Neuro-Fuzzy Techniques under MATLAB/SIMULINK Applied to a Real Plant

Andreas Nürnberger
University of Magdeburg
Faculty of Computer Science
39106 Magdeburg, Germany
E-mail: andreas.nuernberger@cs.uni-magdeburg.de

Rudolf Kruse
University of Magdeburg
Faculty of Computer Science
39106 Magdeburg, Germany
E-mail: rudolf.kruse@cs.uni-magdeburg.de

Abstract

The design and optimization process of fuzzy controllers can be supported by learning techniques derived from neural networks. Such approaches are usually called neuro-fuzzy systems. In this paper, we describe the application of an updated version of the neuro-fuzzy model NEFCON to a real plant. The NEFCON model is able to learn and optimize the rulebase of a Mamdani-type fuzzy controller online by a reinforcement learning algorithm that uses a fuzzy error measure. We used an implementation of this model under MATLAB/SIMULINK. This simulation environment supports the development of real time applications in an easy way.

1. Introduction

The main problems in fuzzy controller design are the construction of an initial rulebase and in particular the optimization of an existing rulebase. The optimization process is usually very time consuming, especially if real plants must be used during optimization. The methods used for the application presented in this paper have been developed to support the user in these cases.

One of the main objectives of our project is to develop algorithms that are able to determine online an appropriate and interpretable rulebase within a small number of simulation runs. Besides, it must be possible to use prior knowledge to initialize the learning process.

This is a contrast to 'pure' reinforcement strategies [2] or methods based on dynamic programming [1, 10] which try to find an optimal solution using neural network structures. These methods need many runs to find even an approximate solution for a given control problem. On the other hand, they have the advantage of using less information about the error of the current system state. However, in many cases a simple error description can be achieved with little effort [7, 9].

In this paper, we describe the application of neuro-fuzzy learning methods to a real plant. We chose MATLAB/SIMULINK [9] as environment for the neuro-fuzzy model, in order to use a standard software tool, that is well suited for the design of industrial applications.

The model was implemented as a toolbox for MATLAB/SIMULINK [9]. Thus, it is available for the interactive design of fuzzy controllers and supports it by learning methods.

The used learning techniques are based on the neuro-fuzzy model NEFCON [7].

1.1. The NEFCON-Model

The NEFCON-Model is based on a generic fuzzy perceptron [6, 7]. An example, which describes the structure of a fuzzy controller with five rules, two inputs, and one output, is shown in Figure 1. The inner nodes R_1, …, R_5 represent the rules, the nodes ξ_1, ξ_1, and η the input and output values, and $\mu_r^{(i)}$, ν_r the fuzzy sets describing the antecedents $A_r^{(i)}$ and consequents B_r.

Rules with the same antecedents use so-called shared weights, which are represented by ellipses in Figure 1. They ensure the integrity of the rulebase. The node R_1 for example represents the rule:

$$R_1\text{: if } \xi_1 \text{ is } \mathrm{A}_1^{(1)} \text{ and } \xi_2 \text{ is } \mathrm{A}_1^{(2)} \text{ then } \eta \text{ is } \mathrm{B}_1 .$$

The model structure allows to learn and optimize the rulebase of a Mamdani-type fuzzy controller [4] online by a reinforcement learning algorithm.

2. The Learning Algorithms

The learning process of the NEFCON model can be divided into two main phases. The first phase is designed

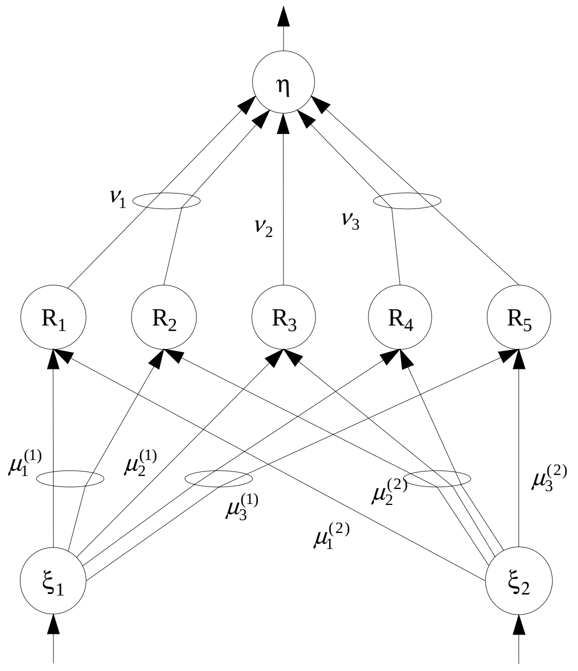

Figure 1: A NEFCON System with two inputs, five rules and one output

to learn an initial rulebase, if no prior knowledge of the system is available. Furthermore, it can be used to complete a manually defined rulebase.

