NEFCON-I: An X-Window Based Simulator for Neural Fuzzy Controllers

Detlef Nauck and Rudolf Kruse
Department of Computer Science
Technical University of Braunschweig
Bueltenweg 74 / 75, D-38106 Braunschweig, Germany
Tel.: +49.531.391.3155, Fax: +49.531.391.5936, Email: nauck@ibr.cs.tu-bs.de

***Abstract*— In this paper we present NEFCON-I, a graphical simulation environment for building and training neural fuzzy controllers based on the NEFCON model [6]. NEFCON-I is an X-Window based software that allows a user to specify initial fuzzy sets, fuzzy rules and a rule based fuzzy error. The neural fuzzy controller is trained by a reinforcement learning procedure which is derived from the fuzzy error backpropagation algorithm for fuzzy perceptrons [7]. NEFCON-I communicates with an external process where a dynamical system is simulated. NEFCON-I is freely available on the internet.**

I. Introduction

NEFCON is a model for neural fuzzy controllers based on the architecture of a fuzzy perceptron [7]. The system consists of 3 layers of units, and the connections between the layers are weighted by fuzzy sets [5, 6]. The learning algorithm defined for NEFCON is able to learn fuzzy sets as well as fuzzy rules.

We present the learning algorithm that uses a rule based fuzzy error measure, and that has been enhanced in comparision to past versions [5, 6]. Using the fuzzy error enables us to define a reinforcement type learning algorithm without using an adaptive critic element as it is needed by other approaches to neural fuzzy control [2]. The learning of a rule base is done by deletion of rules, that means the learning process can work online and does not need sample data as e.g. clustering approaches [3].

To test the model a graphical simulation software has been implemented, that can be used to define, train and test a NEFCON system. The software is able to communicate with a process simulating any dynamical system.

In section II we will shortly refer to the NEFCON architecture, and explicitely describe the learning algorithm. The rest of the paper considers the implementation and presents some simulation results.

II. The NEFCON model

The NEFCON model is derived from a generic model of a 3-layer fuzzy perceptron [7] which consists of an input layer, a (hidden) "rule" layer and an output layer. The feedforward connections between the layers are weighted with fuzzy sets. Each of the layers contains a number of units, where the hidden "rule units" use a t-norm as activation function, and the output unit combines fuzzy sets and applies a defuzzification procedure. The input units just contain the input values and are doing no further computation.

The propagation process of input values is equivalent to the evaluation of a set of fuzzy if-then rules as it is used in usual fuzzy controllers. The learning process in a fuzzy perceptron is based on a fuzzy error that can be derived directly by comparing actual and desired output values, or indirectly by a set of fuzzy rules describing the error in dependence of the performance of the network. This error is propagated back through the architecture to adapt the membership functions.

A NEFCON system (see fig. 1) is a special 3-layer fuzzy perceptron with the following specifications:

(i) The input units are denoted as $\xi_1, \ldots, \xi_n$, the hidden rule units are denoted as $R_1, \ldots, R_k$, and the single output unit is denoted as η.

(ii) Each connection between units ξ_i and R_r is labelled with a linguistic term $A_{j_r}^{(i)}$ $(j_r \in \{1, \ldots, p_i\})$.

(iii) Each connection between units R_r and the output unit η is labelled with a linguistic term B_{j_r} $(j_r \in \{1, \ldots, q\})$.

(iv) Connections coming from the same input unit ξ_i and having identical labels, bear the same fuzzy weight at all times. These connections are called *linked conncetions.* An analogous

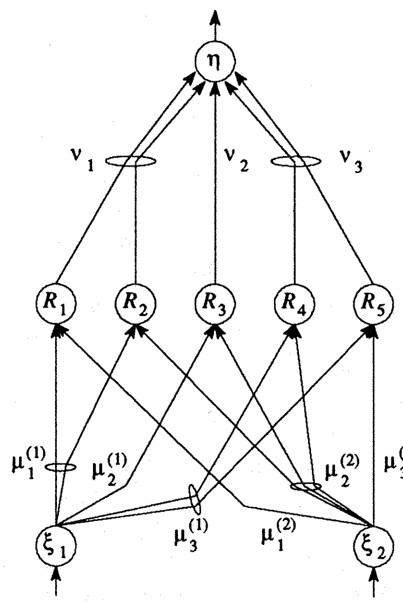


Fig. 1. A NEFCON system with two input variables

condition holds for the connections leading to the output unit η.

