Eingabevariable erst dann aufgenommen, wenn für einen Meß-

wert x der Treffergrad $\mu(x) < \tau_{\min}$ für alle in der Partitionierung schon vorhandenen Fuzzy–Mengen μ ist. In diesem Fall wird eine neue symmetrische, dreieckförmige Fuzzy–Menge μ_{neu} mit $\mu_{\text{neu}}(x) = 1$ in die Partitionierung aufgenommen.

Unabhängig davon, ob eine neue Fuzzy–Menge erzeugt wurde oder nicht, wird, falls notwendig, eine neue Regel erzeugt. Dazu werden diejenigen Fuzzy–Mengen der linguistischen Eingabevariablen gesucht, die für den gegebenen Meßwertevektor die höchsten Treffergrade haben (ggf. enthält dieser Vektor gerade erzeugte Fuzzy–Mengen, denn sie haben den Treffergrad 1). Aus diesen Fuzzy–Mengen wird die Regelprämisse α gebildet. Aus dem erweiterten Fuzzy–Fehler E^* der heuristische Stellwert o_η mit

$$\eta = \begin{cases} |E^*| \cdot y^{\min} & \text{falls } E^* < 0 \\ |E^*| \cdot y^{\max} & \text{falls } E^* \geq 0 \end{cases}$$

berechnet, wobei $y^{\min}$ und $y^{\max}$ die Grenzen des Definitionsbereichs der entsprechenden Ausgabevariable sind.

Befindet sich in der Regelbasis noch keine Regel mit α als Prämisse, so wird eine neue Regel mit der Prämisse α und einer Konklusions–Fuzzy–Menge ν aufgenommen. ν ist entweder eine schon in der Partitionierung der Ausgabevariablen vorhandene Fuzzy–Menge, oder, falls für alle Fuzzy–Mengen ν' der Ausgabevariablen gilt $\nu'(o_\eta) < \tau_{\min}$, dann wird ν neu erzeugt und in die Partitionierung aufgenommen.

Ist α schon Prämisse einer Regel der Regelbasis mit einer Konklusions–Fuzzy–Menge ν_{alt}, so wird nach dem Optimierungsalgorithmus durch Verschieben (siehe Definition 2.2) der Wert δ_{conc} berechnet, und diejenige Fuzzy–Menge ν_{neu} als neue Konklusion genommen, die den höchsten Treffergrad für den Wert $\nu_{\text{alt}}^{-}(o) + \delta_{\text{conc}}$ hat, wobei o der Erfüllungsgrad der Prämisse α und $\nu_{\text{alt}}^{-}(t)$ der Beitrag der alten Regel zum Gesamtsteuerwert ist.

Die Breite β der eingefügten Fuzzy–Mengen ergibt sich aus $\beta = \frac{y^{\max} - y^{\min}}{n_{\max}}$. $n_{\max}$ ist eine für alle linguistischen Variablen voreinstellbare maximale Anzahl von Fuzzy–Mengen in der Partitionierung.

- Phase 2
  In der zweiten Phase wird versucht, die Partitionierung und die Regelbasis

zu optimieren. Dieses geschieht durch den Optimierungsalgorithmus durch Verschieben (siehe Definition 2.2).

Für die Regelkonklusionen wird ggf. eine neue Fuzzy–Menge der Partitionierung angenommen. Die neue Fuzzy–Menge ist dann diejenige mit dem höchsten Treffergrad für $\nu^{-}_{\text{alt}}(o_r) + \delta_{\text{conc}}$ (s.o.).

Das Partitionslernverfahren berücksichtigt auch den Wert des Lernfaktors. Dieser sollte in Abhängigkeit von der Berechnungsmethode für den Fuzzy–Fehler gewählt werden.

Am folgenden Beispiel wird das Verhalten des Lernalgorithmus demonstriert. Der Algorithmus startet mit leeren Partitionierungen der linguistischen Variablen und der Fehler–Wissensbasis aus dem Beispiel 1.3.

Die voreingestellte maximale Anzahl von Fuzzy–Mengen pro linguistische Variable ist 5 und der minimale Treffergrad, ab dem eine neue Fuzzy–Menge in die Partitionierung übernommen wird, ist $\tau_{\min} = 0.1$. Die erste Phase dauert 200 und die zweite 1000 Schritte. Die Optimierung der Fuzzy–Mengen ist ausgeschaltet, bzw. das System ignoriert diese Option per default. Zur Fehlerberechnung wird das ein Mamdani Regler mit der Defuzzifizierung nach MOM gewählt (Definitionsdatei: `exam1_e.lv`, `exam1_e.rg`). Der Lernfaktor wird 0.5 gesetzt.

Um ein besseres Lernergebnis zu erzielen, wurde das Pendel anfangs mit dem Winkel 0 gestartet, und erst nach einigen Lernschritten wurde das Pendel periodisch mit einem zufälligen Winkel neu angestoßen.

Der Lernalgorithmus bennent die Fuzzy–Mengen nach der Lage ihrer Spitze. Für kleine Winkel zeigt die bis zu diesem Zeitpunkt gelernte Wissensbasis schon ein recht gutes Verhalten. Man kann beobachten, daß für die Ausregelung des Pendels nur 3 Fuzzy–Mengen in der Partitionierung der Ausgabevariable benötigt werden. Diese Beobachtung konnte schon beim “Bottom–Up” Lernalgorithmus gemacht werden.

Die erlernte Wissensbasis schafft es, Winkel im Intervall $[-30, 32]$ Grad auszusteuern. Das Regelverhalten ist allerdings nicht ideal, denn bei hohen positiven Winkeln neigt das System dazu, zu übersteuern, und bei negativen hat die Regelkurve einen sehr flachen Verlauf.

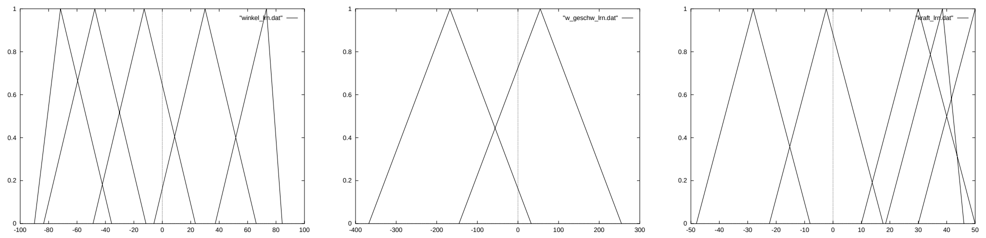

Abbildung 3.6: Partitionierungen der Linguistischen Eingabevariablen Winkel nach der zweiten Lernphase

rule table

<winkel>

<w_geschw>

	-71.65	-47.51	-12.74	30.08	73.24		
-167.42	-2.38		-28.06				
54.81		-28.06	-2.38	29.97	50.00		

Abbildung 3.7: Die Regelbasis nach der zweiten Lernphase

3.4 Zusammenfassende Bemerkungen

Die besten Ergebnisse erzielt das "Bottom–Up" Verfahren. Für das einfache Pendel findet es eine gute Regelbasis, falls korrekte Partitionierungen der linguistischen Variablen vorliegen. Das Partitionslernverfahren liefert nicht so gute Ergebnisse. Die Ergebnisse sind zum Teil sehr stark von der statistischen Verteilung der Meßwerte abhängig. Es liefert fast ausschließlich nur dann gute Ergebnisse, wenn es schafft, den Optimalzustand zu erlernen, also die Regel

`IF` angle `is` 0 `and` velocity `is` 0 `THEN` force `is` 0

zu finden. Wenn die entsprechenden Meßwerte nicht zum Anfang des Lernverfahrens zur Verfügung gestellt werden, verfängt sich der Algorithmus in kompensatorischen Zuständen.

Desweiteren haben Beobachtungen gezeigt, daß sich Optimierungsverfahren für Fuzzy–Mengen zur Zeit noch nicht mit den Regellernalgorithmen vertragen. Aus diesem Grunde ist es empfehlenswert, bei der Arbeit mit der Simulationsumgebung die Option für das Optimieren der Fuzzy–Mengen während des Regellernens auszuschalten. Dies nehmen die Algorithmen an besonders kritischen Stellen zum Teil selbst vor.

Kapitel 4

NEFCON–I

Dieses Kapitel dient als Benutzerhandbuch für das Programmpaket NEFCON–I bestehend aus der Simulationsumgebung NEFCON und zwei Beispielanwendungen, dem inversen Einfachpendel und dem inversen Doppelpendel.

NEFCON–I steht für *Neural–Fuzzy–Controller–InterViews* und implementiert das in den vorangegangenen Kapiteln beschriebene Modell. Anwendungen für NEFCON sind selbständige Programme, die über definierte Schnittstellen (Pipes) mit NEFCON kommunizieren.

NEFCON–I stellt eine auf den InterViews Grafikklassen aufbauende, grafische Benutzeroberfläche zur Verfügung. Es ermöglicht, Wissensbasen, bestehend aus linguistischen Variablen und Regeln, in einfach zu bedienenden Editoren zu definieren. Das Verhalten des Reglers und der Lernalgorithmen wird grafisch veranschaulicht.

4.1 Prozeßregelung mit NEFCON–I

Wie aus dem Bild 4.1 ersichtlich, kann man den Aufbau von NEFCON–I in drei Teile gliedern. Die Anwendung ist ein eigenständiger Prozeß, der seine Meßwerte in den Standard–Ausgabekanal (`stdout`) schreibt und die Steuerwerte aus dem Standard–Eingabekanal (`stdin`) liest.

Die grundlegende Schicht von NEFCON–I ist die *Wissensbasis*. Sie besteht aus der optionalen *Fehler–Wissensbasis*, die nur dann benötigt wird, wenn ein Lernalgo-

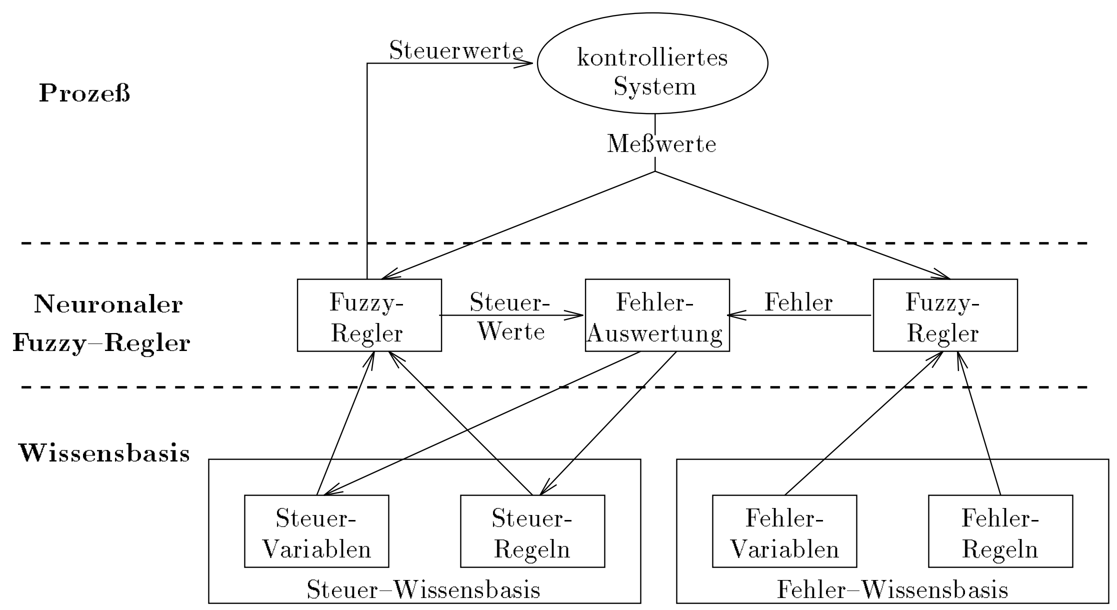

Abbildung 4.1: Schichten der Prozeßregelung mit NEFCON–I

rithmus benutzt wird, und aus der *Steuer–Wissensbasis*. Die Steuer–Wissensbasis kann entweder für einen eigenständigen, einfachen Fuzzy–Regler benutzt werden oder in Verbindung mit der Fehler–Wissensbasis von einem Lernalgorithmus erlernt oder optimiert werden.

Die Fehler–Wissensbasis und die Steuer–Wissensbasis haben die gleiche interne Struktur (und sind in der Tat Instanzen der gleichen C++–Klasse). Sie bestehen jeweils aus einem Satz linguistischer Variablen und Regeln. In Abhängigkeit vom Typ der zugrundeliegenden Anwendung müssen für die Steuer–Wissensbasis Eingabe und Ausgabevariablen mit ihren Partitionierungen definiert werden. Für die Fehler–Variablen werden für die gleichen Eingabevariablen eigene Partitionierungen gewählt. An Stelle der Ausgabevariablen wird die fest vorgegebene Ausgabevariable `error`, die das erweiterte Fuzzy–Fehler–Maß modelliert, definiert.

Der Zusammenhang zwischen den jeweiligen Ein- und Ausgabevariablen wird über linguistische Regeln hergestellt.

Der Neuronale Fuzzy–Regler, der Kern von NEFCON–I, hat die Aufgabe, Meßwerte von der zu regelnden Anwendung anzunehmen und zu verarbeiten. Mit Hilfe des Fehlers beurteilt NEFCON–I den Zustand der Anwendung und modifiziert ggf. seine Steuer–Wissensbasis. Dann berechnet NEFCON–I die von der Anwendung benötigten Steuerwerte und schickt sie an diese. Im Schema besteht der Neuronale Fuzzy–Regler aus drei Komponenten, aus zwei Fuzzy–Reglern und dem Fuzzy–Fehler–Propagierungsalgorithmus. Der erste Fuzzy–Regler dient zur Berechnung der Steuerwerte, und der zweite zur Berechnung des von dem Fuzzy–Fehler–Propagierungsalgorithmus benötigten erweiterten Fuzzy–Fehlerwert. Die beiden letzten Komponenten entfallen, falls NEFCON–I als einfacher Fuzzy–Regler, ohne Lernfähigkeit, betrieben wird.

