Self-tuning Neurofuzzy Control for Nonlinear Systems with Offset

C. W. Chan, X. J. Liu, and W. K. Yeung
Department of Mechanical Engineering
The University of Hong Kong
Pokfulam Road, Hong Kong, China
E-mail: mechan@hkucc.hku.hk

Abstract
A self-tuning neurofuzzy controller with an ability to remove offsets is derived in this paper based on the self-tuning integrating controller derived for the local linear model. The training target for the proposed controllers is derived, and they can be trained by the simplified recursive least squares (RLS) method with a computing time that is linear instead of geometric in the number of weights in the network. Further, the simplified RLS method not only has the same convergence property as the RLS method, it also has a better ability in tracking varying parameters. The performance of the self-tuning neurofuzzy controller is illustrated by examples involving both linear and nonlinear systems.

Keywords: Self-tuning controllers, integrating controllers, nonlinear controllers, neurofuzzy networks.

1. Introduction

The development of nonlinear controllers based on neural networks has attracted considerable interest recently. The main motivation to use neural networks for the implementation of nonlinear controllers is their ability to approximate both linear and nonlinear systems with arbitrary accuracy, and to be trained from experimental data [1, 2]. In general, neural network based controllers are implemented using one or two networks. In the indirect method, two neural networks are usually used, one trained offline to model the system or the inverse of the system, and the other trained offline or online to implement the controller [3]. In [4], neural network based controllers involving a conventional feedback controller is proposed. The neural networks are trained by the feedback-error-learning method either to mimic the inverse of the system, or a nonlinear regulator. An adaptive model reference control scheme based on neural networks is presented in [5]. The neural network controller is trained such that the output of the system follows the output of a given reference model. Single layer neural networks are used in these schemes, and are trained by the slow gradient method.

Neurofuzzy networks have a number of attractive properties to be used in implementing nonlinear controllers. Similar to other neural networks, they can approximate nonlinear functions with arbitrary accuracy [12]. They have compact supports, and are linear-in-weights networks. The latter property enables the weights of neurofuzzy networks to be estimated using well-developed linear parameter estimation methods. To estimate these weights on-line, recursive least squares (RLS) method can be used. However, updating these weights on-line can be time consuming, as the number of weights increases geometrically as the complexity of the network increases. To reduce the computing time, a simplified RLS method is proposed in [6]. From the compact property of neurofuzzy networks, it is shown that there are a large number of zeros in the transformed input of the network. Consequently, it is proposed in [6] to simplify the RLS method with only elements of the covariance matrix associated with non-zero elements of the transformed input are updated, yielding a computing time linear instead of geometric in the number of weights. It is further shown that the convergence of the simplified RLS method is the same as the RLS method.

Since neurofuzzy networks can be interpreted as a network that consists of local models with a smooth transition between them [8], neurofuzzy controllers can be designed by adopting the same design for local linear controllers. As self-tuning controllers are simple to implement and can be trained on-line [9], they are chosen in [7] to be the local linear controllers to develop the self-tuning neurofuzzy controllers. The self-tuning neurofuzzy controllers derived in [9] are based on the generalized minimum variance control law. In this paper, self-tuning integrating controllers that have the ability to remove offset in the nonlinear system [10] are considered.

The structure of this paper is as follows. In Section 2, a brief description of the integrating controllers is presented, followed by the derivation of the neurofuzzy controllers and its training target in Section 3. In Section 4, the on-line training of the controllers is presented. The implementation and the performance of the self-tuning controllers are illustrated by two simulation examples, one involving a linear system, and the other, a nonlinear system, as

presented in Section 5.

2. Integrating Controllers for Local Linear Models

A nonlinear system with finite dimension can be described by [11],

$$y(t) = f[y(t-1), y(t-2), \cdots, y(t-n), u(t-k), u(t-k-1), \cdots, u(t-k-m)] \quad (1)$$

where $\{u(t)\}$, $\{y(t)\}$ are the input and output respectively; $f(.)$ is an unknown smooth nonlinear function; n, m, and k are respectively the orders, and the time delay of the system, which are assumed known. As $f(.)$ is a smooth function, a local linear model can be obtained from the linear term of the Taylor series expansion of $f(.)$. A simple, yet effective approach to control nonlinear systems is to switch between local linear controllers designed at specified operating points. The main advantage of this approach is that only linear controller design techniques are involved. However, the drawback is that the control during the transition from one local model to another may not always be smooth. In contrast, much smoother transition can be obtained from neurofuzzy networks, as they are able to approximate smooth nonlinear functions with arbitrary accuracy [12]. For this reason, neurofuzzy networks are chosen here to implement the nonlinear controllers, which is discussed in details in the following section. A brief description of the integrating controller is presented in this section, and the reader is referred to [10] for further details.