The second phase optimizes the rules by shifting or modifying the fuzzy sets of the rules.

Both phases use a fuzzy error to learn or to optimize the rulebase [7, 9]. The fuzzy error describes the quality of the current system state

2.1. Rulebase Learning

For the presented application, we used the ‘Bottom-Up`-Algorithm [8, 9].

This algorithm starts with an empty rulebase. An initial fuzzy partitioning of the input and output intervals must be given. The algorithm can be divided into two parts.

During the first part, the rules' antecedents are determined by classifying the input values, i.e. finding that membership function for each variable that yields the highest membership value for the respective input value. Then the algorithm tries to ‘guess’ the output value by deriving it from the current fuzzy error.

During the second part, the rulebase is optimized by changing the consequent to an adjacent membership function, if this is necessary.

2.2. Optimization of the Rulebase

To optimize the rulebase we choose the optimization algorithm NEFCON-I [9].

This algorithm is motivated by the backpropagation algorithm for the multilayer perceptron. It optimizes the rulebase by ‘reward and punishment’. A rule is ‘rewarded’ by shifting its consequent to a higher value and by widening the support of the antecedents, if its current output has the same sign as the optimal output. Otherwise, the rule is ‘punished’ by shifting its consequent to a lower value and by reducing the support of the antecedents.

2.3. Description of the System Error

For the description of the system error, we use a linguistic error description [7].

This method is based on the fact that the optimal state of a dynamic system can be described by a vector of system state variable values. Usually the state can not be described exactly, or we are content, if the system variables have roughly taken these values. Thus, the quality of a current state can be described by fuzzy rules.

By use of an error definition that is based on a linguistic error description with fuzzy rules, it is also easily possible to describe compensatory situations. These are situations in which the dynamic system is driven towards its optimal state.

3. Learning applied to a real plant

Since we want to be able to determine an appropriate and interpretable rulebase within a small number of simulation runs, we decided to split the learning process in two main steps.

During the first step, the rulebase will be learned and optimized by use of a simple (linear) model of the real plant. Because of the differences between the real plant and the linear model used for learning, the fuzzy controller will not be able to control the real plant appropriately in most cases. Therefore, a second learning step will be necessary. During the second step, the derived rulebase will be optimized by use of the real plant.

Thus, an appropriate working rulebase can be found with less experimental (and time) effort on the real plant. Besides, the risk of a damage of the real plant is reduced.

As an example for a real plant, we used the well-known inverted pendulum.

3.1. The learning Environment

The used inverted pendulum model is shown in Figure 2. The revolving pendulum is mounted on top of a moving

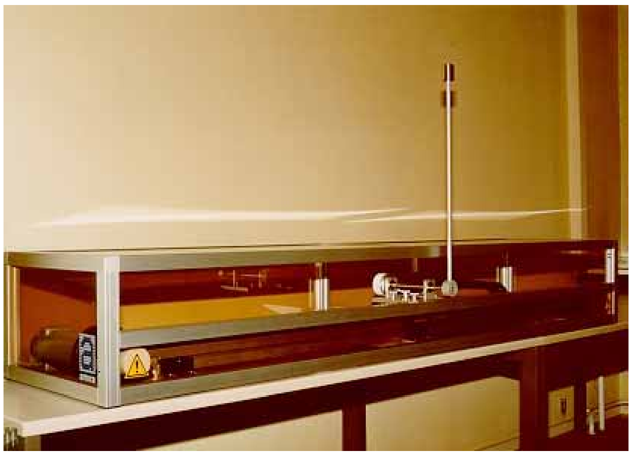
Figure 2: The used inverted pendulum model

base. The moving base can be driven along a track over a length of approximately 1.5m. The moving base is driven by a DC-motor, a toothwheel, a toothbelt and a clutch.

The measured values of the pendulum are the pendulum angle a (|a| < 10°), the cart position x obtained by incremental encoders, and the cart velocity x'.

The plant was connected to a standard Pentium 166 PC running Microsoft Windows NT 4.0. For the data transfer we use of a standard I/O interface card with a scanning rate of at least 10 ms (during simulation we used a scanning rate of 30 ms).