(v) Let $L_{\xi,R}$ denote the label of the connection between the input unit ξ and the rule unit R. For all rule units R, R'

$$(\forall \xi \; L_{\xi,R} = L_{\xi,R'}) \Longrightarrow R = R'$$

holds.

This definition makes it possible to interpret a NEFCON system in terms of a fuzzy controller; each hidden unit represents a fuzzy if-then rule. Condition (iv) specifies that there have to be *shared weights.* If this feature is missing, it would be possible for fuzzy weights representing identical linguistic terms to evolve differently during the learning process. If this is allowed to happen, the architecture of the NEFCON system can not be understood as a fuzzy rule base. Condition (v) determines that there are no rules with identical antecedents. A network that does not adhere to this condition is called *overcommitted NEFCON system,* and it is used for learning a linguistic rule base by deleting rule units [6].

In the following we will denote the membership functions μ_j^i ($i \in \{1,\ldots,n\}$, $j \in \{1,\ldots,p_i\}$) of the connections between the input and the hidden layer as *antecedents* and the membership functions ν_j ($j \in \{1,\ldots,q\}$) between the hidden layer and the output unit as *conclusions.* The antecedents are triangular fuzzy sets with three parameters a, b, c, and the conclusion are using Tsukamoto's monotonic membership functions with parameters d, e [4]. They are defined as follows:

$$\mu_j^{(i)}(x) \overset{def}{=} \begin{cases} \dfrac{x - a_j^{(i)}}{b_j^{(i)} - a_j^{(i)}} & \text{if } x \in [a_j^{(i)}, b_j^{(i)}], \\ \dfrac{c_j^{(i)} - x}{c_j^{(i)} - b_j^{(i)}} & \text{if } x \in [a_j^{(i)}, b_j^{(i)}], \\ 0 & \text{otherwise,} \end{cases}$$

with $a_j^{(i)}, b_j^{(i)}, c_j^{(i)} \in \mathbb{R}, a_j^{(i)} \leq b_j^{(i)} \leq c_j^{(i)}$,

$$\nu_j(y) \overset{def}{=} \begin{cases} \dfrac{d_j - y}{d_j - e_j} & \text{if } \begin{array}{r}(y \in [d_j, e_j] \land d_j \leq e_j) \\ \lor\, (y \in [e_j, d_j] \land d_j > e_j)\end{array}, \\ 0 & \text{otherwise,} \end{cases}$$

with $d_j, e_j \in \mathbb{R}$.

A NEFCON system is used to control a dynamical System S with one control variable η and n variables $\xi_1, \ldots, \xi_n$ describing its state. The performance of NEFCON is measured by a fuzzy error, that is defined by a number of fuzzy rules like
if ξ_1 is *approx. zero* and ξ_2 is *approx. zero*
then the error is *small,*
where ξ_1 and ξ_2 are two state variables of the dynamical system, and input variables of the NEFCON system, respectively. Because the error is defined by fuzzy rules, its value can be determined in the same way as the control output η, i.e. it is possible to use a second NEFCON system for this task. The defuzzified error value obtained from the fuzzy rules is used for the learning algorithm.