Let the i^{th} local linear models be given by

$$A^i(z^{-1})y(t) = z^{-k}B^i(z^{-1})u(t) + C^i(z^{-1})e(t)/\Delta \quad (2)$$

where $e(t)$ is a white noise with zero mean and a variance of σ^2; z^{-1}, the backward-shift operator; k, the delay; $\Delta = 1 - z^{-1}$, the differencing operator, and

$$A^i(z^{-1}) = 1 + a_1^i z^{-1} + \cdots + a_n^i z^{-n},$$

$$B^i(z^{-1}) = b_0^i + b_1^i z^{-1} + \cdots + b_m^i z^{-m} \; ; \; b_0^i \neq 0$$

Rearranging (2) gives,

$$\overline{A}^i(z^{-1})y(t) = z^{-k}B^i(z^{-1})\Delta u(t) + C^i(z^{-1})e(t) \quad (3)$$

where $\overline{A}^i(z^{-1}) = A^i(z^{-1})\Delta$ and $\Delta u(t) = u(t) - u(t-1)$. As the control in (3) is in incremental form, an integrator is being introduced into the system for removing offsets. The integrating controller is obtained by minimizing the following cost function.

$$J = E[\phi^{i\,2}(t+k)] \quad (4)$$

where $\phi^i(t)$ is the auxiliary output given by

$$\phi^i(t+k) = P^i y(t+k) + Q^i u(t) - R^i r(t) \quad (5)$$

P^i, Q^i and R^i are polynomials in z^{-1} and r(t) is the set point. The integrating controller is derived by splitting $\phi^i(t+k)$ into two terms, one of which is set to zero by the integral control $\Delta u(t)$, whilst the other contains the white noise $\{e(t+1), \ldots, e(t+k)\}$. This is achieved by using the Diophantine equation,

$$P^i C^i = E^i \Delta A^i + z^{-k} G^i \quad (6)$$

where E^i and G^i are polynomials in z^{-1} with orders k-1 and n_y respectively,

$$E^i(z^{-1}) = 1 + e_1^i z^{-1} + \cdots + e_{k-1}^i z^{-(k-1)}$$

$$G^i(z^{-1}) = g_0^i + g_{11}^i z^{-1} + \cdots + g_{n_y}^i z^{-n_y}$$

$$n_y = \max(n, n_p + n_c - k)$$

From (2) and (6), $\phi^i(t)$ can be rewritten as,

$$\phi^i(t+k) = \frac{1}{C^i}[G^i y(t) + C^i Q^i u(t) - C^i R^i r(t) + B^i E^i \Delta u(t)] + E^i e(t+k) \quad (7)$$

As the first term on the right hand side of (7) is uncorrelated with the term containing $e(t+k)$, $\phi^i(t)$ is minimized by setting this term to zero, giving

$$C^i Q^i u(t) + B^i E^i \Delta u(t) + G^i y(t) - C^i R^i r(t) = 0 \quad (8)$$

Let $Q^i = Q^{i'}\Delta$, (8) can be rewritten as

$$(C^i Q^{i'} + B^i E^i)\Delta u(t) + G^i y(t) - C^i R^i r(t) = 0 \quad (9)$$

or $$F^i \Delta u(t) + G^i y(t) + H^i r(t) = 0 \quad (10)$$

where $F^i = C^i Q^{i'} + B^i E^i$, and $H^i = -C^i R^i$. From (10), the integrating controller is

$$\Delta u(t) = -\overline{F}^i(z^{-1})\Delta u(t) - \overline{G}^i(z^{-1})y(t) - \overline{H}^i(z^{-1})r(t) \quad (11)$$

where

$$\overline{F}^i(z^{-1}) = F^i(z^{-1})/f_0^i - 1 = \sum_{j=1}^{n_{\Delta u}} \overline{f}_j^i z^{-1}$$

$$\overline{G}^i(z^{-1}) = G^i(z^{-1})/f_0^i = \sum_{j=0}^{n_y} \overline{g}_j^i z^{-1}$$

$$\overline{H}^i(z^{-1}) = H^i(z^{-1})/f_0^i = \sum_{j=0}^{n_r} \overline{h}_j^i z^{-1}$$

f_0^i is the leading constant of $F^i(z^{-1})$ and $n_y, n_{\Delta u}$ are the orders of the system.

