Adaptive Critic Based Approximate Dynamic Programming for Tuning Fuzzy Controllers

Thaddeus T. Shannon, George G. Lendaris
Northwest Computational Intelligence Laboratory and Systems Science Program,
Portland State University
tads@sysc.pdx.edu, lendaris@sysc.pdx.edu

Abstract: **In this paper we show the applicability of the Dual Heuristic Programming (DHP) method of Approximate Dynamic Programming to parameter tuning of a fuzzy control system. DHP and related techniques have been developed in the neurocontrol context but can be equally productive when used with fuzzy controllers or neuro-fuzzy hybrids. We demonstrate this technique on a highly nonlinear 2nd order plant proposed by Sanner and Slotine. Throughout our example application, we take advantage of the TS model framework to initialize our tunable parameters with reasonable problem specific values, a practice difficult to perform when applying DHP to neurocontrol.**

I. Introduction

This paper demonstrates the use of adaptive critic based approximate dynamic programming for tuning a fuzzy controller. A variety of Adaptive Critic Design techniques for training neuro-controllers have appeared in the literature recently, falling into model-based methods such as Dual Heuristic Programming (DHP), and non-model-based methods such as Action Dependent Heuristic Dynamic Programming (ADHDP) or Q-learning [1][5][6][10] [11][12][15][16][19][20].

Previous applications of Adaptive Critic based reinforcement learning to the tuning of fuzzy controllers have relied on non-model based temporal differencing schemes [3][4][7][8]. Equivalent neural network based techniques have been shown to be generally less effective than model based techniques such as DHP [10][11][14].

Model based methods utilize the Jacobian of the coupled plant-controller system to train both the controller and critic networks. These derivatives can be found explicitly from an analytic model, or implicitly, for example by backpropagation through a neural network plant model. In [17] we showed that such derivative information could be explicitly estimated in the form of a Takagi-Sugeno (TS) fuzzy model of the plant [18]. Based on our experiences detailed in [17] we now suggest that the overall approximate dynamic programming technique can easily be adapted to the tuning of fuzzy controllers.

As has been observed in previous work with these techniques [5], there are significant advantages to starting the controller training (tuning) process with a set of computational structures (neural networks, fuzzy rule sets, etc.) that are well matched to the task at hand. Structures that are too small will be unable to satisfactorily learn the desired task, while overly large structures tend to require longer training times and tend to exhibit poor generalization. Problem specific knowledge is required to match computational structures to specific tasks. Doing this "prestructuring" with the neural network architectures commonly used in this arena (feedforward or recurrent networks with sigmoidal activation functions) is a rather opaque task. Using these very general reinforcement learning techniques with controllers set in the fuzzy systems context offers a way around this difficulty in many application contexts.

The first section provides a brief overview of adaptive critic based approximate dynamic programming and then delves into the details of the specific technique we demonstrate, Dual Heuristic Programming (DHP). In the second section we introduce a nonlinear plant proposed by Sanner and Slotine [14] to illustrate this method. In the following section we describe controller and critic architectures. We propose using a fuzzy TS model as our adaptive critic function approximator to show that DHP can be implemented without resort to neural networks. It should be obvious to all that in practical applications, it may be desirable to mix neural network critics with fuzzy controllers. The next section introduces a TS model of the plant for use in the tuning process. We describe the specifics of its estimation and its use in DHP. The last section before the conclusion gives the details of the on-line tuning process and our results on the benchmark problem.

II. Approximate Dynamic Programming

Dynamic Programming is a general approach for sequential optimization applicable under very broad conditions. Fundamental to this approach is Bellman's Principle of Optimality [2]: that an optimal trajectory has the property that no matter how an intermediate point is reached, the rest of the trajectory must coincide with an optimal trajectory as calculated with the intermediate point as the starting point. This principle is applied by formulating a "primary" utility function *U(t)* that embodies a control objective for a particular context in one or more measurable variables. A *secondary* utility function is then formed

$$J(t) = \sum_{k=0}^{\infty} g^{k} U(t+k),$$

which embodies the desired control objective through time. This is Bellman's equation, and the point of Dynamic Programming is to select the sequence of actions (controls) that maximize or minimize *J(t)*. Unfortunately, this optimization is not computationally tractable for most real world problems, thus we are forced to consider potentially tractable approximation methods. A useful identity based on the above equation is the Bellman Recursion

$$J(t) = U(t) + gJ(t+1).$$

This work was supported by the National Science Foundation under grant ECS-9904378.