Definition 1 *For a NEFCON system with n input units $\xi_1, \ldots, \xi_n$ and k rule units $R_1, \ldots, R_k$, and the output unit η the fuzzy error backpropagation learning algorithm for adapting the membership functions is defined by the following steps that have to be repeated until a certain end criterion is met (see below).*

(i) Calculate the NEFCON output o_η:

$$o_\eta = \frac{\sum\limits_R o_R \cdot t_R}{\sum\limits_R t_R}$$

with $o_R = min\{\mu_R^{(1)}(\xi_1), \ldots, \mu_R^{(n)}(\xi_n)\}$ and $t_R = \nu_R^{-1}(o_R)$ which defines the crisp output value of rule unit R that can be calculated directly by the inverse of the monotonic function ν_R. $\mu_R^{(i)}$ connects the rule unit R with input unit ξ_i and ν_R connects the rule unit R with the output unit η.

(ii) Apply the output value to S and determine the new state of S.

(iii) Determine the fuzzy error E from the state of S. This is done by evaluating a set of fuzzy rules describing E.

(iv) Determine the sign of the optimal output value η_{opt} for the new state of S (the actual absolute value of the optimal output is unknown, of course, but its sign has to be known).

(v) Determine for each rule unit R_r the part t_r that it has in the output value η, and calculate the fuzzy rule error E_{R_r} for each R_r ($r \in \{1, \ldots, k\}$):

$$E_{R_r} \stackrel{def}{=} \begin{cases} -o_{R_r} \cdot E & \text{if } \operatorname{sgn}(t_r) = \operatorname{sgn}(\eta_{\text{opt}}) \\ o_{R_r} \cdot E & \text{otherwise.} \end{cases}$$

(vi) Determine the changes in the parameters of the membership functions ν_{j_r} ($j_r \in \{1, \ldots, q\}$, $r \in \{1, \ldots, k\}$):

$$\Delta\, d_{j_r} \stackrel{def}{=} \sigma \cdot E_{R_r} \cdot |d_{j_r} - e_{j_r}|,$$

with a learning rate $\sigma > 0$.
Apply these changes to the fuzzy sets ν_{j_r}, such that certain constraints $\Psi(\nu_{j_r})$ are met (see below).

(vii) Determine the changes in the parameters of the membership functions $\mu_{j_r}^{(i)}$ ($i \in \{1, \ldots, n\}$, $j_r \in \{1, \ldots, p_i\}$, $r \in \{1, \ldots, k\}$):

$$\Delta\, a_{j_r}^{(i)} \stackrel{def}{=} -\sigma \cdot E_{R_r} \cdot (b_{j_r}^{(i)} - a_{j_r}^{(i)}),$$

$$\Delta\, c_{j_r}^{(i)} \stackrel{def}{=} \sigma \cdot E_{R_r} \cdot (c_{j_r}^{(i)} - b_{j_r}^{(i)}).$$

Apply these changes to the fuzzy sets ν_{j_r}, such that certain constraints $\Psi(\mu_{j_r}^{(i)})$ are met (see below).

If the fuzzy error E is smaller than a certain value for a certain number of cycles, this may be used as a criterion to end the learning process. But it is also possible to never stop the learning so that the controller is able to adapt to any changes in the dynamical system. If the controlled system has reached a good state, the error value will be around zero, and the changes in the fuzzy sets will be also near zero, or compensate each other in the long run, respectively.

The constraints Ψ which can be defined on the membership functions can be used to assure that e.g. $a \leq b \leq c$ always holds, or that there is a certain amount of overlapping in the fuzzy sets, etc.

This learning algorithm is used to adapt the membership functions only, and can be applied if a rule base of fuzzy rules is known. These rules can be directly transformed into a NEFCON system. The membership functions have to be initialized. If there is absolutely no knowledge about them, a uniform fuzzy partitioning of the variables can be used. The learning algorithm only changes the width of the fuzzy sets, not their position. This is done to reduce the number of changes caused by the learning procedure, and to keep the algorithm under control. But of course the learning algorithm can be easily generallized, and the instructions for changing the parameters can be analogously defined for the parameters b and e.