3. Neurofuzzy Controllers Based on Local Integrating Controllers

A nonlinear controller based on the linear integrating controller (11) can be expressed as follows,

$$\Delta u(t) = f[y(t), \cdots, y(t-n_y), r(t), \cdots, r(t-n_r), \Delta u(t-1), \cdots \Delta u(t-n_{\Delta u})] \quad (12)$$

where the notations are defined previously. The nonlinear controller (12) is to be implemented by a neurofuzzy network, and under certain conditions, it is

a linear-in-weights network [8], as given below.

$$\Delta u(t) = a^T(x(t))\theta \tag{13}$$

where $\theta = [\theta_1, \theta_2, \ldots, \theta_p]^T$ is the weight vector, x(t) = $[y(t), \ldots, y(t - n_y), r(t), \ldots, r(t - n_r), \Delta u(t - 1), \ldots, \Delta u(t - n_{\Delta u})]^T$, the input vector and $a(x(t))$, the transformed input vector. The dimension of $x(t)$, n, is given by

$$n = n_y + n_r + n_{\Delta u} + 2 \tag{14}$$

and the number of weights, p, in the network is

$$p = (R_y + \rho_y)^{n_y+1}(R_r + \rho_r)^{n_r+1}(R_{\Delta u} + \rho_{\Delta u})^{n_{\Delta u}} \tag{15}$$

where ρ_y, ρ_r, $\rho_{\Delta u}$ are the respective order of the basis functions for $y(t)$, $r(t)$ and $\Delta u(t)$, and R_y, R_r and $R_{\Delta u}$, the respective number of inner knots. The transformed input, $a_i(x(t))$, is the tensor products of the univariate B-spline basis functions $\mu_{A_k^i}(x_k(t))$,

$$a_i(x(t)) = \prod_{k=1}^{n} \mu_{A_k^i}(x_k(t)) \tag{16}$$

From (13), the auxiliary output $\phi(t+k)$ becomes

$$\phi(t+k) = [\Delta u(t) - a^T(x(t))\theta] + Ee(t+k) \tag{17}$$

Rearranging (17) yields,

$$a^T(x(t))\theta = \Delta u(t) - \phi(t+k) + Ee(t+k) \tag{18}$$

As the mean of $e(t)$ is zero, and likewise, the mean of $Ee(t+k)$ is also zero. As $e(t)$ is assumed to be a white noise, $Ee(t+k)$ is uncorrelated with the other terms on the right-hand side of (18). The training target of the neurofuzzy controller, denoted by $\psi(t)$, can be derived from (18) as

$$\psi(t) = \Delta u(t-k) - \phi(t) \tag{19}$$

where $\phi(t+k)$ is computed by (5) for a given $P(z^{-1})$, $Q(z^{-1})$, and $R(z^{-1})$.

4. On-line Training of the Neurofuzzy Controllers

On-line training of the neurofuzzy controller (13) involves updating θ recursively at each sampling interval. The RLS estimate of $\theta(t)$ is given by [10],

$$\left.\begin{aligned} \theta(t) &= \theta(t-1) + \frac{P(t-1)a(x(t-k))[\psi(t) - a^T(x(t-k))\theta(t-1)]}{1 + a^T(x(t-k))P(t-1)a(x(t-k))} \\ P(t) &= P(t-1) - \frac{P(t-1)a(x(t-k))a^T(x(t-k))P(t-1)}{1 + a^T(x(t-k))P(t-1)a(x(t-k))} \end{aligned}\right\} \tag{20}$$

where $P(t)$ is the covariance matrix at t. From the compact support property of neurofuzzy networks [8], the number of nonzero elements in $a(x(t))$ is