A promising collection of such approximation techniques based on estimating the function *J(t)* using this identity with neural networks as function approximators was proposed by Werbos [19][20]. These networks are often called Adaptive Critics, though this term can be applied more generally to any network that provides learning reinforcement to another entity [21]. As a practical matter, any computational structure capable of acting as a universal function approximator can be used in this role (i.e. neural networks, fuzzy rule structures, etc.). The gradient of the estimated *J(t)* can then be used to train or tune a controller. Since the gradient is the important aspect for controller training, some techniques use critics that estimate the derivatives of *J(t)* instead of the function value itself.

The standard classification of these adaptive critic methods is based on the critic's inputs and outputs. In Heuristic Dynamic Programming (HDP) the critic's outputs are estimates of the value of *J(t)*. In Dual Heuristic Programming (DHP) the critic's outputs are estimates of the derivatives of *J(t)*. In the *action dependent* versions of HDP and DHP, the critic's inputs are augmented with the controller's output (action), hence ADHDP and ADDHP.

These approaches to approximate dynamic programming utilize at least two distinct training loops, a controller training loop and a critic training loop [5][6][16]. In the neurocontrol context, the controller training loop adapts a neural network to be an approximately optimal controller. Specifically, the controller is trained to optimize the secondary utility function *J(t)* for the problem context. Since the controller outputs control actions u(t), a gradient based learning algorithm requires estimates of the derivatives $\frac{\partial J(t)}{\partial u_i(t)}$ for controller training. The critic is trained based on the consistency of its estimates through time judged using the Bellman Recursion. The exact implicit relationship is a function of the type of critic used and the structure of the primary utility function.

Our focus in this paper is on the DHP method, where the critic estimates the derivatives of J(t) with respect to the system states, i.e. $l_i(t) = \frac{\partial J(t)}{\partial R_i(t)}$. From Bellman's Recursion we have

$$\frac{\partial}{\partial R_i(t)} J(t) = \frac{\partial}{\partial R_i(t)} \big(U(t) + gJ(t+1)\big),$$

so the identity used for this critic's training is (in tensor notation)

$$l_i(t) = \frac{\partial U(t)}{\partial R_i(t)} + \frac{\partial U(t)}{\partial u_j(t)} \frac{\partial u_j(t)}{\partial R_i(t)} + g\, l_k(t+1) \left[\frac{\partial R_k(t+1)}{\partial R_i(t)} + \frac{\partial R_k(t+1)}{\partial u_m(t)} \frac{\partial u_m(t)}{\partial R_i(t)} \right].$$

To evaluate the right hand side of this equation we need a model of the system dynamics that includes all the terms from the Jacobian matrix of the coupled plant-controller system, e.g. $\frac{\partial R_j(t+1)}{\partial R_i(t)}$ and $\frac{\partial R_j(t+1)}{\partial u_i(t)}$.

Controller training then utilizes the chain rule and the system model to translate critic outputs into estimates of $\frac{\partial J(t)}{\partial u_i(t)}$, i.e.

$$\frac{\partial J(t)}{\partial u_k(t)} = \frac{\partial U(t)}{\partial u_k(t)} + g \sum l_i(t+1) \frac{\partial R_i(t+1)}{\partial u_k(t)}.$$

The entire process can be characterized as a simultaneous optimization problem; gradient based optimization of the critic function approximator together with gradient based optimization of controller parameters based on the *J(t)* estimates obtained from the critic. Different strategies have been utilized to get both these optimizations to converge. A number of authors propose alternating between optimization steps for the critic approximator and optimization of the controller [10][11][12][15]. In past work we have noted that taking simultaneous steps in both optimization processes does not appear to introduce significant instabilities into the dual convergence problem [5][6]. Since the simultaneous stepping approach is about twice as fast as the alternating approach we recommend its use.

As these techniques rely on gradient based optimization of *J(t)*, they inherently suffer from the problem of (unsatisfactory) local optima. Global optimization of *J(t)* in general is subject to the "No Free Lunch Theorem". What approximate dynamic programming techniques offer is a tractable method for local hill climbing on the *J(t)* landscape of controller parameter space. Initialized at a random point in parameter space, these methods may be trapped by a local optimum at an unsatisfactory control law. We can attempt to avoid this case by applying whatever problem specific knowledge is available a priori to the choice of initial controller parameters, in the hope of being near a satisfactorily high hill (or deep valley).