4.2 Programmstart

NEFCON–I wird mit dem Kommando `nefcon` aufgerufen. Beim Start berücksichtigt NEFCON–I einige Optionen und X–Ressourcen, die aus den untenstehenden Tabellen entnommen werden können.

Ressourcen zum Festlegen der äußeren Erscheinung

Ressource	Default
FieldEditor*background	"#90ffa0"
Markierung	"#000000"
cursorFlashRate	"0"
PushButton*font	"*times-bold-r-normal–12*"
NfcEditorFont*font	"*helvetica-medium-r-normal–12*"
NfcInfoFont*font	"*helvetica-bold-r-normal–12*"

Pfade für die Wissensbasis

Resource	Default
DataDirectory	"my_data/"
DefaultLv	"aa.lv"
DefaultErrLv	"aa1_f.lv"
DefaultRg	"aa1.rg"
DefaultErrRg	"aa1_f.rg"
DefaultApplication	"run"
protocol	"protocol.nfc"

Ressourcen für Protokoll und Lernoptionen

Resource	Optionen	Beschreibung
ErrorComputation	**Mamdani_MOM**, Goedel_COA, Mamdani_dis_COA, Mamdani_COA, Mamdani_average_COA	Gibt die Methode zur Berechnung des erweiterten Fuzzy Fehlers an.
Defuzzification	Mamdani_MOM, **Mamdani_dis_COA**, Mamdani_COA, Mamdani_average_COA	Die Methode zur Berechnung des Stellwertes.
RuleLearnMode	**bottom-up**, NEFCON-originally, learn_partition	Der Regellernalgorithmus.
RegelRep	none, info, as_board, **as_table**	Die Protokolloption die angibt auf welche Art Regeln von der Kontrolleroberfläche dargestellt werden.
LingVarRep	as_graph, **none**	Die Protokolloption die angibt ob die ling. Variablen von der Kontrolleroberfläche dargestellt werden.
OutputRep	as_graph, **none**	Die Protokolloption die angibt ob die akkumulierte Ausgabefuzzymenge von der Kontrolleroberfläche dargestellt wird.

Resource	Optionen	Beschreibung
WerteRep	numerically, **graphically**, none	Die Protokolloption die angibt auf welche Weise die Systemwerte (Stellwert, Fehlerwert, die Werte der Eingabevariablen) dargestellt werden.
WithTrigger	on, **off**	Option die angibt ob das Triggersignal für die Anwendung bereitgestellt werden soll.
TriggerAveError	Float–Wert **0.02**	Die Schranke für den durchschnittlichen Fehlerwert bei dem das Triggersignal abgeschickt wird.
ApplicationTriggerTime	Int–Wert **20**	Das Zeitintervall nach dem das Triggersignal abgeschickt wird, falls der durchschnittliche Fehler vom Betrag die Schranke unterschritten hat.
MoveAndStretch	on, **off**	Option die bewirkt, daß sowohl der Algorithmus für Optimierung durch Veränderung der Breite als auch der Algorithmus zur Veränderung der Lage benutzt wird.

Werden keine Ressourcen, gesetzt so nimmt NEFCON–I die in den obigen Tabellen hervorgehobenen Werte für Ressourcen an.

Nach dem Start von NEFCON–I erscheint zusätzlich zum NEFCON–Hauptfenster 4.2 das NEFCON–Statusfenster 4.3, das über die wichtigsten Ressourcen informiert.

Das NEFCON–Hauptfenster besteht aus einer Menüleiste, unterschiedlichen But-

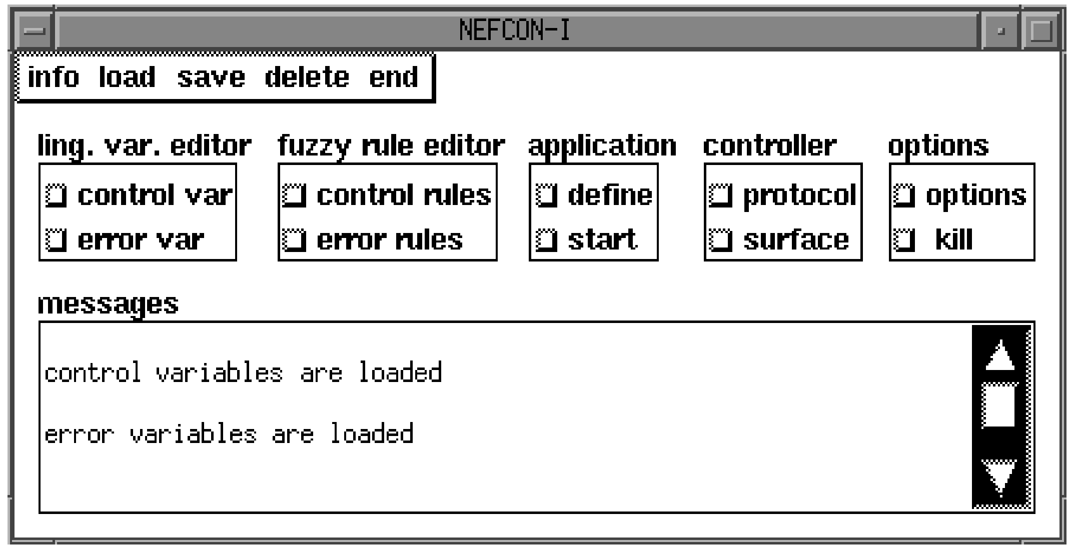

Abbildung 4.2: Das NEFCON–I Hauptfenster

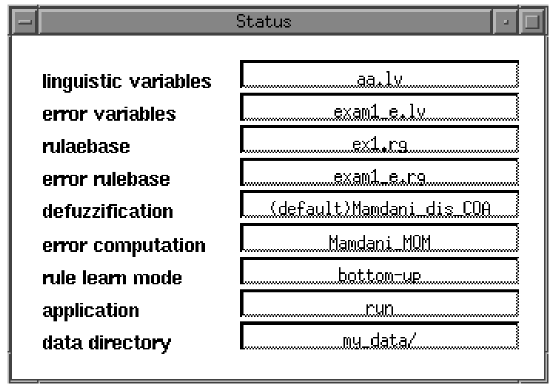

Abbildung 4.3: Das NEFCON–I Statusfenster

tons und einem Ausgabefeld für Meldungen. Editoren und die Regleroberfläche werden in separaten Fenstern dargestellt. Sie werden durch Maus–Klicks auf die entsprechenden Buttons aufgerufen.

Sollte NEFCON–I ohne eine durch Ressourcen voreingestellte Wissensbasis gestartet worden sein, oder sollte eine ganz neue Wissensbasis erzeugt werden, müssen folgende Schritte durchgeführt werden:

1. Starten von NEFCON–I als einfacher Fuzzy–Regler ohne Lernen
   (a) Definition von linguistischen Variablen
   (b) Definition von linguistischen Regeln
   (c) Definition einer Anwendung
   (d) Auswahl einer passenden Defuzzifizierungsstrategie
   (e) Start des Anwendungsprozesses
2. Starten von NEFCON–I als Neuronaler Fuzzy–Regler
   (a) Definition von linguistischen Variablen
   (b) Definition von linguistischen Fehler–Variablen
   (c) Definition von linguistischen Regeln
   (d) Definition von linguistischen Fehler–Regeln
   (e) Definition einer Anwendung
   (f) Auswahl der Berechnunsstrategie für den Fehler
   (g) Auswahl einer passenden Defuzzifizierungsstrategie für die Ausgabefuzzymenge
   (h) Auswahl eines Lernverfahrens
   (i) Start des Anwendungsprozesses

Um die Definition der Wissensbasen zu erleichtern, enthält NEFCON–I zwei Typen von Editoren. Der erste dient zum Edieren von linguistischen Variablen (`ling. var. editor`). Er wird durch anklicken eines der beiden Buttons $\boxed{\text{control var}}$ oder $\boxed{\text{error var}}$ gestartet. Der andere Editor dient zur Definition von Fuzzy–Regeln (`fuzzy rule editor`) und wird gestartet mit den Buttons control rules oder error rules.

Mit den Buttons aus der Gruppe `application` läßt sich mit define eine Anwendung definieren und mit start starten. Die Definition einer Anwendung besteht aus der Angabe ihres Namens und der Festlegung der Reihenfolge der Eingabe- und der Ausgabevariablen. Eine Anwendung läßt sich erst dann definieren, wenn die linguistischen Variablen definiert sind.

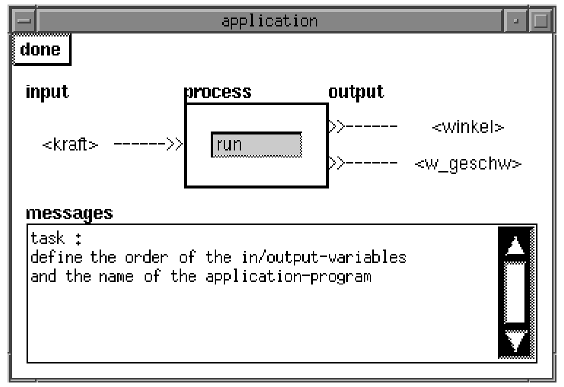

Abbildung 4.4: Definition einer Anwendung

Die Kontroller–Oberfläche (siehe Abbildung 4.2) erhält man nach dem Anklicken des Buttons surface, wobei das Erscheinungsbild der Oberfläche sich mit Hilfe von Optionen, die nach dem Anklicken von protokoll präsentiert werden (siehe Abbildung 4.5), festlegen läßt. Die genaue Bedeutung der einzelnen Optionen wird im Abschnitt 4.6 vorgestellt.

Besondere Beachtung verdient die Option create a protocol file. Mit ihrer Hilfe wird eine Datei angelegt, die Informationen über die Wissensbasis vor und nach dem Regelvorgang, der gleichzeitig auch ein Lernvorgang sein kann, sowie auch die Meßwerte, Stellwerte und Fehlerwerte für alle Regelzyklen enthält. Der Umgang mit der Protokolldatei wird im Abschnitt 4.7 besprochen.

Mit dem Button options lassen sich einige (zum Teil wichtige) Optionen einstellen (siehe Abbildung 4.6). Die Optionen umfassen die Auswahlmöglichkeiten für die Berechnungsstrategie für den erweiterten Fuzzy–Fehler (`error computation`), die Defuzzifikationsmethode (`defuzzification method`) und Voreinstellungen für das Regellernen und den Optimierungsalgorithmus für Fuzzy–Mengen. Besonders erwähnenswert ist die Trigger–Option. Sie erlaubt es unter der Bedingung, daß der

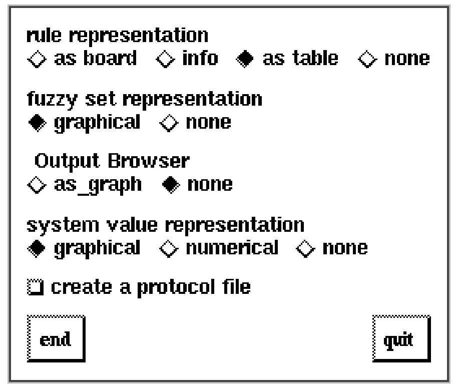

Abbildung 4.5: Definition des Protokolls

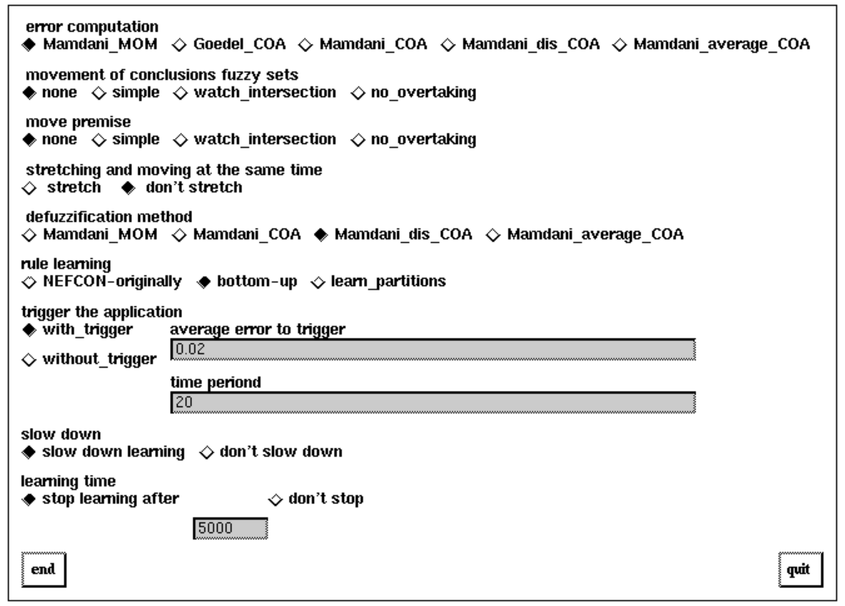

Abbildung 4.6: Definition der globalen Optionen

durchschnittliche Fehlerwert über eine bestimmte Zeit von Zyklen (`time period`) unter der Schranke (`average errorto trigger`) bleibt das Signal `t` an die Anwendung zu schicken, die, falls sie dieses Signal richtig interpretiert mit einem zufälligen Startwert neu gestartet wird. Sollte mit Anwendungen gearbeitet werden, die das Triggersignal `t` nicht verstehen, muß diese Option ausgeschaltet sein.