$$p' = \rho_y^{n_y+1}\rho_r^{n_r+1}\rho_{\Delta u}^{n_{\Delta u}} \tag{21}$$

As an example, the neurofuzzy network shown in Fig. 2 consists of two inputs fuzzified by second order basis functions, i.e., $n_y = n_r = n_{\Delta u} = 0$, and $\rho_y = \rho_r = 2$. Let the number of inner knots for y be chosen to be 5, and that for w be chosen to be 4. From (15) and (21), p and p' are 42 and 4 respectively. From Fig. 2, only the basis functions associated with the four squares surrounding "1" are non-zero, when the knots y_2 and w_2 are activated. Similar results are obtained for knots $\{y_2, w_3\}$, $\{y_3, w_2\}$, and $\{y_3, w_3\}$, showing that, in this example, two inner knots of each variable are activated each time. For the "x" shown in Fig. 2, the knots $\{y_2, y_3\}$ in y and $\{w_2, w_3\}$ in w are activated. As the activated regions overlap each other, only nine squares are activated, as shown in Fig. 2. For convenience, let the non-zero elements, denoted by $a_1(x(t))$, be arranged at the top of $a(x(t))$,

$$a(x(t)) = [a_1^T(x(t)) \;\; 0^T]^T \tag{22}$$

where 0 is a column zero vector of dimension $(p - p')$. Let $\theta(t)$ and $P(t)$ be similarly rearranged,

$$\left.\begin{aligned} \theta(t) &= [\theta_1^T(t) \;\; \theta_2^T(t)]^T \\ P(t) &= \begin{bmatrix} P_{11}(t) & P_{12}(t) \\ P_{21}(t) & P_{22}(t) \end{bmatrix} \end{aligned}\right\} \tag{23}$$

From (22) and (23), (20) can be rewritten as,

$$\theta_1(t) - \theta_1(t-1) = \left(\frac{P_{11}(t-1)a_1(x(t-k)) - \frac{P_{11}(t-1)a_1(x(t-k))a_1^T(x(t-k))P_{11}(t-1)a_1(x(t-k))}{1 + a_1^T(t-k)P_{11}(t-1)a_1(t-k)}\right)\xi(t) \tag{24a}$$

$$\theta_2(t) - \theta_2(t-1) = \left(\frac{P_{21}(t-1)a_1(x(t-k)) - \frac{P_{21}(t-1)a_1(x(t-k))a_1^T(x(t-k))P_{11}(t-1)a_1(x(t-k))}{1 + a_1^T(x(t-k))P_{11}(t-1)a_1(x(t-k))}\right)\xi(t) \tag{24b}$$

$$\left.\begin{aligned} P_{11}(t) &= P_{11}(t-1) - \frac{P_{11}(t-1)a_1(x(t-k))a_1^T(x(t-k))P_{11}(t-1)}{1 + a_1^T(x(t-k))P_{11}(t-1)a_1(x(t-k))} \\ P_{12}(t) &= P_{21}^T(t) = P_{12}(t-1) - \frac{P_{11}(t-1)a_1(x(t-k))a_1^T(x(t-k))P_{12}(t-1)}{1 + a_1^T(x(t-k))P_{11}(t-1)a_1(x(t-k))} \end{aligned}\right\} \tag{25a}$$

$$P_{22}(t) = P_{22}(t-1) - \frac{P_{21}(t-1)a_1(x(t-k))a_1^T(x(t-k))P_{12}(t-1)}{1 + a_1^T(x(t-k))P_{11}(t-1)a_1(x(t-k))} \tag{25b}$$

where $\xi(t) = \psi(t) - a_1^T(x(t-k))\theta_1(t-1)$. Let $\theta_2(t)$ be the weights of all basis functions that have not been

activated, and the choice of the initial choice of the covariance matrix, $P(0)$, be a diagonal matrix. Then $P_{12}(0) = P_{21}(0) = 0$, and $P_{22}(0) = \lambda I$, where λ is a constant, and I, a unit matrix. From (25), $P_{12}(t)$, $P_{21}(t)$, and $P_{22}(t)$, and hence $\theta_2(t)$ remain unchanged from their initial values. Further, they do not affect the estimate of the weights associated with basis functions that have been activated. It implies that the input domain, or the number of inner knots, of the input variables can be expanded at any time without re-computing both $P(t)$ and $\theta(t)$. In other words, the structure of the neurofuzzy controllers can be expanded without re-training if the RLS method is used with $P(0)$ set to a diagonal matrix. This unique property of neurofuzzy networks does not generally apply to other neural networks.