III. A Nonlinear Plant

To illustrate this technique we use a nonlinear, time-invariant 2nd order system proposed by Sanner and Slotine [14], given as

$$\ddot{y} = 4\left(\frac{\sin(4py)}{py}\right)\left(\frac{\sin(p\dot{y})}{py}\right)^2 + \big(2 + \sin(3py - 1.5p)\big)u\,.$$

Sanner and Slotine used this plant to illustrate a direct adaptive tracking control architecture based on a large identification network of Gaussian radial basis functions (RBFs). They demonstrated very precise tracking of a bandwidth limited small amplitude signal using several thousand elements in their network. More recently, Liu et al. [9] demonstrated an adaptive control scheme based on an RBF network using a variable grid method. They showed that their method could meet a somewhat less stringent error bound for tracking a sinusoidal target using several orders of magnitude fewer elements (~ 70). For the purpose of demonstrating DHP we adopt their basic problem statement and initialize the system at $y(0) = 0,\ \dot{y}(0) = 0$ with a desired trajectory given as

$$\hat{y}(t) = \sin(t).$$

The system is sampled at 20 Hz and the numerical simulation is performed with a fixed step size 4th order Runge-Kutta algorithm.

IV. Controller and Critic Structures

To keep this demonstration of using DHP to tune fuzzy controllers relatively simple, we propose a naïve Takagi-Sugeno (TS) controller for the above plant. A more sophisticated implementation would doubtlessly yield finer control, but our simple controller is adequate for our demonstration. The DHP method turns out to be quite capable of fine tuning our naïve structure to increase control quality. The controller consists of 16 rules of the form

$$u(t) = a_i y(t) + b_i \dot{y}(t) + w_i \left(y(t) - \hat{y}(t+1)\right),$$

where the time indices indicate the sampling interval, with Gaussian membership functions

$$m_i(y(t), \dot{y}(t)) = m_i(\bar{y}(t)) = \exp\left(- \frac{(\bar{y}_i - y(t))^2}{s_1} - \frac{(\dot{\bar{y}}_i - \dot{y}(t))^2}{s_2} \right).$$

These rules are uniformly spaced over the region ([-1.2, 1.2], [-1.2, 1.2]) so as to completely cover the anticipated operating and training range of the controller. Note that this is not a particularly advantageous distribution of computational resources, e.g. see [9]. While we could apply the DHP method to tuning all controller parameters, for clarity we limit the tuning process to the rule consequent parameters α_i, β_i, ω_i. Thus we have a fixed grid of rules imposed upon the plant's state space.

We construct a TS model based on this grid, to approximate the DHP critic function. The DHP critic mapping in this case takes $y(t)$, $\dot{y}(t)$, and $\left(y(t) - \hat{y}(t)\right)$ as inputs and produces $\frac{\partial J(t)}{\partial y(t)}$ and $\frac{\partial J(t)}{\partial \dot{y}(t)}$. Our critic system is therefore composed of 16 rules with consequents

$$\frac{\partial J(t)}{\partial y(t)} = l_1(t) = q_{1,i} y(t) + j_{1,i} \dot{y}(t) + y_{1,i} \left(y(t) - \hat{y}(t)\right),$$

$$\frac{\partial J(t)}{\partial \dot{y}(t)} = l_2(t) = q_{2,i} y(t) + j_{2,i} \dot{y}(t) + y_{2,i} \left(y(t) - \hat{y}(t)\right),$$

and membership functions identical to those used for the controller. Once again we choose to only tune the consequent parameters of these rules.

For our primary utility function we use

$$U(t) = (y(t) - \hat{y}(t))^2,$$

which we seek to minimize over time. The choice of an appropriate discount factor for forming *J(t)* is much less critical in DHP, where *J(t)* is not explicitly estimated, as it is in HDP where explicit estimates of *J(t)* are made. For this example, we choose a discount factor of 0.9. The final requirement for implementing DHP is to have a differentiable model of the plant from which to obtain partial derivatives. In the present example we have an analytic expression for the plant and hence could evaluate the derivatives explicitly. More realistically, we have to estimate a plant model.