Einige dieser Optionen können schon als Ressourcen gesetzt werden.

Mit dem Button [kill] hat der Benutzer die Möglichkeit das Signal `SIGTERM` an den Anwendungsprozeß zu schicken und diesen somit zu beenden.

Selbstverständlich besteht die Möglichkeit, gelernte oder edierte Wissensbasen abzuspeichern und wieder zu laden. Das geschieht mit Hilfe der Optionen `load` und `save`, die in der Menüleiste des NEFCON–Hauptfensters (siehe Abbildung 4.2) erreichbar sind.

Mit Hilfe der Menüleistenoption `end` läßt sich NEFCON–I beenden.

4.3 Der Editor für linguistische Variablen

Der Aufruf des Editors (siehe Abbildung 4.7) geschieht aus dem Hauptfenster durch einen der Buttons [ling. var. editor] oder [error. var. editor]. Abhängig vom Button bezieht sich der Aufruf also entweder auf die Steuer- oder Fehler–Variablen.

Falls schon entsprechende Variablen in der Wissensbasis vorhanden sind, wird ihre Kopie in den Editor geladen. Der Editor wird verlassen durch das Anklicken des Buttons [done], der eine Wahl zwischen [exit], also übernehmen der gemachten Änderungen, oder [quit], dem Verlassen des Editors, wobei die Änderungen storniert werden.

Die Abbildung 4.7 zeigt den Editor bei der Arbeit. Die Editoroberfläche gliedert sich in vier Bereiche. Das Feld links oben dient zum Edieren von linguistischen Variablen. Eine linguistische Variable wird erzeugt durch die Angabe eines Namens, ihres Definitionsbereichs und ihres Typs in die entsprechenden Eingabefelder. Mit dem Button [OK] wird die linguistische Variable übernommen.

Eine besondere Eigenschaft bietet der Button [StdPart]. Er erzeugt eine standardisierte Partitionierung der linguistischen Variablen, wobei die Partitionierungsgröße im Feld `number`, und die Schnitthöhe der einzelnen Fuzzy–Mengen im Feld `hight`

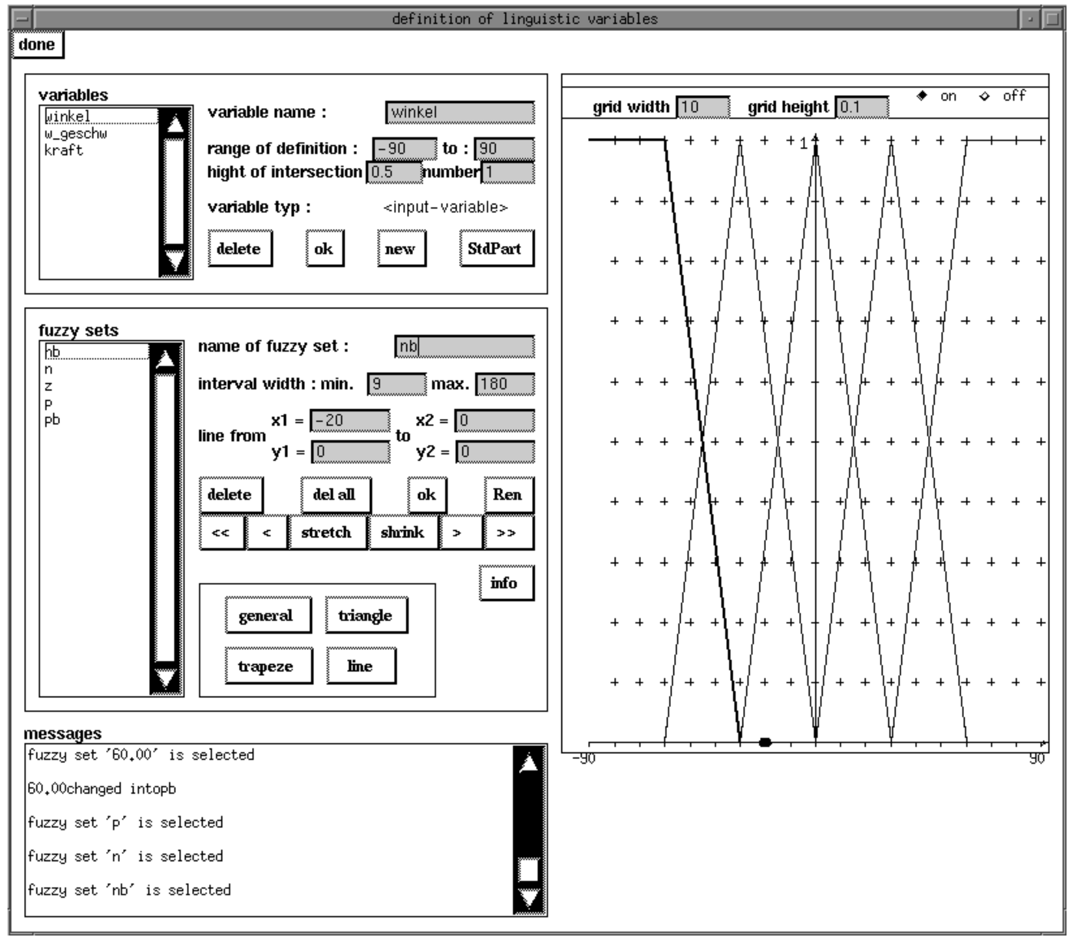

Abbildung 4.7: Der Editor für Fuzzy–Mengen

`of intersection` festgelegt werden. Die in der Abbildung 4.7 sichtbare Partitionierung ist eine solche automatisch erzeugte Partitionierung.

Das Feld unter dem Editor für linguistische Variablen bietet die Möglichkeit, einzelne Fuzzy–Mengen zu erzeugen und zu verändern.

Allgemeine Fuzzy–Mengen lassen sich aus Linien, die als Paare von Koordinatenpaaren $((x_1, y_1), (x_2, y_2))$ dargestellt werden, erzeugen. Daneben ist es auch möglich, vorgefertigte Formen von Fuzzy–Mengen einfach durch Anklicken eines der Buttons triangle, trapeze oder line zu erzeugen.

Mit line wird eine Linie erzeugt, wobei nur die x_1, x_2 Werte der Koordinaten in ein spezielles Eingabefenster eingegeben werden müssen. Die erzeugte Fuzzy–Menge hat dann den Graphen $((x_1, 0), (x_2, 1))$.

Entsprechend erhält man bei triangle oder trapeze durch Eingabe eines x–Koordinatentupels die Fuzzy–Mengen
$((x_1, 0), (x_2, 1), (x_3, 0))$ oder $((x_1, 0), (x_2, 1), (x_3, 1), (x_4, 0))$.

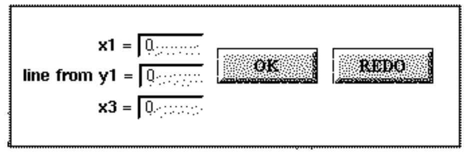

Abbildung 4.8: Definition der standardisierten dreieckförmigen Fuzzy–Menge

Die standardisierten Fuzzy–Mengen werden als geschultert erzeugt, falls sie über den Definitionsbereich der zugehörigen linguistischen Variablen hinausgehen.

Mit den Buttons <<, <, stretch, shrink, >, >> ist es möglich, die Fuzzy–Mengen zu verschieben (> in kleinen, >> in großen Schritten) und sie zu strecken oder zu stauchen. Der dazu maßgebliche Faktor ist der `grid width` Wert (s.u.), der in diesem Zusammenhang die Schrittweite angibt.

Mit OK wird eine Fuzzy–Menge in die Partitionierung aufgenommen und mit del gelöscht. Der del all Button dient zum Löschen der gesamten Partitionierung. Das Löschen der Partitionierung wird erst dann wirksam, wenn der OK Button im linguistische Variablen Feld als Bestätigung betätigt wird.

Im Koordinatensystem links in der Editoroberfläche wird die gesamte Partitionierung der linguistischen Variablen dargestellt, wobei die aktuell bearbeitete Fuzzy–Menge hervorgehoben dargestellt wird.

Es besteht auch die Möglichkeit, Fuzzy–Mengen innerhalb des Koordinatensystems zu malen. Es empfiehlt sich dazu, den Grid–Modus einzuschalten, um exakter positionieren zu können. Die rechte Maustaste besitzt innerhalb des Koordinatensystems eine besondere Bedeutung. Mit ihr wird der *Auswahlmodus* gestartet, bzw. nach einem weiteren Tastendruck beendet. Der Auswahlmodus gilt für Intervalle der gerade in Bearbeitung befindlichen Fuzzy–Menge. Im Auswahlmodus werden die Anfangs- und die Endpunkte der einzelnen Intervalle der die Fuzzy–Menge repräsentierenden Funktion in die $x_1, \ldots, y_2$ Felder übernommen und das Intervall durch große Punkte an den Enden gekennzeichnet. Das ausgewählte Intervall kann nun verändert werden.

Es besteht zusätzlich noch die Möglichkeit, Fuzzy–Mengen nachträglich umzubenennen. Dies geschieht durch Anwählen einer Fuzzy–Menge, Änderung ihres Namens im Namenseditorfeld und abschließendem Betätigen des Buttons $\boxed{\text{Ren}}$. Diese Option ist vor allem dann von Nutzen, wenn automatisch erzeugte Partitionierungen mit ansprechenden Namen versehen werden sollen. Dies kann entweder nach der Benutzung der Option zur Erzeugung von Standardpartitionierungen oder nach dem Partitionslernverfahren notwendig werden.

4.4 Der Regeleditor

Der Aufruf des Editors erfolgt, genau wie beim Editor für linguistische Variablen, im Hauptfenster durch die Buttons $\boxed{\text{control rules}}$ oder $\boxed{\text{error rules}}$. Auch hier wird, je nach Button, der Editor entweder für die Steuer-Regeln oder die Fehler--Regeln aufgerufen. Der Editor ist in Abb. 4.9 abgebildet.

Man erkennt in der Abbildung vier voneinander getrennte Bereiche, diese sind von oben nach unten :

Die Regeltafel (`rule board`):
Dies ist ein Auswahlfeld, das alle Regeln darstellt und die Möglichkeit bietet, einzelne auszuwählen.

Der Eingabebereich für die Regeln (`rule definition`):
In diesem Bereich können Regeln definiert oder gelöscht werden. Die Definition erfolgt über die einzelnen Eingabefelder und das Löschen über den entsprechenden Button.

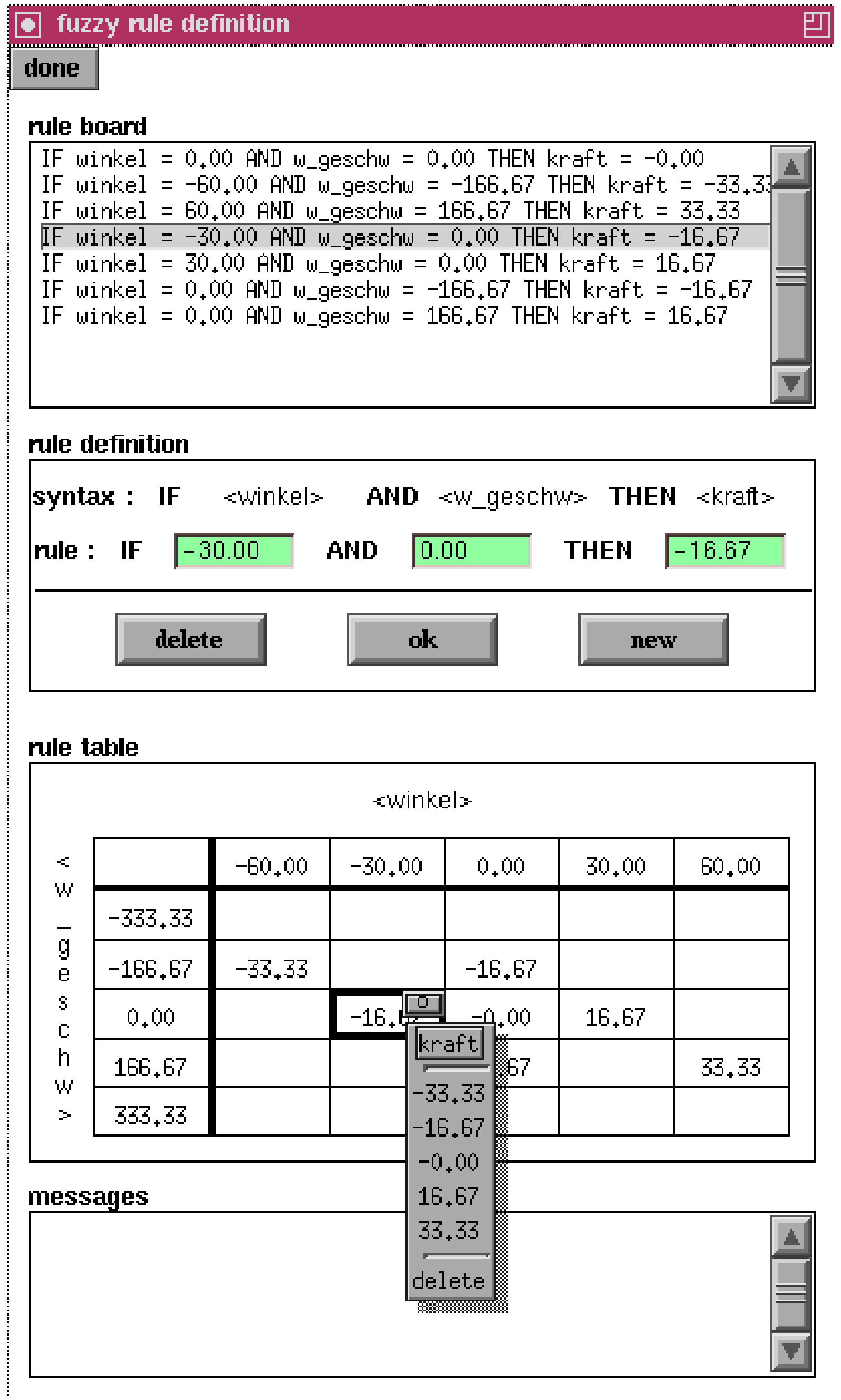

Abbildung 4.9: Der Editor für linguistische Regeln

Die Regeltabelle (`rule table`):
Hier werden die Regeln grafisch in einer Tabelle dargestellt und können mit der Maus ausgewählt werden. Es ist außerdem möglich, mit Hilfe der Maus neue Regeln zu definieren.