4.1 Simplified Recursive Least Squares Method

As the number of weights p given by (15) can be quite large, updating θ on-line may be quite time consuming. To reduce computing time, the RLS method can be simplified using the local change property of neurofuzzy networks discussed earlier. Instead of updating the whole $P(t)$, only $P_{11}(t)$, $P_{12}(t)$ and $P_{21}(t)$ are updated, as indicated by the shaded areas in Fig. 3. In this case, only $p'(p-(p'-1)/2)$ elements of $P(t)$ are computed instead of $p(p+1)/2$ elements in the RLS method, and the computing time is now linear in the number of weights. The saving in computing time is significant, especially if p is much larger than p'. Rearranging (25b) gives,

$$P_{22}(t) - P_{22}(t-1) = -\frac{P_{21}(t-1)a_1(x(t-k))a_1^T(x(t-k))P_{12}(t-1)}{1 + a_1^T(x(t-k))P_{11}(t-1)a_1(x(t-k))} \quad (26)$$

From (26), the update of $P_{22}(t)$ involves subtracting a positive definite matrix from $P_{22}(t\text{-}1)$. Consequently, if $P_{22}(t)$ is not updated, it is equivalent to adding a positive definite matrix to it. As the right hand side of (26) approaches zero as time tends to infinity [6], adding a positive definite sub-matrix to P(t) at each sampling interval is effectively adding a positive definite matrix to P(t) over the whole training period. Since it is well known that adding a positive definite matrix to $P(t)$ does not alter the convergence of the RLS estimate [12], the simplified RLS method has the same convergence property as the RLS method. A further advantage of adding a positive definite matrix to the covariance matrix is that its ability to track varying parameters is enhanced [7].

4.2 On-line training procedure

The on-line training of the self-tuning neurofuzzy controllers can be summarized below.

(1) Select (i) the number of input variables, i.e., n and m, the delay k in eqn. 1, (ii) the order of the basis functions, the range and the number of inner knots for each input variable, (iii) $P(z^{-1})$, $Q(z^{-1})$ and $R(z^{-1})$ in the generalized system output given by (5).
(2) Initialize $\theta(0)$, say to 0.1, and $P(0)$, say to $100I$.
(3) Measure the output of the system $y(t)$, and update $\theta(t)$ by (24), and $P(t)$ by (25a).
(4) Compute the control $u(t)$ by (13).
(5) Repeat steps (3) and (4).

5. Simulation Examples

Two examples, one involving a linear system, and the other, a nonlinear system are presented. As expected, the performance of the self-tuning neurofuzzy controller is similar to the self-tuning controller in the linear case, but is superior to the self-tuning controller in the nonlinear case.

Example 1 Linear system

Consider the following linear system,

$$y(t) = 1.5y(t-1) - 0.7y(t-2) + u(t-1) + 0.5u(t-2) + e(t)/\Delta \quad (27)$$

where e(t) is a normally distributed noise with zero mean and a standard deviation of 1. Let P=1, R=1 and Q=0, then $\phi(t)$ becomes

$$\phi(t) = y(t) - r(t-1) \quad (28)$$

Assuming (27) is known, the integrating controller obtained from (11) is,

$$\Delta u(t) = -[2.5y(t) - 2.2y(t-1) + 0.7y(t-2) + 0.5\Delta u(t-1) - r(t)] \quad (29)$$

As the parameters in system (27) are assumed to be unknown, the controller parameters in (29) are estimated recursively on-line by the RLS method given by (20) with $\theta(0) = 0.1$, and $P(0) = 100I$. The set point $r(t)$ is a triangular wave given by

$$r(t) = \begin{cases} 0.2rem(t/100) & if \quad 0 \le rem(t/100) < 50 \\ 10 - 0.2rem(t/100) & if \quad 50 \le rem(t/100) < 100 \end{cases} \quad (30)$$

The system output using the integrating controller is shown in Fig. 4. The self-tuning neurofuzzy controller (13) is implemented with x(t) given by

$$x(t) = [y(t), y(t-1), y(t-2), r(t), \Delta u(t-1)] \quad (31)$$

As (27) is linear, two triangular basis functions are selected for each input, giving $\rho_y = \rho_r = \rho_u = 2$, and $R_y = R_r = R_{\Delta u} = 0$. From (15), the number of weights of the neurofuzzy network, p, is 32. The range of $y(t)$ and $r(t)$ is chosen to be between –5 and 10, whilst that for $\Delta u(t)$ is between –5 and 5. The initial weights $\theta(0)$ are set to 0.1, and the initial covariance matrix, $P(0)$, to 100I. The system output for the neurofuzzy controller

is shown in Fig. 5. From the accumulated cost functions shown in Fig. 6, the performance of the self-tuning integrating controller and the self-tuning neurofuzzy controller are similar.