If only some or no rules at all are known, the learning algorithm can be extended to learn the fuzzy rules, too. This is achieved by starting with an overcommitted NEFCON system and deleting those rule units that accumulate the highest error values during a training phase. If there are no known fuzzy rules, the system starts with $N = q \cdot \prod_{i=1}^{n} p_i$ rule nodes, which represent all possible fuzzy rules that can be defined due to the partitioning of the variables (i.e. the number of fuzzy sets for each variable has to be specified before). In the following definition $\mathcal{R}$ denotes the set of all rule units, $Ant(R)$ denotes the antecedents of a rule unit, and $Con(R)$ denotes the conclusion of a rule unit.

Definition 2 *For an overcommitted NEFCON system with $N = q \cdot \prod_{i=1}^{n} p_i$ initial rule units with*

$$\begin{aligned} &(\forall R, R' \in \mathcal{R}) \\ &(Ant(R) = Ant(R') \wedge Con(R) = Con(R')) \\ &\Longrightarrow R = R', \end{aligned}$$

and n input units $\xi_1, \ldots, \xi_n$ representing variables partitioned by p_i fuzzy sets and an output unit η representing a variable partitioned by q fuzzy sets, the extended fuzzy error backpropagation learning algorithm for deleting unnecessary rule units uses the following steps.

(i) For each rule unit R_r a counter C_r initialized to 0 is defined ($r \in \{1, \ldots, N\}$).
For a fixed number m_1 of iterations the following steps are carried out:

(a) Determine the current NEFCON output o_η using the current state of S.

(b) For each rule R_r determine its part t_r in the overall output o_η ($r \in \{1, \ldots, N\}$).

(c) Determine $\operatorname{sgn}(\eta_{\text{opt}})$ for the current input values.

(d) *Delete each rule unit R_r with* $\mathrm{sgn}(t_r) \neq \mathrm{sgn}(\eta_{\mathrm{opt}})$ *and update the value of* N.

(e) *Increment the counters* C_r *for all* R_r *with* $o_{R_r} > 0$.

(f) *Apply* o_η *to* S *and determine the new input values.*

(ii) *For each* R_r *a counter* Z_r *initialized to 0 is defined.*
For a fixed number m_2 *of iterations the following steps are carried out:*

(a) *From all subsets*

$$\mathcal{R}_j = \{R_r | Ant(R_r) = Ant(R_s), (r \neq s) \wedge (r, s \in \{1, \ldots, N\})\} \subseteq \mathcal{R}$$

one rule unit R_{r_j} *is selected arbitrarily.*

(b) *Determine the NEFCON output* o_η *using only the selected rule units and the current state of* S.

(c) *Apply* o_η *to* S *and determine the new input values.*

(d) *Determine the part* t_{r_j} *of each selected rule unit* R_{r_j} *in the overall output value* $(r_j \in \{1, \ldots, N\})$.

(e) *Determine* $\mathrm{sgn}(\eta_{\mathrm{opt}})$ *using the new input values.*

(f) *Add the error value* $E_{R_{r_j}}$ *of each selected rule unit* R_{r_j} *to its counter* Z_{r_j}.

(g) *For all seleceted rule units* R_{r_j} *with* $o_{R_{r_j}} > 0$ C_{r_j} *is incremented.*

Delete all rule units R_{s_j} *for all subsets* $\mathcal{R}_j$ *from the network for which there is a rule unit* $R_{r_j} \in \mathcal{R}_j$ *with* $Z_{r_j} < Z_{s_j}$, *and delete all rule units* R_r *with* $C_r < \frac{m_1 + m_2}{\beta}$, $\beta \geq 1$ *from the network, and update the value of* N.

(iii) *Apply the fuzzy error backpropagation algorithm to the NEFCON system with* $k = N$ *remaining rule units (see Def. 1).*

The idea of the rule learning algorithm is to try out the existing rules, and to valuate them. Rule units that do not pass this test are eliminated from the network. In the first phase all rule units producing an output with a sign different from the optimal ouput value are deleted. In the second phase the algorithm has to choose from subsets that consist of rules with identical antecedents one rule and delete all other rules of each subset. Doing this we come from an overcommitted NEFCON system to a regular NEFCON system. In the third phase finally the fuzzy sets are adapted.