Grundsätzlich existieren also zwei Alternativen zur Darstellung und Eingabe 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 den Vorteil einer sehr übersichtlichen Darstellung von Regeln, zumindest bei Systemen mit zwei Eingabe- und einer Ausgabevariablen. Bei einer größeren Anzahl von Variablen geht die Übersichtlichkeit schnell verloren. Außerdem ist es nicht möglich, mit der Regeltabelle Regeln mit nur einer Bedingung in der Prämisse zu definieren. Es müssen, aufgrund der Konstruktion der Tabelle, mindestens zwei Bedingungen angegeben werden. Regel der Form

IF *Bedingung* THEN *Bedingung*

können also nur im Eingabebereich und nicht in der Tabelle definiert werden. Dies ist auch der Grund, weshalb die Regeltabelle nicht im Regel-Editor erscheint, wenn nur eine Eingabevariable definiert ist.

Bei der Definition von Regeln ist zu beachten, daß eine Regelbasis immer konsistent sein muß. Das heißt, sie darf keine Regeln mit gleicher Prämisse und widersprüchlicher Konklusion enthalten. Die folgenden beiden Regeln stellen ein solches Beispiel dar.

$$\text{IF } winkel = NE \text{ AND } geschw = PZ \text{ THEN } kraft = PS$$

$$\text{IF } winkel = NE \text{ AND } geschw = PZ \text{ THEN } kraft = NZ$$

Bei der Eingabe dieser beiden Regeln, wäre die zuletzt eingegebene gültig, und die erstere würde gelöscht werden.

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

Die Regeltafel

Die eigentliche Regeltafel ist, wie schon gesagt, ein Auswahlfeld, das die gesamte Regelbasis darstellt, und in dem durch einen Doppel-Klick mit der Maus einzelne Regeln selektiert werden können. Die jeweils selektierte Regel wird im Eingabebereich dargestellt.

Der Eingabebereich enthält zwei Zeilen, eine mit Toggle-Buttons und eine mit Eingabefeldern, zur Darstellung bzw. Eingabe einer Regel und die Buttons $\boxed{\text{delete}}$, $\boxed{\text{Ok}}$ und $\boxed{\text{new}}$.

Für jede linguistische Variable ist je ein Toggle-Button und ein Eingabefeld vorhanden. Die Buttons legen fest, zu welchen Variablen die Fuzzy-Mengen gehören, deren Namen in den Eingabefeldern stehen. Das heißt, daß jeder Button mit seinem dazugehörigen Eingabefeld genau eine Bedingung der Form

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

repräsentiert. Die Regel, die im Eingabebereich von Abb. 4.9 dargestellt wird, lautet also :

$$\text{IF } \textit{winkel} = \textit{NE} \text{ AND } \textit{wi\_geschw} = \textit{ZE} \text{ THEN } \textit{kraft} = \textit{NM}$$

In dieser Regel werden in der Prämisse nicht alle Eingabevariablen benutzt, sondern nur zwei. Die Toggle-Buttons, die keine Bedingung darstellen, werden so eingestellt, daß sie den Text *leer* anzeigen. Das dazugehörige Eingabefeld darf dann keinen Eintrag enthalten.

Prämisse und Konklusion einer Regel müssen je mindestens eine Bedingung enthalten. Die Toggle-Buttons sind so ausgelegt, daß in der Prämisse nur Eingabe- und in der Konklusion nur Ausgabevariable ausgewählt werden können. Falls nur eine Variable des entsprechenden Typs existiert, entfällt außerdem die Möglichkeit, diesen Button auf *leer* zu setzen.

Bei der Auswahl einer bestimmten Regel wird diese automatisch im Eingabebereich dargestellt. Die Auswahl erfolgt entweder durch doppeltes Anklicken im Auswahlfeld oder durch Anklicken der Regel in der Regeltabelle. Eine ausgewählte Regel wird immer sowohl in dem Auswahlfeld als auch in der Tabelle graphisch hervorgehoben.

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

Der Button delete löscht die Regel, die im Eingabebereich dargestellt wird. Um eine bestimmte Regel aus der Regelbasis zu entfernen, wird diese also zuerst mit der Maus ausgewählt, und dann der Button delete betätigt. Der Button new macht alle Änderungen rückgängig, die seit dem letzten Setzen im Eingabebereich gemacht wurden.

Die Regeltabelle

In der obersten Zeile und in der linken Spalte der Tabelle sind die Namen aller Fuzzy-Mengen von je einer Eingabevariablen dargestellt. Die Namen der entsprechenden Variablen werden durch zwei Toggle-Buttons neben der Tabelle angezeigt. Die restlichen Tabellenfelder enthalten als Einträge die Namen von Fuzzy-Mengen einer Ausgabevariablen.

Jeder dieser Einträge entspricht 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-Mengen in der Konklusion durch den eingetragenen Namen.

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

- Welche Variablen werden in der Tabelle dargestellt ?
- Für die Variablen, die nicht in der Tabelle dargestellt werden, muß eine Belegung mit Fuzzy-Mengen angenommen werden.

'Eine Belegung für eine Variable annehmen' heißt dabei, daß nur Regeln betrachtet werden, in der diese Variable mit einer bestimmten Fuzzy-Menge 'belegt' ist.

Das erste Problem, die Auswahl der darzustellenden Variablen, wird für die Eingabe- und Ausgabevariablen auf unterschiedliche Art und Weise gelöst. Für die Eingabevariablen stehen zwei Toggle-Buttons zur Verfügung, durch die die Variablen bestimmt werden können, deren Fuzzy-Mengen in den Zeilen bzw. Spalten eingetragen werden ,siehe Abb. 4.9. Die Auswahl der Ausgabevariablen erfolgt über ein

Menü, das in der linken oberen Ecke der Tabelle erscheint, wenn mehr als eine Variable dieses Typs vorhanden ist. In Abb. 4.9 existiert lediglich eine Ausgabevariable, und das Menü ist somit nicht zu sehen.

Die Belegung der nicht in der Tabelle dargestellten Variablen erfolgt ebenfalls durch ein Menü, dieses ist in Abb. 4.9 über der Tabelle zu sehen. Dieses Menü bzw. diese Menüleiste enthält einen Menüpunkt für jede Variable. Sie können dann entweder mit einer Fuzzy-Menge belegt werden oder werden als *leer* definiert. Wenn nur zwei Eingabe- oder nur eine Ausgabevariable existieren, erscheinen die Variablen des entsprechenden Typs nicht im Menü.

Für die Darstellung der Regeln in der Tabelle dient die Belegung der nicht dargestellten Variablen als Maske. Das heißt, wenn eine Variable *var* mit der Fuzzy-Menge *fuz* belegt ist, werden in der Tabelle nur Regeln angezeigt, die die Bedingung

$$var = fuz$$

enthalten. Wenn Regeln dargestellt werden sollen, in denen eine bestimmte Variable nicht vorkommt, muß diese mit *leer* belegt werden.

Innerhalb der Tabelle wird immer das Tabellenfeld, in dem sich der Mauszeiger befindet, durch einen dickeren Rahmen markiert. Außerdem erscheint rechts oben in diesem Feld ein Menü, durch das Regeln definiert oder gelöscht werden können, dieses Menü ist in Abb. 4.9 ausgeklappt zu sehen. Der erste Eintrag des Menüs ist der Name der Ausgabevariablen, die in der Tabelle dargestellt wird, er dient lediglich der Information und kann nicht ausgewählt werden. Es folgen die Namen der einzelnen Fuzzy-Mengen und ein Menüeintrag zum Löschen von Regeln. Wenn eine der Fuzzy-Mengen gewählt wird, erscheint dessen Name in dem Tabellenfeld, und es wird eine neue Regel definiert, die diesem Tabelleneintrag entspricht. Durch den Untermenüpunkt *löschen* wird die Regel gelöscht, die im jeweiligen Tabellenfeld dargestellt ist.

Nach dem Anklicken einer in der Tabelle dargestellten Regel mit der Maus, gilt diese als ausgewählt und wird ebenfalls in der Regeltafel und im Eingabebereich markiert bzw. dargestellt.

4.5 Definition und Start einer Anwendung

Um einen Prozeß mit Hilfe von NEFCON-I zu regeln, muß das Programm, das diesen Prozeß simuliert, näher definiert werden. Das heißt, es muß festgelegt werden, wie der Name des Programms lautet, und in welcher Reihenfolge die Eingabe- bzw. Ausgabevariable ein- bzw. ausgegeben werden. Zu diesem Zweck existiert ein eigenes Eingabefenster, das im Hauptfenster, zu sehen in Abb. 4.10, durch den Button define aufgerufen wird.

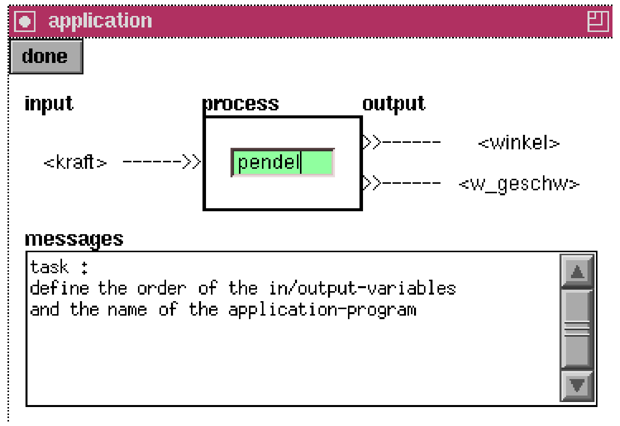

Abbildung 4.10: Fenster zur Definition der Anwendung

Das Fenster für die Definition des Anwendungsprogramms, Abb. 4.10, enthält ein Eingabefeld für den Namen des Programms und zwei Gruppen von Toggle-Buttons, die die Reihenfolge der Ein/Ausgabevariablen festlegen.

Bei der Angabe des Programmnamens ist zu beachten, daß der gesamte Pfadname angegeben werden muß, unter dem das Programm zu finden ist.

Die Toggle-Buttons geben, von oben nach unten ausgewertet, die Reihenfolge der Ein/Ausgabevariablen an. Die Bezeichnungen 'Eingabe' und 'Ausgabe' sind in diesem Fenster aus der Sicht des Anwendungsprogramms zu verstehen. Eingabevariablen aus der Sicht des Reglers sind Ausgabevariablen für den Prozeß und umgekehrt. Das Beispiel inAbb. 4.10 stellt ein System mit vier Eingabe- und einer Ausgabevariablen dar.

Um das Programm zu starten, wird im NEFCON-I-Hauptfenster der Button start gedrückt.

Die Kommunikation zwischen NEFCON–I und der Anwendung geschieht über definierte Nachrichten, die über den Standardein- und Ausgabekanal verschickt werden. Die einzelnen Nachrichten werden durch ein `\nl` getrennt. Die Anwendung schickt das ASCII Zeichen `r`, falls sie neu gestartet wird und getrennt durch jeweils ein `\nl` die Meßwerte (ASCII–codiert) und wartet dann auf die Antwort des Reglers. Der Regler schickt den Stellwert an die Anwendung. Je nachdem, ob die Anwendung die Nachricht `t` (Trigger) versteht, kann der Regler mit seiner Hilfe die Anwendung veranlassen, mit einem zufälligen Wert neu zu starten. Dazu muß selbstverständlich die Trigger–Option (siehe Abbildung 4.6) eingeschaltet sein. Die Anwendung signalisiert einen Fehlschlag oder einen erneuten Start durch die Nachricht `t`. Das Zeichen `EOF` signalisiert der Anwendung, daß NEFCON–I beendet wird.

4.6 Die Regleroberfläche

Die Regleroberfläche wird aus dem NEFCON Hauptfenster mit dem [surface] Button gestartet. Das Erscheinungsbild der Oberfläche kann durch die mit dem [protokol] Button erreichbaren Optionen 4.5 festgelegt werden.

Der obere Teil des Fensters 4.11 enthält den sogenannten *graphischen Protokollbereich*. In diesem Bereich können Systemwerte und die Steuerregeln dargestellt werden.

Für die Systemwerte kann unter zwei Darstellungsarten gewählt werden: graphisch, wie im Bild 4.11 oder numerisch wie im Bild 4.12.

Die Steuerregeln können auf drei Arten dargestellt werden:

1. Als Tabelle (`as board`), wobei alle Regeln textlich als `IF-THEN`–Terme dargestellt werden, und diejenige mit dem höchsten Treffergrad hervorgehoben wird (siehe Bild 4.13).

2. Als Info–Tabelle (`info`), wobei hier alle Regeln, die aktiviert wurden als `IF-THEN`–Terme zusammen mit den Regelerfüllungsgraden und dem Regelausgabewert (siehe Bild 4.14) dargestellt werden.