Example 2 Nonlinear System

Consider the following nonlinear model,

$$y(t) = 0.3y(t-1) + 0.6y(t-2) + [u(t-1)]^{1/3} + e(t)/\Delta \quad (32)$$

where $e(t)$ is zero mean with a variance of 1. Again, the generalized output $\phi(t)$ is

$$\phi(t) = y(t) - r(t-1) \quad (33)$$

The set point is the square wave given by

$$r(t)=7.5+2.5\text{sign}(\cos(2\pi t/100)) \quad (34)$$

The integrating control law is

$$\bar{f}_0\Delta u(t) - \bar{g}_0 y(t) - \bar{g}_1 y(t-1) - \bar{g}_2 y(t-2) - r(t) = 0 \quad (35)$$

Using the same initial values as in Example 1, the output using the self-tuning integrating controller is shown in Fig. 7. Large oscillations after step changes are observed.

The simulation is repeated using the neurofuzzy controller (13) implemented with x(t) given by

$$\mathrm{x}(t) = [y(t), y(t-1), y(t-2), r(t)] \quad (36)$$

The training target of the neurofuzzy controller is

$$\psi(t) = \Delta u(t-1) - \phi(t) \quad (37)$$

Triangular basis functions is used for each input, giving $\rho_y = \rho_r = 2$, and $R_y = R_r = 1$. The range of $y(t)$ and $r(t)$ are selected to be between 3 and 12. The number of weights of the neurofuzzy network is 81. The output using the self-tuning neurofuzzy controller is shown in Fig. 8, and is less oscillatory, and much better than that using the self-tuning integrating controller.

6. Conclusion

A self-tuning neurofuzzy controller with the ability to eliminate offsets is derived based on self-tuning integrating controllers for the local linear model. It is shown that the proposed controllers can be trained on-line using the simplified RLS method. Not only the computing time can be significantly reduced, the simplified RLS method also has a better parameter tracking ability. This is because updating the covariance matrix in the RLS method partially has the effect of adding a positive definite matrix to the covariance matrix. The implementation and the performance of the self-tuning neurofuzzy controllers are illustrated by simulation examples involving both a linear and a nonlinear system. As expected, the performance of the self-tuning neurofuzzy controller for the linear system is similar to that of the self-tuning controller, but is superior to the self-tuing controller for the nonlinear system.

References

[1] K. J. Hunt, D. Sbarbaro, R. Zbikowski and P. J. Gawthrop, "Neural Networks for Control Systems – A Survey," *Automatica, 28* (6), 1992, pp. 1083-1112.

[2] F. L. Lewis, "Nonlinear Network Structures for Feedback Control," *Asian J. of Control, 1*(4), 1999, pp. 205-228.

[3] K. S. Narendra and K. Parthasarathy, "Identification and Control for Dynamic Systems Using Neural Networks," *IEEE Trans. on Neural Networks, 1*, 1990, pp. 4-27.

[4] H. Gomi and M. Kawato, "Neural network control for a closed-loop system using feedback-error-learning for neural control of nonlinear systems," *Neural Network*, 6, 1993, pp. 933-946.

[5] M. Yuan, A. N. Poo, and G. S. Hong, "Direct neural control system: Nonlinear extension of adaptive control," *IEE Proc.-Contr. Theory Appl.*, 142(6), 1995, pp. 661-667.

[6] C. W. Chan, K. C. Cheung, and W. K. Yeung, "A computation-efficient on-line training algorithm for neurofuzzy networks," *Int. J. Sys. Sciences*, 31, 2000, pp. 297-306.

[7] W. K. Yeung, C. W. Chan, and K. C. Cheung, "Self-tuning control for nonlinear systems based on neurofuzzy networks," *UKACC Int. Conf. on Control*, Cambridge, UK, 4-7 Sept. 2000.

[8] M. Brown, and C. J. Harris, *Neurofuzzy Adaptive Modelling and Control*, Prentice Hall, 1994.

[9] D. W. Clarke, and P. J. Gawthrop, 'Self-tuning controller,' *Proc. IEE*, 122, 1975, pp. 929-934.

[10] D. W. Clarke, and P. S. Tuffs, "Self-tuning control of offset: a unified approach," *Proc. IEE, Pt. D*, 132, 1985, pp.100-110.