For the first learning step, we used a simple linear model of the real pendulum. This model was implemented in the SIMULINK learning environment, which was constructed for this learning approach (see Figure 3).

The derivation of a linear model for an inverted pendulum is described in detail in [5].

As mentioned above, the angle velocity could not be obtained directly from the plant. The use of two measured angles to calculate a local derivative was not possible due to the measurement errors. These errors were caused by the simple mechanical construction of the pendulum and its control environment. Therefore, we used an observer to derive the missing value (see, for example, [11, 12]). The observer calculates the missing value, in this case the angle velocity a', by use of the measured values and the control values applied to the plant (see the block *observer* in Figure 3).

The derivation of the observer for the pendulum is described in [5], too.

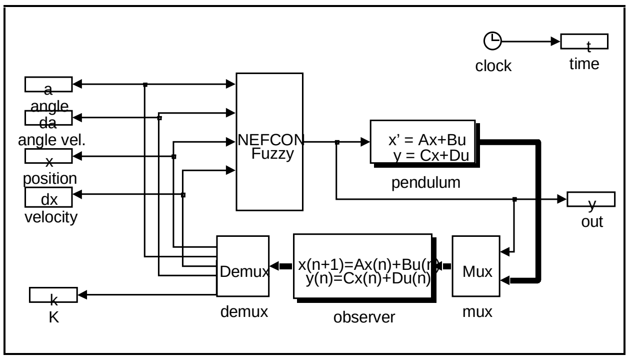

Figure 3: SIMULINK simulation environment for the pendulum

3.2. The Used Error Description

The required system error for the learning algorithms was defined by the rulebase depicted in Figure 4. The input domains of the fuzzy system, which defines the error description, are partitioned by three triangular membership functions and the output domain by five triangular membership functions.

1	If (a is n) and (da is p)	then (err is z)
2	If (a is p) and (da is n)	then (err is z)
3	If (a is p) and (da is p)	then (err is n)
4	If (a is n) and (da is n)	then (err is p)
5	If (a is p) and (da is z)	then (err is n)
6	If (a is n) and (da is z)	then (err is p)
7	If (a is z) and (da is n)	then (err is p)
8	If (a is z) and (da is p)	then (err is n)
9	If (a is z) and (da is z) and (x is p) and (dx is z)	then (err is pz)
10	If (a is z) and (da is z) and (x is n) and (dx is z)	then (err is nz)
11	If (a is z) and (da is z) and (x is z) and (dx is p)	then (err is p)
12	If (a is z) and (da is z) and (x is z) and (dx is n)	then (err is n)
13	If (a is z) and (da is z) and (x is n) and (dx is n)	then (err is n)
14	If (a is z) and (da is z) and (x is p) and (dx is p)	then (err is p)
15	If (a is z) and (da is z) and (x is n) and (dx is p)	then (err is z)
16	If (a is z) and (da is z) and (x is p) and (dx is n)	then (err is z)

(a - angle; da - angle velocity; x - position; dx - velocity; err - error)

Figure 4: Rulebase of the linguistic error description

The rules 1 to 8 (Figure 4) are used to define the error of the pendulum.

The rules 9 to 16 define the position error of the cart. The error signal derived by these rules is only used (unequal to zero), if the pendulum is well balanced (*a is zero* and *da is zero*). Thus, a straightforward and non-optimal control strategy is implicitly defined by this error description.

3.3. Learning and Optimization

The learning algorithm was initialized with an empty rulebase. The input and output intervals have been partitioned by five triangular membership functions. Each simulation cycle was started with random initial conditions for the angle and angle velocity of the pendulum.

For the first learning approach we used only the angle and angle velocity to learn a rulebase. Therefore, the error description was restricted to the rules 1 to 8 (Figure 4) and

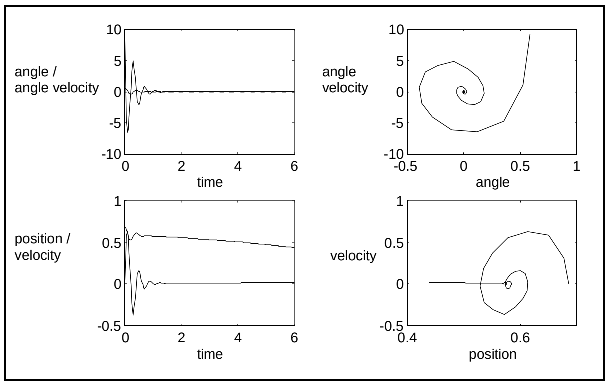

Figure 5: Model based learned rulebase (restricted to a and a’) applied to the simulated model

a NEFCON system with two input and one output values was used.