   Die Zeile

   $$0.98\text{:}\ \ \mathtt{Z}\ \ (0.00)\ \&\ \mathtt{Z}\ \ (0.00) \Longrightarrow \mathtt{Z}\ \ (0.00)$$

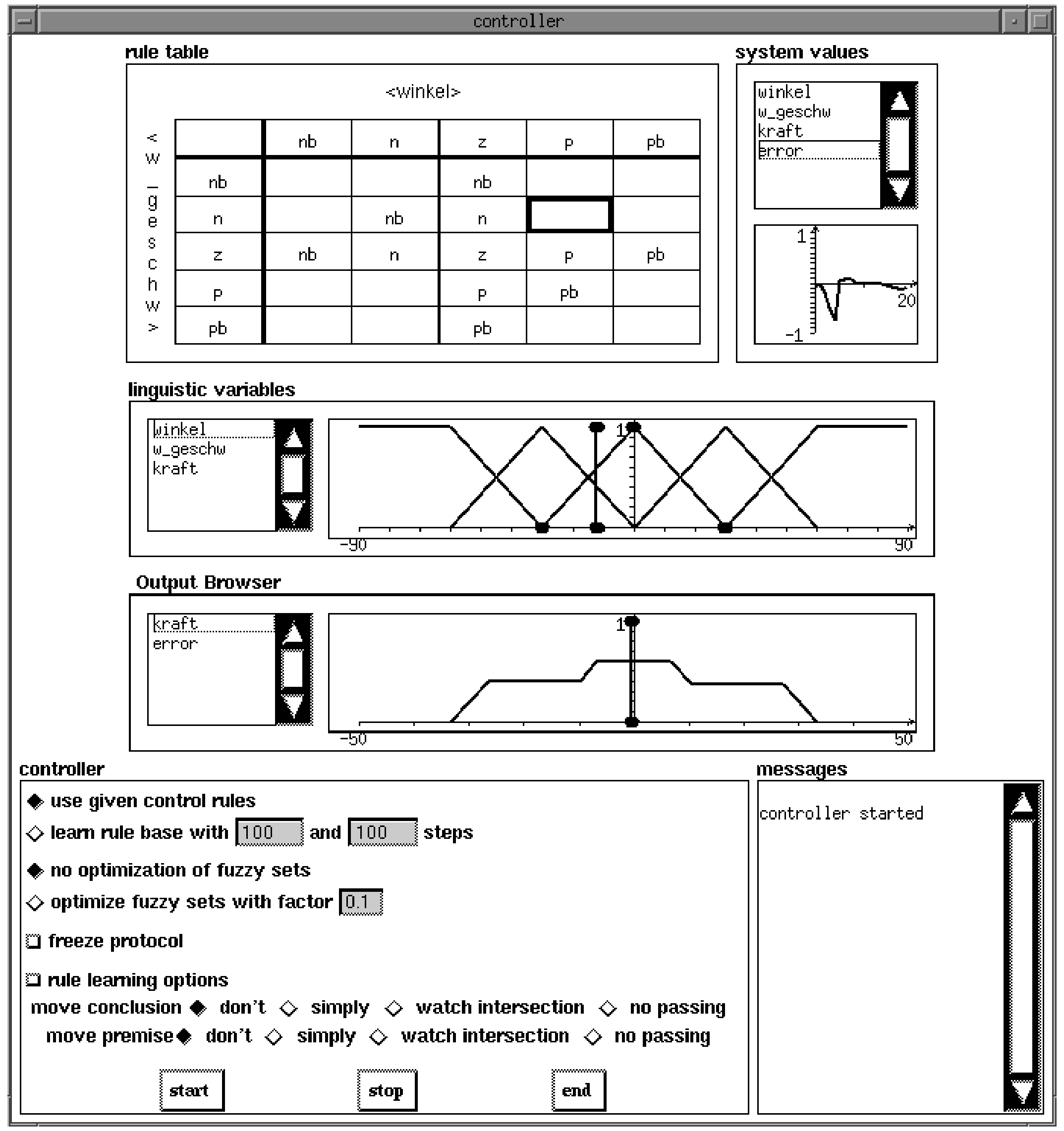

Abbildung 4.11: Die Regleroberfläche

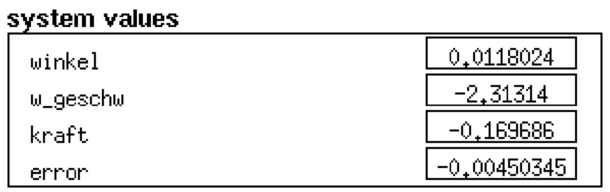

Abbildung 4.12: Numerische Darstellung der Systemwerte

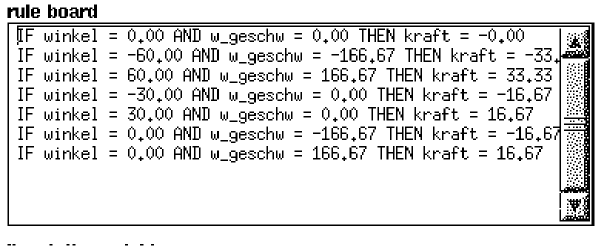

Abbildung 4.13: Darstellung der Regeln als Tabelle

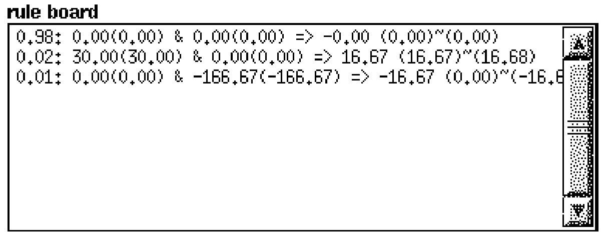

Abbildung 4.14: Darstellung der Regeln als Info–Tabelle

bedeutet, daß die Prämisse der Regel

`IF` Winkel `is Z AND` Winkelgeschwindigkeit `is Z THEN` Kraft `is Z`

den Erfüllungsgrad 0.98 hat, und daß diese Regel an dem aktuellen Regelzyklus beteiligt war. Die Zahlen in Klammern geben die Lage der Spitze der entsprechenden Fuzzy–Mengen.

3. Als Tafel (`as table`), wobei diese Darstellungsart nur bei Systemen mit zwei Meßwerten Sinnvoll ist. Auch hier wird die Regel mit dem höchsten Treffergrad hervorgehoben (siehe Bild 4.11).

Zusätzlich zu den Steuerwerten kann die Partitionierung (Protokolloption `fuzzy set representation`: $\boxed{\text{graphical}}$) einer linguistischen Variablen dargestellt werden (siehe Bild 4.11), so daß ihre Veränderung während des Lernvorgangs mitverfolgt werden kann. Für die Ausgabevariable wird auch der defuzzifizierte Steuerwert und für die Eingabevariablen der jeweilige Meßwert dargestellt. Ist eine Ausgabevariable angewählt, so wird diejenige Fuzzy–Menge hervorgehoben (durch große Punkte an den Enden), die zu der Regel mit dem höchsten Erfüllungsgrad als Regelprämisse gehört.

Um die Entstehung des Ausgabewertes mitverfolgen zu können, kann im `output browser` (Protokolloption $\boxed{\text{as graph}}$) der Graph der Ausgabe–Fuzzy–Menge für die aktuelle Ausgabevariable oder für die Variable `error` dargestellt werden (siehe Bild 4.11).

Da die grafischen Ausgabemodi sehr viel Rechenzeit kosten, sollte man sie nur bei Bedarf einschalten und ggf. mit dem Button [freeze protokol] ausschalten.

Links unten auf der Regleroberfläche befindet sich ein Schalterbereich. In diesem Bereich kann unter anderem auch der Regelvorgang gestartet ([start]), gestoppt ([stop]) oder abgeschlossen werden ([end]). Mit den Schaltern aus diesem Bereich kann auch das Lernverhalten gesteuert werden. Es kann bestimmt werden, ob und wie Regeln gelernt oder die Fuzzy–Mengen optimiert werden können.

Man kann mit Hilfe des Buttons [use given contr rules] das Regellernen und mit dem Button [no optimization of fuzzy sets] die Optimierung der Fuzzy–Mengen ausschalten und NEFCON–I als einfachen, nicht optimierenden Fuzzy–Regler benutzen.

Sollten Fuzzy–Mengen optimiert oder Regeln gelernt werden, muß einer der vorhandenen Lernalgorithmen vor dem Start des Regelvorgangs aktiviert werden. Die Algorithmen für die Optimierung der Fuzzy–Mengen werden mit dem Button [optimize fuzzy sets] eingeschaltet, wobei man im zugehörigen Eingabefeld die Lernrate festlegen kann. Falls die beiden Switches `move conclusion` und `move premise` ausgeschaltet sind wird automatisch der ursprüngliche NEFCON–Lernalgorithmus (siehe Definition 1.3) aktiviert. Sobald einer der beiden Schalter nicht ausgeschaltet ist, wird der neue Lernalgorithmus durch Veränderung der Lage der Fuzzy–Mengen (siehe Definition 2.2) eingeschaltet und der ursprüngliche ausgeschaltet. Sollten beide Algorithmen gleichzeitig benutzt werden, was eigentlich nur bei der ausschließlichen Verwendung von monotonen Fuzzy–Mengen in den Regelkonklusionen sinnvoll ist, muß in dem Optionenfenster, das aus dem Hauptfenster (Abbildung 4.2 über den [options] Button erreichbar ist, die Option `move and stretch` eingeschaltet werden.

Für den neuen Lernalgorithmus durch Veränderung der Lage der Fuzzy–Mengen stehen drei verschieden restriktive Verschiebemethoden zur Verfügung. `simply` besitzt keine Restriktionen. `watch intersection` ist sehr restriktiv, denn sie erhält die Schnitte zwischen benachbarten Fuzzy–Mengen. `no passing` verbietet, daß Fuzzy–Mengen einander überholen. Die Restriktiven Verschiebeverfahren werden von den Lernverfahren nicht benötigt, können aber zum experimentieren benutzt werden.

Das Regellernen wird mit dem Button learn rule base eingeschaltet. In den Eingabebereichen rechts des Button werden die Anzahlen der Lernzyklen für die Lernphasen eingegeben.. Es ist zu beachten, daß die Lernphasen bei den Lernverfahren unterschiedliche Bedeutungen haben, Die zur Verfügung stehenden Lernmethoden werden in dem Optionenfenster (Abbildung 4.6) eingegeben.

Die Art des Regellernens muß schon vor dem Start der Regleroberfläche geschehen. Dazu benutze man das aus dem Hauptfenster erreichbare Optionenfenster (siehe Bild 4.6).

Je nach Art des Regellernverfahrens werden unterschiedliche Lernoptionen angeboten. Man kann sie durch anklicken des `rule learning options` Buttons erreichen.

Optionen für das Erlernen von Regeln

- **Ursprüngliche NEFCON Regellernverfahren**

  Dieses Regellernverfahren gliedert sich in zwei Phasen. Die erste Phase startet mit einer Regelbasis, die aus allen Regeln besteht, die sich aus allen Kombinationen von Fuzzy–Mengen der linguistischen Ein- und Ausgabevariablen bilden läßt. In der ersten Lernphase werden Regeln aus der Regelbasis gelöscht die ein Vorzeichen haben, das verschieden zum Vorzeichen des optimalen Stellwertes ist. In der zweiten Lernphase wird versucht, aus den verbliebenen Regeln die passendste auszuwählen. Außerdem werden, Regeln,

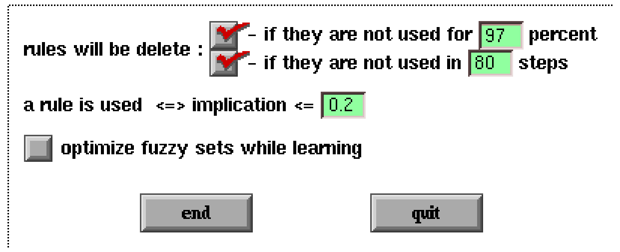

Abbildung 4.15: Optionen zum NEFCON original Regellernen

  die für falsch oder überflüssig gehalten werden aus der Regelbasis gelöscht. Dazu werden die Regellernoptionen (siehe Abbildung 4.15) berücksichtigt. Mit Hilfe der Option `optimize fuzzy sets while learning` kann zusätzlich der Optimierungsalgorithmus für Fuzzy–Mengen eingeschaltet werden.

- **Das Bottom–Up Lernverfahren**

  Beim Buttom–Up Regellernverfahren stehen zwei Lernphasen zur Verfügung. Ihre Bedeutung ist allerdings eine andere als die oben beschriebene. In der ersten Lernphase werden Regeln strukturell gelernt, beginnend mit der leeren Regelbasis (siehe Abschnitt 3.3). In der zweiten Lernphase wird versucht, die Konklusionen zu optimieren.

  Als Versuch habe ich noch die Möglichkeit eingebaut, die Regelbasis vor dem Start durch Aktivieren des Buttons $\boxed{\text{preinstall}}$ im Regellernoptionenfenster vorzuinstallieren. Dies geschieht nach einem heuristischen Algorithmus, der die Regelbasis aus der Fehlerregelbasis ableitet. Falls diese Option gewählt ist, bekommen die erste und die zweite Lernphase die gleiche Bedeutung.

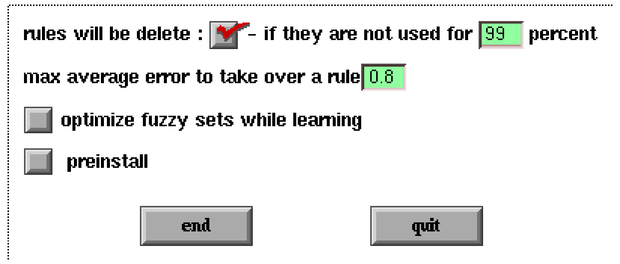

Abbildung 4.16: Optionen zum "Bottom–Up" Regellernen

  Nach Abschluß jeder Lernphase wird die Regelbasis erzeugt, wobei die Regellernoptionen (siehe Bild 4.16) berücksichtigt werden. Dies verhindert, daß überflüssige oder falsche Regeln in die Regelbasis aufgenommen werden.