The rule learning algorithm becomes very expensive, when there are a lot of fuzzy sets defined for a lot of variables. For this reason one should always try to use partial knowledge to avoid that all possible rules have to be created. If there are no known rules for certain input states, then for these particualar states only all possible rules have to be created. This way the number of initial rule units can be reduced.

III. NEFCON-I – The Implementation

The interactive graphical simulation environment NEFCON-I (I means InterViews) is based on the X-Window system and uses the graphical interface builder InterViews. NEFCON-I has been implemented on SUN workstations, and is both freely available in binary and in source code versions on the internet[1].

After starting the program the user is able to load the variables of a dynamical system with their fuzzy sets, and a fuzzy rule base. If there are no fuzzy sets or rules, they can be defined by graphical editors. The fuzzy error is specified in the same way.

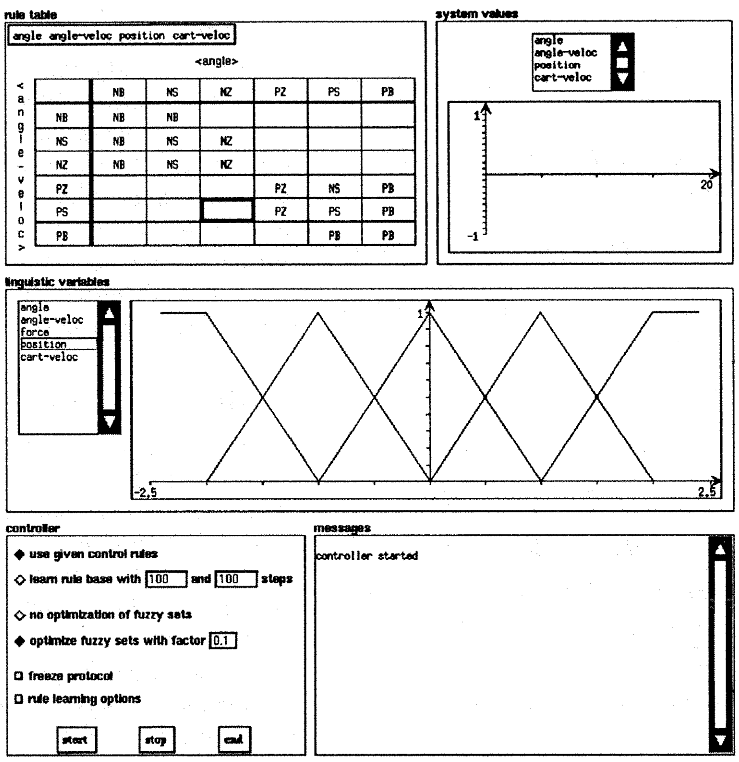
After the necessary data has been entered, a NEFCON system is created and a graphical interface for supervising the learning procedure is shown (see fig. 2). The user can specify parameters for the learning algorithm and view the changes in the fuzzy sets of the variables. If the fuzzy rules have to be learned, the rule table is empty at first. After the rule base has been created by the learning algorithm the fuzzy rules are shown.

To start a learning process and to view the performance of the controller the user has to specify the name of a program containing a simulation of a dynamical system. For an example we use an implementation of the inverted pendulum using a Runge-Kutta procedure and the equations and constants as they are presented in [1].

IV. Simulation Results

In this section we present some results of the learning capabilites of NEFCON applied to the well known inverted pendulum. To be able to show the changes in the control surface we only consider the angle $\theta \in [-90, 90]$ and the angle velocity $\dot{\theta} \in [-200, 200]$. The fuzzy error for all experiments shown below has been

[1] To obtain NEFCON-I (available in Jan. '94) establish an anonymous ftp connection to ftp.tu-bs.de (134.169.34.15), change to the directory local/nefcon, and retrieve and read the readme file first.