V. Constructing a Model

It is relatively straight forward to construct an appropriate model for this DHP training context. We use a TS model containing 16 rules with consequents of the form

$$y(t + \Delta t) = z_{1,i} y(t) + h_{1,i} \dot{y}(t) + x_{1,i} u(t),$$

$$\dot{y}(t + \Delta t) = z_{2,i} y(t) + h_{2,i} \dot{y}(t) + x_{2,i} u(t),$$

and membership functions identical to those of the 16 controller and critic rules. Training is based on minimizing squared one-step prediction error using a backpropagation of error approach. The model was trained for five passes through a set of 10,000 randomly sampled input/output pairs. We can use our a priori expectations of the consequent coefficient's values to assign preliminary values before training begins. Since the plant is a 2nd order system, we can assert that

$$z_{1,i} \approx 1, \quad h_{1,i} \approx \Delta t, \text{ and } h_{2,i} \approx 1.$$

The rest of the coefficients are initialized to zero.

After training, we take advantage of this model structure by noting that the coefficients in the output model of each rule correspond to local approximations of the partial derivatives we need to obtain during DHP tuning. For example, we can obtain

$$\left.\frac{\partial y(t+1)}{\partial u(t)}\right|_{(\bar{y}(t))} = \frac{\sum_i m_i(\bar{y}(t)) x_{1,i}}{\sum_i m_i(\bar{y}(t))}.$$

Our previous work with such models showed that as long as the approximate derivative values obtained in the above manner had the correct sign (positive or negative) *most* of the time, the model was adequate for use in DHP [16] [17].

VI. The Tuning Sequence and Results

Prior to tuning, all the controller consequent parameters were prestructured using the local models embodied in the above model. For each local model we write the approximate local control law

$$u(t) = \frac{1}{x_{1,i}} \left(- \left(z_{1,i} - 1\right) y(t) - h_{1,i} \dot{y}(t) - \left(y(t) - y(t+1)\right) \right),$$

which provides us with the initial values for the controller consequent parameters

$$a_i = \frac{-\left(z_{1,i} - 1\right)}{x_{1,i}}, \quad b_i = \frac{-h_{1,i}}{x_{1,i}}, \quad \text{and } w_i = \frac{-1}{x_{1,i}}.$$

All critic consequent parameters were initially set to zero. The plant simulation was then initialized and run for 50 seconds with the controller providing *u(t)*, and both controller and critic updates taking place each sampling period. Training commenced in this fashion with the plant reinitialized every 50 seconds. The specific simulation/update sequence used was:

1) calculate control (t);
2) simulate one sampling interval (t);

3) evaluate critic (at new state = t+1);
4) calculate target critic value (for old state = t);
5) calculate controller error signal (t);
6) update controller (t);
7) evaluate critic (at old state = t);
8) update critic using difference between actual and target (t);

The controller update equations are of the form

$$a_i(t+1) = a_i(t) + \left(\frac{lc * m_i(\bar{y}(t))}{\sum_k m_k(\bar{y}(t))} \right) * \left(\frac{\partial J(t)}{\partial u(t)} \right) * y(t),$$

where *lc* is the learning coefficient, and

$$\frac{\partial J(t)}{\partial u(t)} = l_1(t+1)\frac{\partial y(t+1)}{\partial u(t)} + l_2(t+1)\frac{\partial \dot{y}(t+1)}{\partial u(t)}.$$

The critic update equations are of the form

$$q_{j,i}(t+1) = q_{j,i}(t) + \left(\frac{lc * m_i(\bar{y}(t))}{\sum_k m_k(\bar{y}(t))} \right) * \left(\hat{l}_j(t) - l_j(t) \right) * y(t),$$

where *lc* is the learning coefficient, and the $\hat{l}_j(t)$ are of the form

$$\hat{l}_1(t) = \frac{\partial U(t)}{\partial y(t)} + 0.9\left[l_1(t+1)\left(\frac{\partial y(t+1)}{\partial y(t)} + \frac{\partial y(t+1)}{\partial u(t)}\frac{\partial u(t)}{\partial y(t)} \right) + l_2(t+1)\left(\frac{\partial \dot{y}(t+1)}{\partial y(t)} + \frac{\partial \dot{y}(t+1)}{\partial u(t)}\frac{\partial u(t)}{\partial y(t)} \right) \right].$$

The average RMS error of the controller during the tuning process is shown in Figure 1. Average tracking error declined by 60%, from 0.00208 to 0.00084. Maximum pre-tuning overshoot was 0.00548 compared to 0.00228 after tuning (a 58% decline), while maximum undershoot decreased from 0.00661 to 0.00253 (a 62% decline). A comparison of the pre and post tuning tracking error over one period of the target trajectory is shown in Figure 2. While the initial controller performance could be deemed satisfactory in some contexts, DHP tuning improves controller performance by 60%.