During the rulebase learning phase, noise was added to the reference signal to improve the coverage of the system state space [2]. Each cycle took 10 seconds of simulation time. If the pendulum fell down, the current cycle was terminated immediately.

The learning algorithm was able to generate an appropriate working rulebase in only three cycles of rulebase learning and three cycles for optimization. The control behavior is depicted in Figure 5.

Afterwards, the learned rulebase was applied to the real pendulum model. This was done by simply replacing the linear pendulum model block (*pendulum*) in the SIMULINK simulation environment (see Figure 3). The block was replaced by a control block, which communicates directly with the plant, by use of the I/O interface card of the PC.

The controller was able to balance the pendulum without further optimization quite well. However, since we used no restrictions for the cart movement during learning, the cart moved slightly against the system boundaries. The simulation results are depicted in Figure 6.

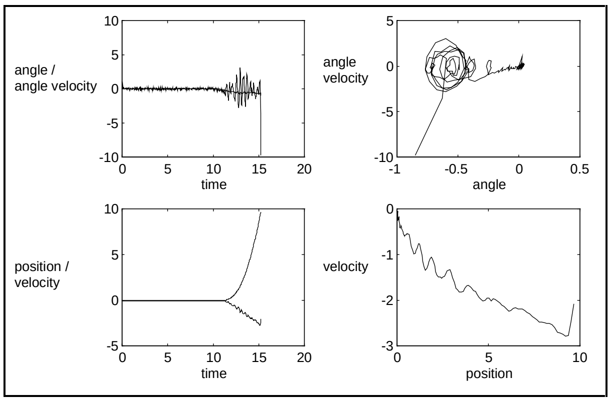

Figure 6: Model based learned rulebase (restricted to a and a’) applied to the real plant

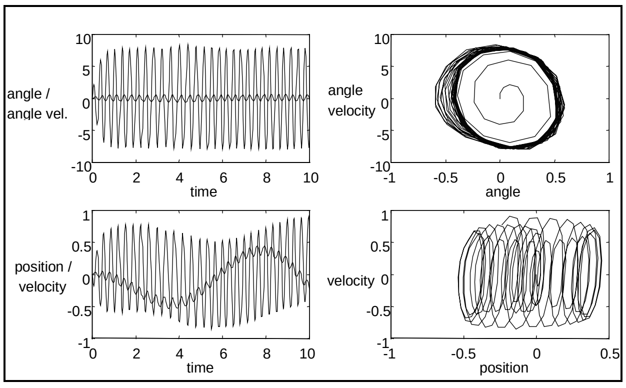

Figure 7: Model based learned rulebase applied to the simulated model

For the next learning approach, we used the simulation environment as presented in (Figure 3) and the complete error description (Figure 4).

The learning algorithm was able to generate an appropriate working rulebase within only five cycles for rulebase learning and three cycles for the optimization of the fuzzy sets. The control behavior is shown in Figure 7.

Nevertheless, the control behavior is not optimal. The presented control behavior is a typical sample of system control with an automatically learned rulebase. The system ’swings’ slightly around its optimal state, but remains stable. Better results could be obtained, for example, by starting the learning procedure with different initial conditions or by refining the used error description. Furthermore, the intervals for the input and output values could be changed or prior knowledge could be used to initialize the rulebase of the fuzzy controller.

Finally, we applied the rulebase, which was learned by use of the simple linear model, to the real pendulum. The control results are depicted in Figure 8.

The controller was able to balance the pendulum, as

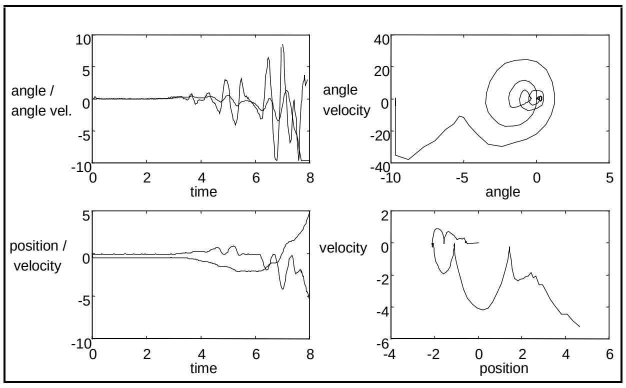

Figure 8: Model based learned rulebase applied to the real plant

expected, only for a short period (about seven seconds). This is caused by the model-based differences between the real pendulum and the linear model used for learning. Further tests have shown, that even rulebases obtained by more exact models work only slightly better, when they were applied to the real model.