  Dieses Lernverfahren geht von einer korrekten Partitionierung der linguistischen Variablen aus. Es ist daher nicht empfehlenswert es zusammen mit dem Optimierungsverfahren für Fuzzy–Mengen zu starten.

  Als bisher einziges Lernverfahren berücksichtigt dieses Lernverfahren den Lernfaktor. Dieser ist vor allem in Abhängigkeit der Berechnungsmethode für den Fuzzy–Fehler zu wählen. Bei den COA–Methoden ist er 1.0 und bei dem MOM–Verfahren ca. 0.5 zu wählen.

- **Das Partitionslernverfahren**

  Das Partitionslernverfahren gliedert sich auch in zwei Phasen, die jedoch eine andere Bedeutung haben als in den oben beschrieben Verfahren. In der ersten Lernphase werden Partitionierungen der linguistischen Variablen hergestellt. Dazu werden die mit dem Button $\boxed{\text{rule learning options}}$ erreichbaren Optionen berücksichtigt. Die Optionen geben an, wie viele Fuzzy–Mengen maximal in die Partitionierung einer gegebenen linguistischen Variable aufgenommen werden sollen, und unter welchen Umständen eine neue Variable benötigt wird. In dieser Implementierung wird eine neue Fuzzy–Menge in die Partitionierung aufgenommen, falls für einen Eingabewert der Treffergrad für keine Vorhandene Fuzzy–Menge den Wert `min score to insert a new fuzzy set` übersteigt.

Vor dem Start des Partitionslernverfahrens ist es nötig, die Steuer–Regeln zu löschen. Dies kann mit Hilfe des Deletemenüs im NEFCON–Hauptfenster erfolgen.

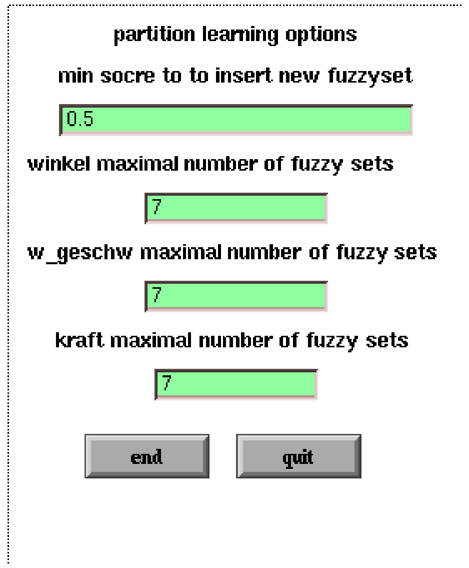

Abbildung 4.17: Optionen zum Partitionslernverfahren

4.7 Die Protokolldatei

Mit Hilfe der aus dem Protokolloptionenfenster erreichbaren Option `create a protocol file` läßt sich NEFCON–I während des Lernvorgangs dazu veranlassen, eine Protokolldatei anzulegen. Diese Datei enthält wichtige Informationen über die Steuer–Wissensbasis vor und nach dem Regelvorgang. Außerdem werden die wichtigsten Systemdaten dokumentiert.

Das Format der Protokolldatei steht im Anhang A dieser Arbeit.

Mit Hilfe einiger zum NEFCON–I–Paket gehörenden Programme lassen sich aus der Protokolldatei Daten im *gnuplot*–Format Extrahieren und mit Hilfe von *gnuplot* graphisch darstellen.

Das Programm *nfc2gpl* dient zum extrahieren von Systemwerten. Es nimmt die Eingaben von der Standardeingabe und schreibt sie in die Standardausgabe. Es wird also wie folgt gestartet:

```
nfc2gpl < protokoll.nfs > protokoll.dat
```

Die Datei `protokoll.dat` kann mit Hilfe von *gnuplot* graphisch dargestellt werden. Sollte zum Beispiel für das Pendelbeispiel die Stellwerte in Abhängigkeit der Meßwerte dargestellt werden, muß in *gnuplot* einfach

```
splot ‘‘protokoll.dat’’ using 1:2:3 with points
```

eingegeben werden.

Das Programm *nfc2fuzgpl* extrahiert die Steuerwissensbasen. Es nimmt die Eingaben von Standardeingabe und wird beispielsweise wie folgt gestartet:

```
nfc2fuzgpl < protokoll.nfc
```

Dabei werden neue Dateien angelegt, die nach den Namen der linguistischen Variablen benannt sind. Dateien für linguistische Variablen, die direkt nach dem Start des Reglers angelegt wurden, bekommen das Appendix `_org` und Dateien, die nach dem Stop des Reglers angelegt wurden, tragen das Appendix `_lrn`. Alle hier erzeugten Dateien haben das Appendix `.dat`. bekommen.

Die so extrahierten Partitionierungen der linguistischen Variablen können mit Hilfe von *gnuplot* graphisch dargestellt werden. Zum Beispiel würde die Partitionierung der linguistischen Variablen *winkel* nach Abschluß des Lernverfahrens von *gnuplot* aus mit

```
plot ‘‘winkel_lrn.dat’’ with lines
```

angezeigt werden. Weitere Informationen über die Darstellungsmöglichkeiten von *gnuplot* entnehme man am besten den Manpages.

Um den Verlauf der durch die Steuer–Wissensbasis definierten Regelfunktion graphisch darstellen zu können, ist das einfache Programm `datagen` mit in das NEFCON–I–Paket integriert worden. `datagen` ist ein Anwendungsprogramm, das nur Paare von Meßwerten an NEFCON–I schickt und den Stellwert ignoriert.

`datagen` eröffnet ein Fenster (siehe Abbildung 4.18), in dem die passenden Definitionsbereiche der Eingabevariablen definiert werden können. Nach dem Start geht `datagen` diese Definitionsbereiche in zwei ineinander geschachtelten Schleifen durch, wobei die Werte jeweils um die ebenfalls einzugebende Schrittweite erhöht werden. Um einen Graphen der Regelfunktion zu erhalten sollte der NEFCON–I mit ausgeschaltetem Lernalgorithmen, ausgeschaltetem Trigger, ausgeschalteter Protokollausgabe im Kontrollerfenster und eingeschaltetem Dateiprotokoll betrieben werden.

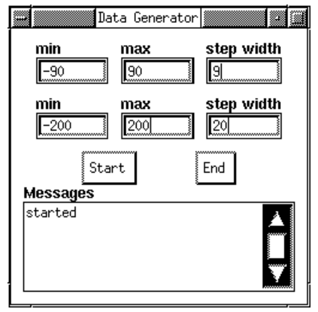

Abbildung 4.18: Die Anwendung `datagen`

NEFCON–I erzeugt dann die Protokolldatei `protokoll.nfc` (es sei denn man hat einen anderen durch Ressourcen vorbestimmt), die mit `nfc2gpl < protokoll.nfc > protokoll.dat` in das *gnuplot*–Format umgewandelt wird und von *gnuplot* aus mit `set param`
`splot ''protokoll.dat'' using 1:2:3 with lines`
dargestellt werden kann. Man erhält einen 3–dimensionalen Graphen der Regelfunktion.

4.8 Das Doppelpendel

Das inverse Doppelpendel ist eine neue Anwendung für ein zu regelndes dynamisches System. Beim inversen Doppelpendel wird versucht, die beiden durch ein Gelenk miteinander verbundenen Stäbe, die auf einem Wagen befestigt sind, im Schwerefeld vertikal, nur durch Bewegung des Wagens auszubalancieren. Das Schema der Versuchsanordnung zeigt die Abbildung 4.19. Die Stäbe haben die Länge l_1 für den unteren und der Länge l_2 für den oberen Stab. Sie werden als masselos angenommen. Die Massen m_1 und m_2 konzentrieren sich punktförmig an den

Enden der Stäbe.

Der Wagen selbst hat die Masse M. Als Stellwert für dieses System gilt die Kraft F, die auf den Wagen entlang der horizontalen Achse wirkt. Als Meßwerte können die Winkel der Stäbe zur vertikalen Achse φ_1 und phi_2 und ihre Ableitungen (Winkelgeschwindigkeiten) $\dot{\varphi}_1$ und $\dot{\varphi}_2$ verwendet werden. Die Reibung wird bei dieser Anordnung vernachlässigt.

Als Einheiten werden für die Kraft Newton und für die Winkel Grad bzw. Grad pro Sekunde für die Winkelgeschwindigkeiten genommen. Die Längen der Stäbe werden in Meter und die Massen in Kilogramm angegeben. Die Konstante g hat den üblichen Wert von 9.81 $\frac{kg \cdot m}{sec^2}$

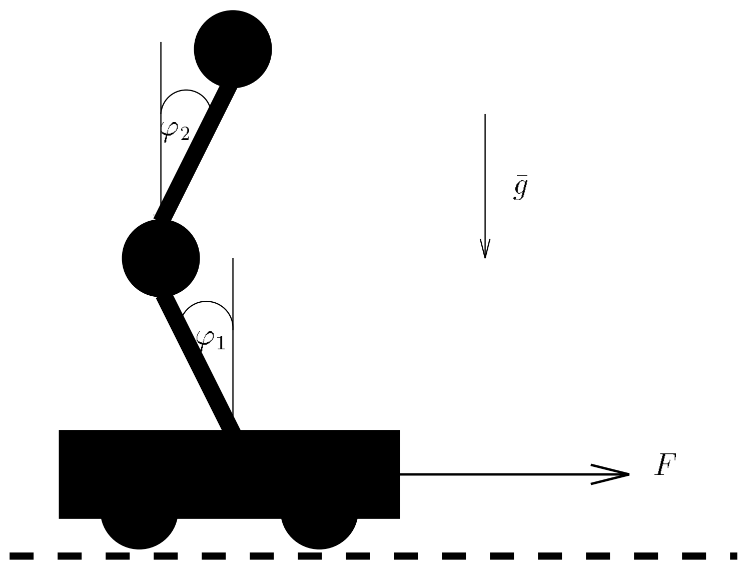

Abbildung 4.19: Das Doppelpendel

Unter der Verwendung des D'Alembertschen und der Lagrandschen Differentialgleichungen (siehe [Goldstein]) habe ich die untenstehende Bewegungsgleichungen für das inverse Doppelpendel aufgestellt. Die aus dem Runge–Kutta Verfahren resultierende Lösung dieser Gleichungen wird für die Implementierung einer Simulation benutzt.

$$\ddot{x}_M = \frac{F - l_1(m_1+m_2)\left[\ddot{\varphi}_1 \cos\varphi_1 - \dot{\varphi}_1^2 \sin\varphi_1\right] - m_2 l_2 \left[\ddot{\varphi}_2 \cos\varphi_2 - \dot{\varphi}_2^2 \sin\varphi_2\right]}{(m_1+m_2+M)}$$

$$\ddot{\varphi}_1 = \frac{(m_1+m_2)\left[g\sin\varphi_1 - \ddot{x}_M\cos\varphi_1\right] - m_2 l_2\left[\ddot{\varphi}_2\cos(\varphi_1-\varphi_2) + \ddot{\varphi}_2^2\sin(\varphi_1-\varphi_2)\right]}{(m_1+m_2)l_1}$$

$$\ddot{\varphi}_2 = \frac{g\sin\varphi_2 - l_1\ddot{\varphi}_1\cos(\varphi_1-\varphi_2) + \sin(\varphi_1-\varphi_2)l_1\dot{\varphi}_1^2 - \ddot{x}_M\cos\varphi_2 +}{l_2}$$

Das Simulationsprogramm für das inverse Doppelpendel wird durch den Befehl `pendel2` gestartet. Die Anwendung gibt ihre Meßwerte über den Standardausgabekanal aus und liest den Stellwert über der Standardeingabekanal. Alle Nachrichten müssen mit dem Zielenendszeichen '$\backslash nl$' abgeschlossen werden. Das Programm versteht das Triggersignal '`t`' und sendet beim Start und beim Anstoßen mit zufälligen Winkelwerten die Restart Nachricht '`r`'.

Damit das Triggersignal richtig interpretiert und nicht nur überlesen wird, muß im Hauptfenster des Simulationsprogramms der Button $\boxed{\text{trigger}}$ eingeschaltet werden (siehe Abbildung 4.20).