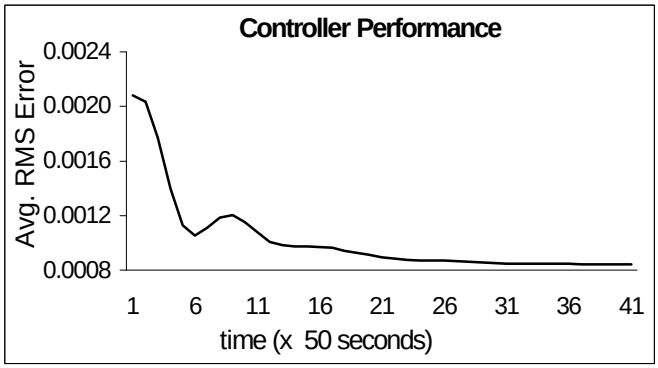

Fig. 1. Controller Performance Curve

VII. Conclusion

We have demonstrated that the DHP methodology (a model-based adaptive critic method) can successfully tune a fuzzy Controller using a fuzzy plant model. While this approximate dynamic programming technique has previously been used for training neurocontrollers, embedding DHP in a fuzzy framework more easily allows *a priori* knowledge to be used. In particular, use of first order TS models offers a direct approach to representing the characteristics of the plant relevant to the needs of model based approximate dynamic programming. The TS models used also permit a simple approach to prestructuring the controller.

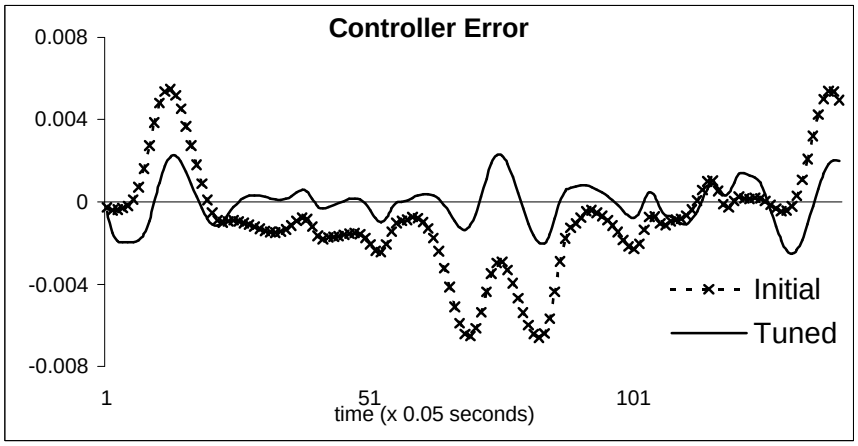

Fig. 2. Controller Tracking Error

Our belief is that DHP and related techniques are ideally suited for neuro-fuzzy hybrid implementations. Fuzzy controllers and models more easily allow the incorporation of a priori knowledge, while neural networks may be more natural as function approximators in the critic role. While the above demonstration was limited to tuning consequent parameters, DHP can also be used for training membership parameters if the structure of the fuzzy rule base is suitable. Thus DHP and related techniques should be helpful for state space segmentation and partitioning. Another feature of adaptive critic based approximate dynamic programming techniques is the potential to use the critic function as a guarantor of system stability, e.g. [10][13].

It is also important to notice the applicability of these techniques to adaptive control problems. Our current example illustrated DHP for tuning a controller for a time invariant plant. For non-stationary plants, the controller, critic and model can all be continuously update on-line to track changes in the plant's dynamics [5].

REFERENCES

[1] Barto, A.G., R.S. Sutton & C.W. Anderson, "Neuron-like Adaptive Elements That Can Solve Difficult Learning Control Problems", *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 13, pp. 834-846, 1983.