4. Conclusion and future work

By the implementation of the updated NEFCON model under MATLAB/SIMULINK, it is possible to use the model conveniently for the design of fuzzy controllers for different dynamic systems.

As presented, the rulebase obtained by use of a simple model of a plant can be applied to a real plant in an easy way. Nevertheless, the derived rulebase has to be optimized by using the real plant in most cases, to be able to control the plant appropriately.

Currently, this optimization cannot be done in real time, due to some performance problems with the current release of the MATLAB/SIMULINK environment and the announced, but still missing, MATLAB compiler. The next release of our NEFCON tool will likely support online learning in real time.

Nevertheless, in case of more complex dynamic systems, the quality of the results greatly depends on the definition of the fuzzy error measure. This is caused by the fact that the NEFCON algorithms use only a simple approach to include the dynamics of the controlled system in the optimization process (see, for example, the credit assignment problem [2]).

Some variations of reinforcement strategies [1, 3] have to be analyzed in order to determine, if it will be possible to integrate them into the optimization phase of the presented algorithms. It has to be studied whether they improve the quality of the controller without increasing the number of runs for learning significantly.

5. Remarks

The used development tool for fuzzy controllers under MATLAB/SIMULINK can be obtained free of charge for non-commercial purposes via the Internet from http://fuzzy.cs.uni-magdeburg.de/nefcon.

MATLAB/SIMULINK is a simulation tool developed by ‘The Mathworks’ Inc., 24 Prime Park Way, Natick, Mass. 01760; (WWW: http://www.mathworks.com).

The used inverted pendulum was provided by the Institute of Automation (IFAT), University of Magdeburg, Germany (WWW: http://infaut.et.uni-magdeburg.de).

The used plant (PS600 Position Control and Inverted Pendulum) was constructed by amira GmbH, Düsseldorf, Germany (WWW: http://www.amira.de).

6. References

[1] Barto, A. G., Bradtke, S. J., and Singh, S. P., Learning to act using real-time dynamic programming, *Artificial Intelligence, Special Volume: Computational Research on Interaction and Agency*, **72**(1), 81-138, 1995

[2] Barto, A.G., Sutton R. S., and Anderson, C. W., Neuronlike adaptive elements that can solve difficult learning control problems, *IEEE Transactions on Systems, Man and Cybernetics*, **13**, 834-846, 1983

[3] Lin, C.T., *Neural Fuzzy Control Systems with structure and Parameter Learning*, World Scientific Publishing, Singapore, 1994

[4] Mamdani, E. H., and Assilian S., An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller, *International Journal of Man-Machine Studies*, **7**, pp. 1-13, 1973

[5] Mori, Shozo, Nishihara, Hiroyoshi, and Furuta, Kattsuhisa, Control of an unstable mechanical system, Control of pendulum, Int. J. Control, Vol. 23, No. 5, pp. 673-692, 1976

[6] Nauck, D., 1994, A Fuzzy Perceptron as a Generic Model for Neuro-Fuzzy Approaches, *Proceedings of the 2nd German GI-Workshop Fuzzy-Systeme ’94*, München, Germany, October.

[7] Nauck, D., Klawonn, F., and Kruse, R., *Foundations of Neuro-Fuzzy Systems*, John Wiley & Sons, Inc., New York, Chichester, 1997

[8] Nauck, D., Kruse, R., and Stellmach, R., New Learning Algorithms for the Neuro-Fuzzy Environment NEFCON-I, *Proceedings of the 3rd German GI-Workshop Fuzzy-Neuro-Systeme ’95*, Darmstadt, Germany, November, pp. 357-364, 1995

[9] Nürnberger, A., Nauck, D., Kruse, R., Merz, L., A Neuro-Fuzzy Development Tool for Fuzzy Controllers under MATLAB/SIMULINK, *In Proc. of the 5th European Congress on Intelligent Techniques & Soft Computing (EUFIT ’97)*, Aachen, Germany, 1997

[10] Riedmiller, M., and Janusz, B., Using Neural Reinforcement Controllers in Robotics, *Proceedings of the 8th Australian Conference on Artificial Intelligence*, Canberra, Australia., 1995

[11] Tou, J. T., *Modern Control Theory*, McGraw Hill, New York., 1964

[12] Unbehauen, H., *Regelungstechnik II*, Vieweg Verlag, Braunschweig, Germany, 1987