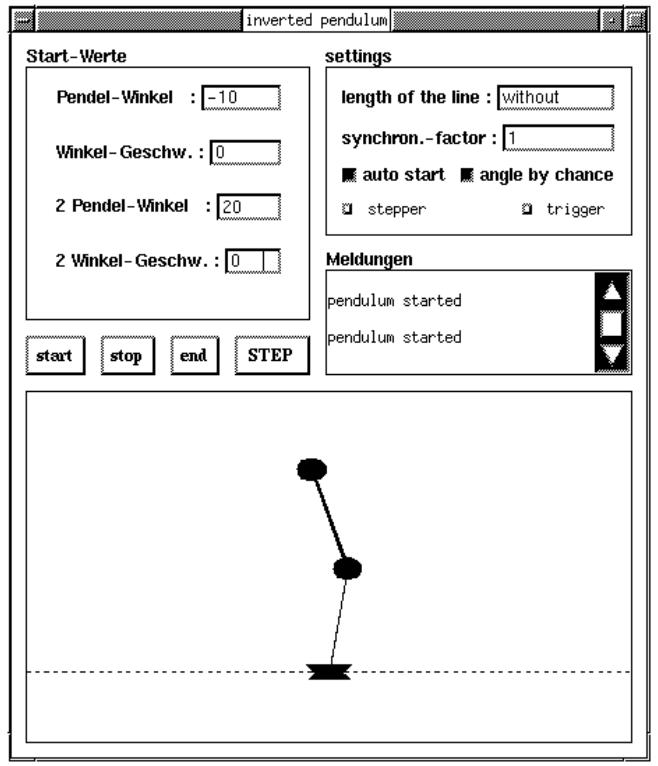

Abbildung 4.20: Das Simulationsprogramm für das inverse Doppelpendel

In der Abbildung 4.20 wird die Benutzeroberfläche der Doppelpendelsimulation dargestellt. Im unteren Bereich des Fensters wird das inverse Doppelpendel graphisch animiert dargestellt, und die Strecke, auf der sich das Pendel bewegt, wird durch eine gestrichelte Linie markiert. Die Strecke kann als endlos angesehen werden. Das System stellt vier Meßgrößen zur Verfügung. Sie können in den dafür vorgesehenen Eingabefeldern als Startwerte voreingestellte werden. Das System startet mit zufälligen Werten für die Winkel, falls der Button angle by chance eingeschaltet wird. Es wird in zufälligen Zeitabständen immer wieder mit einem zufälligen Wert gestartet es sei denn, die Zeitsteuerung wird durch das externe Triggersignal übernommen.

Es besteht die Möglichkeit, die Regelstrecke zu begrenzen. Dazu muß ihre Länge in das Eingabefeld `length of the line` eingetragen werden. Wird in dieses Eingabefeld `without` eingetragen, so wird die Strecke als endlos betrachtet. Da das Programm in jetziger Version keine Positionsdaten für den Wagen ausgibt, ist es nicht sinnvoll, eine beschränkte Strecke zu wählen.

Um den Systemwerteverlauf besser beobachten zu können, sieht das Simulationsprogramm eine Schritt-Option vor. Sie wird durch den Button stepper eingeschaltet. Bei eingeschaltetem Stepper sendet die Simulation ein Meßwertetupel, empfängt den Stellwert, berechnet ihren neuen Zustand und wartet auf die Betätigung des Step Buttons.

Der Syncronisationsfaktor `syncron. factor` gibt an, wie oft der Zustand des Pendels zwischen zwei Meßwerten berechnet werden soll. Beim ersten Mal wird die eingelesene Kraft zur Zustandsberechnung herangezogen. Bei den darauffolgenden Berechnungszyklen wird die Kraft, die auf den Wagen einwirkt als Null angesehen. Die Erhöhung des Syncronisationsfaktors bewirkt eine Steigerung der Geschwindigkeit des simulierten Pendels relativ zur Geschwindigkeit des Reglers.

Die Simulation des inversen Doppelpendels ist als Testbeispiel für das NEFCON–I System entwickelt worden. Zur Zeit ist mir aber keine Fehler–Wissensbasis bekannt, mit deren Hilfe eine Steuer–Wissensbasis gelernt werden könnte. Für diese Anwendung scheint eine Regelstrategie nötig zu sein, die mit einem einfachen Fuzzy–Regler nur mit äußerstem Aufwand zu bewerkstelligen ist. Das einzige mir bekannte Fuzzy–System für das inverse Doppelpendel besteht aus einem kaskadierten Fuzzy–Regler, der aus zwei gewöhnlichen Reglern besteht, die jeweils für die Ausregelung eines der Stäbe zuständig sind. Die Strategie besteht nun darin, als

Regelungsziel für den Regler für den oberen Stab den Winkel Null zu erreichen. Der Regler für den unteren Stab bekommt die Aufgabe, dem oberen Stab zu folgen und den gleichen Winkel für den unteren Stab zu erreichen.

Eine solche Strategie ist auf das NEFCON–I System, das aus einem einzigen Fuzzy–Regler für die Steuerung besteht, nicht übertragbar. Das NEFCON–I müßte dazu dahingehend modifiziert werden, daß man mehrere NEFCON Systeme hintereinander schalten könnte, die dann, ihr eigenes Lernziel verfolgend, die Regelstrategie erlernen.

Kapitel 5

Implementierung

Die Implementierung des NEFCON–I Paketes ist ursprünglich von Hermann–Josef Diekgerdes im Rahmen seiner Diplomarbeit [Diekgerdes, 1993] vorgenommen worden. Die hier vorgestellte Erweiterung des Paketes basiert auf dieser Implementierung. Die wesentlichen Änderungen beschränken sich auf die Erweiterungen und Umgestaltungen der graphischen Bedieneroberfläche, sowie auf das Hinzufügen neuer Lernalgorithmen.

Die Erweiterungen hatten zufolge, daß die Basisklassen erweitert und aufgeteilt werden mußten. Dabei habe ich darauf geachtet, das ursprüngliche Konzept, die grundlegenden Datenstrukturen und die Klassenhierarchie so weit wie möglich zu erhalten.

Die Beschreibung der Implementierung von Hermann–Josef Diekgerdes aus dem vierten Kapitel seiner Diplomarbeit [Diekgerdes, 1993] gilt grundsätzlich immer noch. Die wesentlichen Unterschiede werden im folgenden aufgezeigt.

Die Abbildung 5.1 zeigt die für die Implementierung von NEFCON–I definierten Module. Ein Modul (z.B. `contr`) entspricht einem Paar von C++ Quelldateien bestehend aus der Headerdatei (z.B. `contr.h`) und der Codedatei (z.B. `contr.c`). Die Module sind in Gruppen zusammengefaßt und geben in ihrer Gliederung eine hierarchische Ordnung wieder. Module aus unteren Ebenen bauen auf Modulen aus oberen Ebenen auf. Auch innerhalb der Gruppen bauen Module, die weiter unten stehen auf Modulen, die weiter oben stehen, auf.

Die wichtigste Änderung an der Programmstruktur im Vergleich zur ursprünglichen Implementierung von Hermann–Josef Diekgerdes ist durch das Hinzufügen

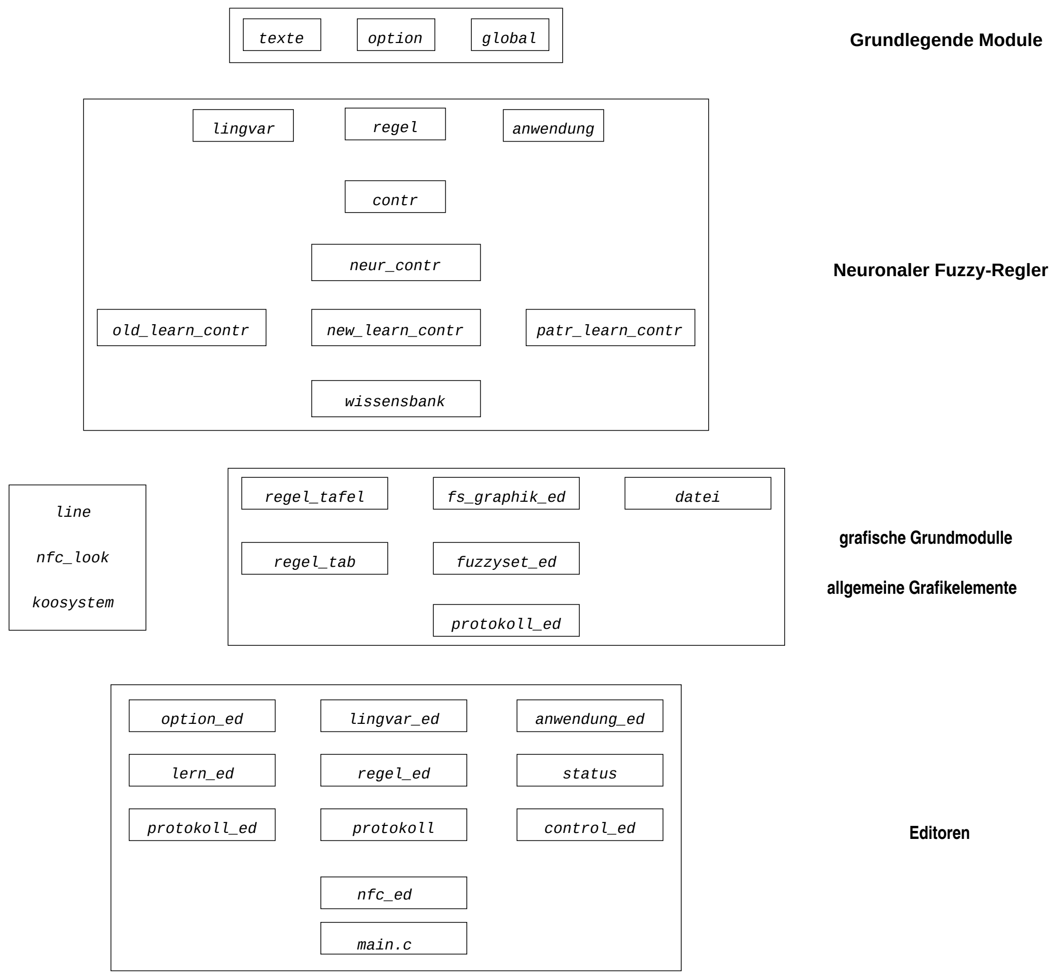

Abbildung 5.1: Die Modulhierarchie von NEFCON–I

neuer Lernalgorithmen und Controllertypen notwendig geworden. Das Modul `controller` aus der ursprünglichen Implementierung ist in zur Zeit fünf neue Module unterteilt worden, wobei ich darauf geachtet habe, daß das neue Konzept das Einfügen weiterer Lernalgorithmen möglichst leicht macht.

Jeder neue Lernalgorithmus muß in die Klassenhierarchie, die in Abbildung 5.2 dargestellt ist, eingebunden werden.

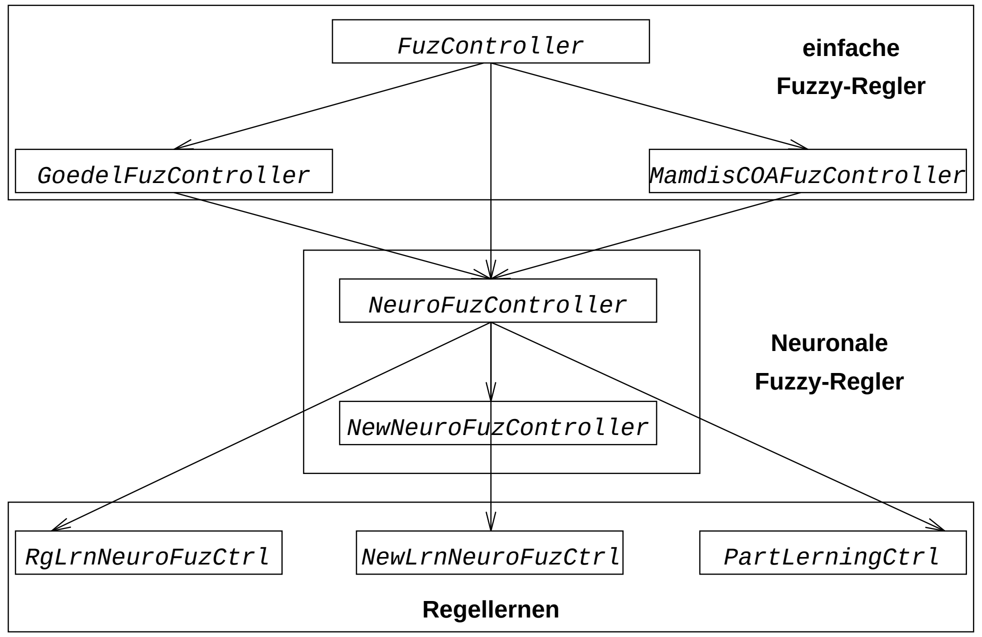

Abbildung 5.2: Klassenhierarchie des Neuronalen Fuzzy–Reglers

Die Pfeile zeigen die C++ Vererbungshierarchie. Man kann die einzelnen Klassen in Gruppen unterteilen. Die erste, auf der alle anderen aufbauen, ist die Gruppe, die die einfachen Fuzzy–Regler enthält. Diese Regler habe noch keine Lernfähigkeit und stellen unterschiedliche Methoden der Stellwertberechnung zur Verfügung. Die Vererbung in diesem Ausschnitt der Klassenhierarchie ist wegen der Mehrfachvererbung in der Gruppe der einfachen Fuzzy–Regler *virtual*.

Von allen einfachen Fuzzy–Reglern wird die Grundklasse der Neuronalen Fuzzy–Regler `NeuroFuzController` abgeleitet. Diese Klasse ist selbst eine Instanz eines der einfachen Fuzzy–Regler, die zusätzlich noch als privates Objekt eine Instanz eines zweiten Fuzzy–Reglers besitzt, der zur Fuzzy–Fehler–Berechnung herangezogen wird. Die Klasse `NeuroFuzController` benutzt den ursprünglichen Optimierungsalgorithmus für Fuzzy–Mengen als Lernmethode. Von dieser Klasse werden weitere

Neuronale Fuzzy–Regler und Regler mit der Möglichkeit zum Regellernen abgeleitet. Bei einer solchen Implementierung ist es natürlich unmöglich, den neuen Optimierungsalgorithmus, der ein Objekt der Klasse `NewNeuroFuzController` ist, zusammen mit einem der Regellernverfahren zu benutzen. Um dieses zu ermöglichen, müßten die Regellernklassen zusätzlich die Eigenschaften von `NewNeuroFuzController` erben, die dann gegebenenfalls vom Lernalgorithmus mitbenutzt werden könnten.