rule table

angle angle-veloc position cart-veloc

<angle>

<angle-veloc>

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

controller

◆ use given control rules

◇ learn rule base with 100 and 100 steps

◇ no optimization of fuzzy sets

◆ optimize fuzzy sets with factor 0.1

☐ freeze protocol

☐ rule learning options

messages

controller started

Fig. 2. Learning and control mode of NEFCON-I

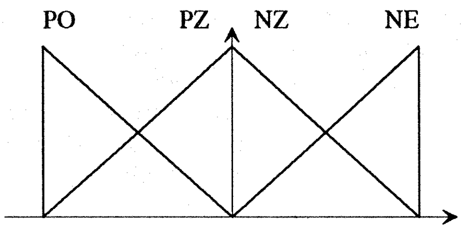
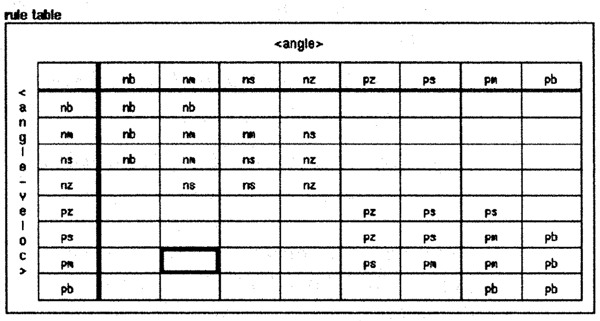


Fig. 3. Fuzzy sets of θ and $\dot{\theta}$ for the definition of the fuzzy error


rule table

<angle>

<angle-veloc>

	nb	nm	ns	nz	pz	ps	pm	pb
nb	nb	nb						
nm	nb	nm	nm	ns				
ns	nb	nm	ns	nz				
nz		ns	ns	nz				
pz					pz	ps	ps	
ps					pz	ps	pm	pb
pm					ps	pm	pm	pb
pb							pb	pb

Fig. 5. The rule base of NEFCON

		θ NE	NZ	PZ	PO
$\dot{\theta}$	NE	NE	NE	NE	PZ
	NZ	NE	NZ	PZ	PO
	PZ	NE	NZ	PZ	PO
	PO	NZ	PO	PO	PO

Fig. 4. The fuzzy error rules

defined by the fuzzy sets and rules shown in fig. 3 and 4.

For the first experiment we used the rule base shown in fig. 5. In fig. 6 the fuzzy sets of θ are shown before and after learning. With the initial partitioning the controller was not able to balance the pendulum. After a few failures and after restarting the pendulum at random angles, the final fuzzy sets developed and the pendulum was stable. This solution took about 2000 cycles which is about 2 min. computing on a SUN 2 Sparcstation.

The figures 7 and 8 show the control surface before and after the learning process. The final control surface is much smoother than the initial one, resulting in a better control behavior.

The second experiment considers the learning of a rule base. For this the fuzzy sets learned during the first experiment are used. After the learning algorihtm ran for 6000 cycles, it came up with the rule-base shown in fig. 9. The controller can balance the pendulum with this rule base, and after the fuzzy sets have been adapted for a few hundred additional cycles the controller can even cope with disturbances of ±20 degrees.