[2] Bellman, R.E., *Dynamic Programming*, Princeton University Press, 1957.

[3] Berenji, H.R. & P. Khedkar, "Learning and Tuning Fuzzy Logic Controllers Through Reinforcement", *IEEE Transactions on Neural Networks*, 1992.

[4] Lee, C.C., "A Self-Learning Rule-Based Controller Employing Approximate Reasoning and Neural Net Concepts", *International Journal of Intelligent Systems*, Vol. 6, pp.71-93, 1991.

[5] Lendaris, G. & T. Shannon, "Designing (Approximate) Optimal Controllers Via DHP Adaptive Critics & Neural Networks", in *The Handbook of Applied Computational Intelligence*, Karayiannis, Padgett & Zadeh, eds., CRC Press, to appear 2000.

[6] Lendaris, G.G., T.T. Shannon & A. Rustan, "A Comparison of Training Algorithms for DHP Adaptive Critic Neuro-Control", in *Proceedings IJCNN'99*, Washington D.C., IEEE, 1999.

[7] Lin, C.T. & C.S.G. Lee, *Neural Fuzzy Systems*, Prentice Hall, Upper Saddle River, NJ, 1996.

[8] Lin, C.T., & Y.C. Lu, "A Neural Fuzzy System with Linguistic Teaching Signals", *IEEE Transactions on Fuzzy Systems*, Vol. 3, # 2, 1995.

[9] Liu, G.P., V. Kadirkamanathan & S.A. Billings, "Variable Neural Networks for Adaptive Control Of Nonlinear Systems", *IEEE Transactions on Systems Man & Cybernetics - Part C*, vol. 29, #1, February 1999, pp. 34-43.

[10] Prokhorov, D., *Adaptive Critic Designs and their Application*, Ph.D. Dissertation, Department of Electrical Engineering, Texas Tech University, 1997.

[11] Prokhorov, D. & D. Wunsch, "Adaptive Critic Designs", *IEEE Transactions on Neural Networks*, vol. 8 (5), 1997, pp. 997-1007.

[12] Prokhorov, D., R. Santiago & D. Wunsch, "Adaptive Critic Designs: A Case Study for Neurocontrol", *Neural Networks*, vol. 8 (9), pp. 1367 - 1372, 1995.

[13] Prokhorov, D.V. & L.A. Feldkamp, "Generalized Adaptive Critics and their Applications", presented at IJCNN'99, session 6.5, Washington D.C., July 1999.

[14] Sanner, R.M. & J.J.E. Slotine, "Gaussian Networks for Direct Adaptive Control", *IEEE Transactions on Neural Networks*, vol. 13, pp. 837-863, June 1992.

[15] Santiago, R. & P.J. Werbos, "New Progress Towards Truly Brain-Like Control", *Proceedings of WCNN'94*, San Diego, CA, 1994, pp. 27-33.

[16] Shannon T.T., "Partial, Noisy and Qualitative Models for Adaptive Critic Based Neurocontrol", in *Proceedings of IJCNN'99*, Washington D.C., IEEE, 1999.

[17] Shannon T.T., G.G. Lendaris "Qualitative Models for Adaptive Critic Neurocontrol", in *Proceedings of SMC'99*, IEEE, 1999.

[18] Takagi, T, & M. Sugeno, "Fuzzy Identification of Systems and its Application to Modeling and Control", *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 15, # 1, 1985.

[19] Werbos, P.J., "A Menu of Designs for Reinforcement Learning Over Time", in Miller, W.T., R.S. Sutton, & P.J. Werbos eds., *Neural Networks for Control*, MIT Press, Cambridge, MA, 1990, pp. 67- 95.

[20] Werbos, P.J., "Approximate Dynamic Programming for Real-Time Control and Neural Modeling", in D.A. White & D.A. Sofge eds., *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*, Van Nostrand Reinhold, New York, 1992, pp. 493 - 525.

[21] Widrow, B., N Gupta & S. Maitra, "Punish/Reward: Learning with a Critic in Adaptive Threshold Systems", *IEEE Transactions on Systems, Man & Cybernetics*, vol. 3 (5), pp. 455-465, 1973.

[22] Yen, J. & R. Langari, *Fuzzy Logic*, Prentice Hall, Upper Saddle River, NJ, 1999.