Das Prinzip des Einfügens neuer Lernalgorithmen ist also ziemlich einfach. Um einen neuen Optimierungsalgorithmus für Fuzzy–Mengen einzufügen, muß eine für diesen Algorithmus eigene Klasse von der Klasse `NeuroFuzController` abgeleitet werden und nur die virtuelle Methode `_optimieren(FuzzyTyp*)` neu geschrieben werden. Auf ähnliche Art- und Weise werden neue Regellernmethoden und Stellwertberechnungsmethoden implementiert. Natürlich müssen noch in den Modulen `option`, `option_ed`, `controller_ed` und `nfc_ed` Möglichkeiten implementiert werden, die neuen Klassen aus der Bildschirmoberfläche heraus einzuschalten. Dazu folgt man am besten dem vorgegenem Schema. Sollten die neuen Möglichkeiten auch als X–Ressourcen gesetzt werden können, muß auch in der Datei `main.c` die Resource eingetragen werden. Zur Verwaltung von Ressourcen und Styles lese man am besten [Nye, 1993] und [Linton et al. 1992].

Abschlußbemerkungen

Der Einsatz von Fuzzy–Technologien erfreut sich in der Regelungstechnik zunehmender Beliebtheit. Um den Entwurfsvorgang eines Fuzzy–Reglers zu erleichtern, werden viele Entwicklungs–Tools mit graphischer Bedieneroberfläche angeboten. Sie helfen jedoch nicht, die konzeptionellen Probleme, die sich durch den Einsatz heuristischer Methoden beim Entwurf und der Optimierung eines Fuzzy–Reglers ergeben, zu beseitigen.

Das in dieser Arbeit vorgestellt NEFCON–Modell ist im Stande, bei der Optimierung und dem Entwurf eines Fuzzy–Reglers wichtige Hilfestellung zu leisten. Durch die Verknüpfung der Methoden aus den Bereichen der Neuronalen Netze und der Fuzzy–Systeme besitzt ein NEFCON–System sowohl die Lernfähigkeit als auch die Transparenzeigenschaft.

Das ursprüngliche NEFCON–Modell (siehe [Nauck 1993] und [Diekgerdes, 1993]) ist trotz seiner Einschränkungen sehr erfolgreich gewesen. Die in dieser Arbeit vorgestellten Erweiterungen des Modells verbessern die Lerneigenschaften erheblich. Der neue Optimierungsalgorithmus für Fuzzy–Mengen schafft es in kurzer Zeit, auch sehr fehlerhafte Partitionierungen zu optimieren. Der "Bottom–Up"–Regellernalgorithmus, der auf dem neuen Optimierungsalgorithmus für Fuzzy–Mengen basiert, findet beinahe Optimale Regelbasen.

Die in der vorliegenden Arbeit vorgestellten Erweiterungen schöpfen die in dem NEFCON–Modell steckenden Erweiterungsmöglichkeiten noch nicht aus. Für viele Regelprobleme –wie z.B. für das inverse Doppelpendel– ist eine Regelstrategie notwendig. Solche Strategien lassen sich nur durch kaskadierte Fuzzy–Regler nachbilden. Die Möglichkeit der Hintereinanderschaltung widerspricht nicht dem NEFCON–Modell. Sie ließe sich einfach durch Hintereinanderschaltung zweier oder mehrerer NEFCON–Systeme, bei denen die Lernalgorithmen parallel und unabhängig voneinander laufen und möglicherweise die Ausgaben des einen als zusätzliche Eingeben seines Nachfolgers genommen werden, realisieren. Offensichtlich würde eine solche Erweiterung keine Erweiterung des formalen Modells nach sich ziehen. Es müßte nur die implementierte Entwicklungsumgebung NEFCON–I dementsprechend erweitert werden.

Zusätzlich ließen sich für das NEFCON–Modell neue Lernalgorithmen finden, die nicht nur eine eingeschränkte Auswahl von Parametern der Fuzzy–Mengen wie nur die Breite oder nur die Lage verändern können, sondern alle zugleich. Denkbar wäre auch der Einsatz von anderen Formen von Fuzzy–Mengen. Die Lernalgorithmen sind zur Zeit vor allem für den Einsatz mit dreieckförmigen oder monotonen Fuzzy–Mengen gedacht.

Ein weiterer Punkt, in dem Erweiterungen und Optimierungen des Modells angestrebt werden könnten, ist der Informationsgehalt der Fehler–Wissensbasis. Man könnte versuchen, Lernalgorithmen zu finden, die auf die Angabe des Vorzeichens des Optimalen Stellwertes verzichten. Erreicht werden könnte dies durch die zusätzliche Einführung eines adaptiven (Fuzzy–) Kritikernetzes, das die Fehler–Wissensbasis trainiert oder den optimalen Stellwert erlernt.

Kapitel 6

Anhang

Anhang A: Installation

NEFCON–I benötigt zum Laufen ein UNIX und X–Window System mit entsprechenden Libraries, sowie das InterViews Entwicklungssystem, bestehend aus der Library `libIV`, den dazugehörigen Headerdateien und dem Makro `ivmkmf` zur automatischen Erstellung von Makefiles. NEFCON–I müßte auf den meisten die obigen Voraussetzungen erfüllenden UNIX–Systemen lauffähig sein. Ausprobiert wurde es auf SUN–Workstations und auf LINUX–Systemen.

Zur Übersetzung des NEFCON–Paketes benötigt man einen C++–Compiler. Das Paket wurde Versuchsweise mit den Compilern CC-3.0 von SUN und g++ in den Versionen 2.5.8 und 2.6.0 übersetzt. Der g++ hat ab der Version 2.6.0 die trigonometrischen Funktionen als '`double`', definiert was zu vernachlässigbaren Warnings bei der Übersetzung führt.

Das NEFCON–I Paket liegt in einer mit dem tar–Programm von gnu gepackten und komprimierten Version mit dem Namen `nefcon.tgz` vor. Es wird mit Hilfe des Befehls

```
tar xzBpf nefcon.tgz
```

ausgepackt und liegt in einer zur Übersetzung fertigen source–Version im Verzeichnis `./nefcon` vor. Dieses Verzeichnis besteht aus diversen Unterverzeichnissen:

Verzeichnis	Inhalt
`./nefcon`	Die Source des Programms zur Simulationsumgebung NEFCON–I.
`./applications`	Enthält in den Unterverzeichnissen `./pendel`, `two_stage`, `datagen` die entsprechenden Anwendungsprogramme für NEFCON–I.
`./tools`	Enthält die Sourcen zu den Hilfsprogrammen `nfc2gpl` und `nfc2fuzgpl` (siehe Abschnitt 4.7).
`./examples`	Enthält einige Beispielswissensbasen für die Anwendungen für NEFCON–I.

Die einfachste Möglichkeit, das Paket insgesamt zu übersetzen, ist das im Verzeichnis ./nefcon zur Verfügung stehende Makefile zu benutzen. Vor dem Übersetzen von NEFCON–I sollte auf jedem Fall sichergestellt werden, daß die Environment–Variable `CPU` gesetzt ist. Auf LINUX–PC's mit der `bash`–shell setzt man diese Variable mit dem Befehl

```
export CPU=LINUX
```

und auf SUN–Workstations mit z.B./ der `csh`-shell mit

```
set CPU SUN4.
```

Nach der Eingabe von `make` in dem ./nefcon Verzeichnis wird das gesamte NEFCON–I Paket mit allen Anwendungsprogrammen, den Tools und natürlich auch NEFCON–I selbst lauffertig installiert. Die ausführbare Simulationsumgebung NEFCON–I kann dann aus dem Verzeichnis ./nefcon/nefcon mit dem Befehl `nefcon` gestartet werden. Innerhalb dieses Verzeichnisses befindet sich der gesamte Quellcode von NEFCON–I. Das ausführbare Programm steht eigentlich im Verzeichnis `$CPU`. `nefcon` ist ein symbolischer Link. Das Installations-Makefile stellt noch weitere symbolische Links her. `pendel`, `datagen` sind Links auf die Binaries der Anwendungen und `examples` ist ein Link auf das von NEFCON–I standardmäßig benutztes Verzeichnis für Wissensbasen.

Sollten die zum NEFCON–I Packet gehörenden Binaries wieder gelöscht werden, so reicht es, das Makro `make clean` zu benuzten, mit dessen Hilfe der Zustand vor der Übersetzung wiederhergestellt wird.

In einigen Fällen wird es nötig sein, die Simulationsumgebung oder die Anwendungsbeispiele unabhängig voneinander zu übersetzen. Dazu muß im entsprechenden Verzeichnis die Befehlssequenz

```
ivmkmf -a; make clean; make depend; make all,
```

eingegeben werden, nachdem die CPU Variable gesetzt worden ist. Für die Übersetzung wird das zum InterViews gehörende Makro `ivmkmf` benutzt. Dieses Makro stellt aus dem Imakefile ein Makefile und legt ein Verzeichnis für die Objektdateien und das Binary an.

Das Imakefile für das `nefcon` Programm selbst hat die folgende Form:

```
XCOMM
XCOMM nefcon
EXTRA_CCFLAGS = $(EXTRA_CCFLAGS) -Wall
XCOMM EXTRA_CCFLAGS = -g -DDEBUG
AOUT = nefcon

#ifdef InObjectCodeDir

Use_libInterViews()
ComplexProgramTarget(nefcon-I)

#define Obj(file) MakeObjectFromSrc(file)

Obj(texte)
Obj(global)
Obj(option)

Obj(lingvar)
Obj(regel)
Obj(anwendung)
Obj(contr)
Obj(neur_contr)
Obj(old_learn_contr)
Obj(new_learn_contr)
Obj(part_learn_contr)
Obj(wissensbank)
```

```
Obj(nfclook)
Obj(koosystem)

Obj(datei)
Obj(fs_graphic_ed)
Obj(fuzzyset_ed)
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(option_ed)
Obj(nfc_ed)
Obj(status)
Obj(main)

#else

MakeInObjectCodeDir()

#endif
```

In dem Imakefile sind alle für das Programm benötigten Dateien aufgeführt. Wird das Programm um ein neues Modul erweitert, so muß dieses Modul, bestehend aus einer Header und einer Quelldatei, in dem Imakefile eingetragen werden. Sollten beim Übersetzen zusätzliche Parameter an den Compiler übergeben werden, so müssen sie in der Zeile `EXTRA_CCFLAGS` mit aufgeführt werden. Sollte irgendwann die Notwendigkeit bestehen, das Programm zu debuggen, so muß der Parameter `-g` an den Compiler übergeben werden. Das geschieht am besten durch Unkommentieren der Zeile `XCOMM EXTRA_CCFLAGS = -g -DDEBUG`, die dann die Form `EXTRA_CCFLAGS = -g -DDEBUG` hat.

Anahang B: Dateiformate

Das Format der Protokolldatei

***.lv**	$::=$	**lf lvs names zycle terminated lvs**
lf	$::=$	`start of the controller with learning factor:` wert des Lernfaktors
lvs	$::=$	`linguistic Variables :` $\mathbf{lv}^{\text{Anzahl der ling. Variablen}}$
lv	$::=$	Name der ling. Variablen `:` $\mathbf{fs}^{\text{Anzahl der Fuzzy–Mengen}}$
fs	$::=$	Name der Fuzzy–Menge `:` $\mathbf{iv}($`+` $\mathbf{iv}^{\text{ Anzahl der Intervalle}})$
iv	$::=$	**iva** `,` **ivb**
iva	$::=$	`a = (` $\mathbf{iva}_x$ `,` $\mathbf{iva}_y$ `)`
ivb	$::=$	`b = (` $\mathbf{ivb}_x$ `,` $\mathbf{ivb}_y$ `)`
names	$::=$	$\text{Namenstring}^{\text{Anzahl der ling. Variablen}}$
zycles	$::=$	$(\mathbf{zycle})^*$
zycle	$::=$	$(\text{Meßwerte})^{\text{Anzahl Meßwerte}}$, Stellwert, Fehlerwert
terminated	$::=$	`termination of the controller after` Anzahl der Zyklen `steps`

Literaturverzeichnis

[Aptrnix, 1992] Aptronix Inc. (1992). *Two Stage Inverted Pendulum, Application Note*. Aptronix Inc. The Fuzzy Logic Technology Company.

[Bandemer und Gottwald 1992] H. Bandemer und S. Gottwald (1992). *Einführung in Fuzzy–Methoden*. Akademie Verlag, Berlin.

[Cannon, 1967] R. H. Cannon (1967). *Dynamics in Physical Systems*. New York, Mac-Graw-Hill, 1967.

[Diekgerdes, 1993] H.-J. Diekgerdes (1993). *NEFCON–I: Entwurf und Implementierung einer Entwicklungsumgebung für Neuronale Fuzzy–Regler*. Diplomarbeit, TU Braunschweig.

[Goldstein] Goldstein. *Klassische Mechanik*.

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

[Kürsten, 1993] T. Kürsten (1993). *InterViews Tutorial Introduction Guide for Beginners*. Technischer Bericht, TU München, Institut für Informatik.

[Linton et al. 1992] M. A. Linton et al. (1992). *InterViews Reference Manual Version 3.1*. The Boeard of Trustees of the Leland Stanford Junior UNiversity.

[Mayer, 1993] S. Mayer (1992). *Einführung in InterViews*. Technischer Bericht. TU München, Institut für Informatik.

[Nauck 1993] D. Nauck (1993). *Modellierung Neuronaler Fuzzy–Regler*. Dissertationsschrift, TU–Branschweig.

[Nauck et al. 1994] D. Nauck, F. Klawonn und R. Kruse (1994). *Neuronale Netze und Fuzzy–Systeme*. Vieweg, Künstliche Intelligenz.

[Nye, 1993] A. Nye (1993). *Xlib Programming Manual for Version 11.* O'Reilly & Associates, Inc.