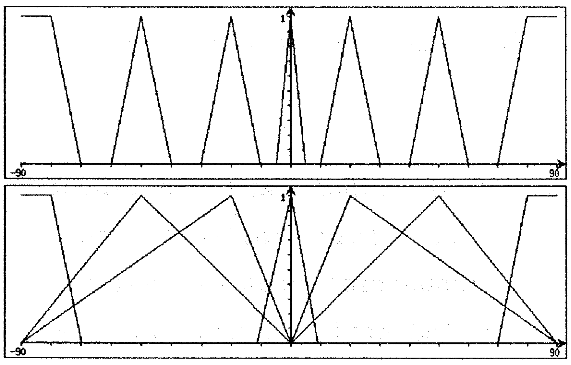


Fig. 6. The fuzzy sets of θ before and after learning

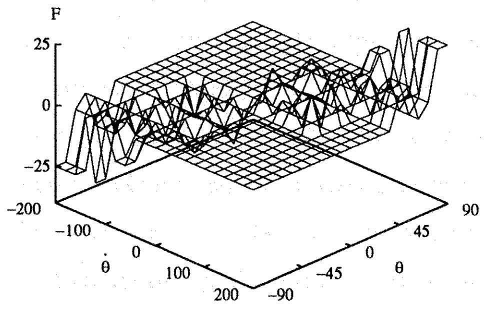


Fig. 7. Control surface before learning

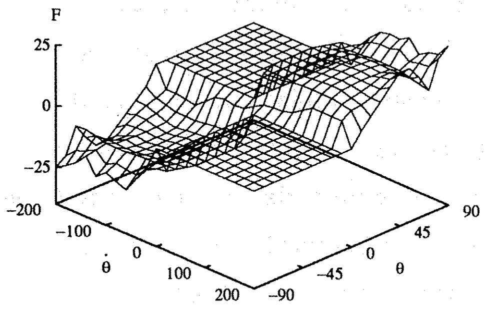
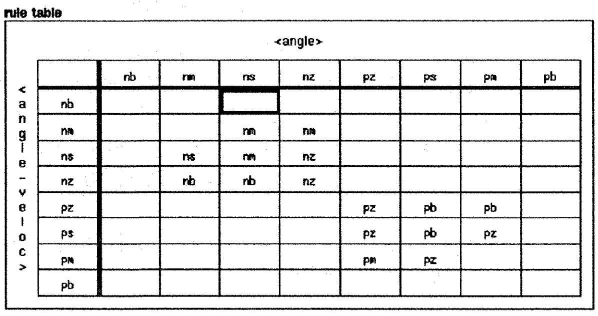


Fig. 8. Control surface after learning


rule table

<angle>

<angle-veloc>

	nb	nm	ns	nz	pz	ps	pm	pb
nb								
nm			nm	nm				
ns		ns	nm	nz				
nz		nb	nb	nz				
pz					pz	pb	pb	
ps					pz	pb	pz	
pm					pm	pz		
pb								

Fig. 9. Rule base learned by NEFCON-I

V. Conclusions

We have presented the learning algorithm of the NEFCON model which is able to learn the fuzzy sets and the fuzzy rules of a neural fuzzy controller by backpropagating a fuzzy error measure through the NECON architecture. Using a knowledge based error enables the reinforcement type learning procedure to work without an adaptive critic element.

The model has been implemented in a graphical simulation environment showing that the algorithms can successfully learn the necessary parameters while controlling the simulation of a dynamic system. Further research will be concentrated on generalizing the learning algorithm, e.g. to be able to use center of gravity defuzzification.

References

[1] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Trans. Systems, Man & Cybernetics*, 13:834–846, 1983.

[2] Hamid R. Berenji and Pratap Khedkar. Learning and tuning fuzzy logic controllers through reinforcements. *IEEE Trans. Neural Networks*, 3:724–740, September 1992.

[3] Bart Kosko. *Neural Networks and Fuzzy Systems. A Dynamical Systems Approach to Machine Intelligence.* Prentice-Hall, Englewood Cliffs, 1992.

[4] Chuen Chien Lee. Fuzzy logic in control systems: Fuzzy logic controller, part i. *IEEE Trans. Systems, Man & Cybernetics*, 20:404–418, 1990.

[5] Detlef Nauck and Rudolf Kruse. A neural fuzzy controller learning by fuzzy error propagation. In *Proc. Workshop of North American Fuzzy Information Processing Society (NAFIPS92)*, pages 388–397, Puerto Vallarta, December 1992.

[6] Detlef Nauck and Rudolf Kruse. A fuzzy neural network learning fuzzy control rules and membership functions by fuzzy error backpropagation. In *Proc. IEEE Int. Conf. on Neural Networks 1993*, pages 1022–1027, San Francisco, March 1993.

[7] Detlef Nauck and Rudolf Kruse. A fuzzy perceptron as a generic model of multilayer fuzzy neural networks. December 1993. submitted.