[11] S. A. Billings, and W. S. F. Voon, "Piecewise linear identification of non-linear systems," *Int. J. Control, 46*(1), 1987, pp. 215-235.

[12] Kosko B., *Neural Networks And Fuzzy Systems*, Prentice-Hall, 1992.

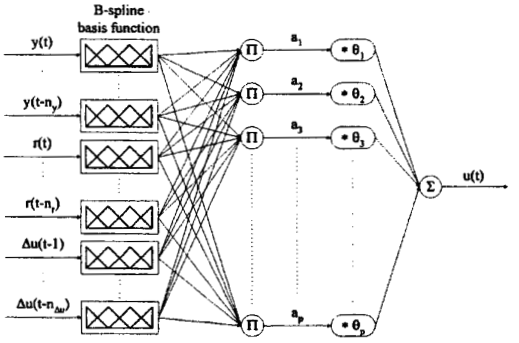

Fig. 1 Neurofuzzy controller

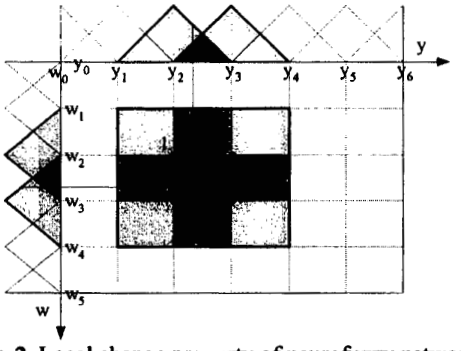

Fig. 2 Local change property of neurofuzzy networks

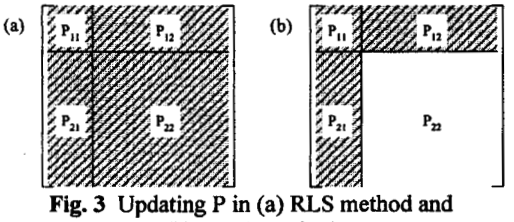

Fig. 3 Updating P in (a) RLS method and (b) SRLS method

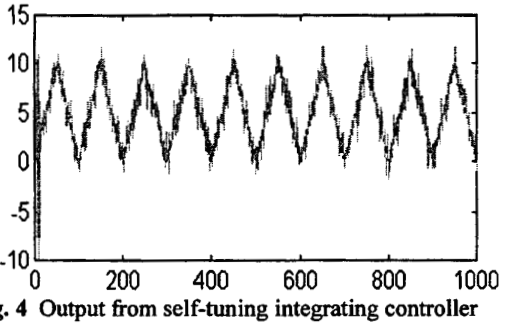

Fig. 4 Output from self-tuning integrating controller

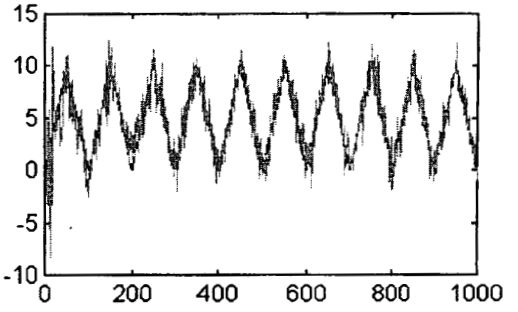

Fig. 5 Output from self-tuning neurofuzzy controller

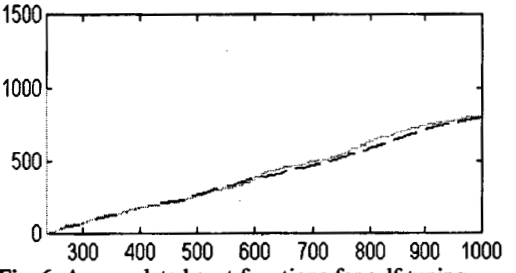

Fig. 6 Accumulated cost functions for self-tuning integrating controller (solid line) and self-tuning neurofuzzy controller (dashed line)

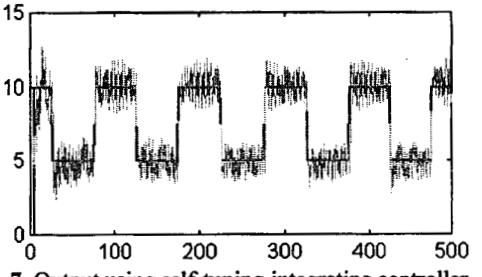

Fig. 7 Output using self-tuning integrating controller for nonlinear system

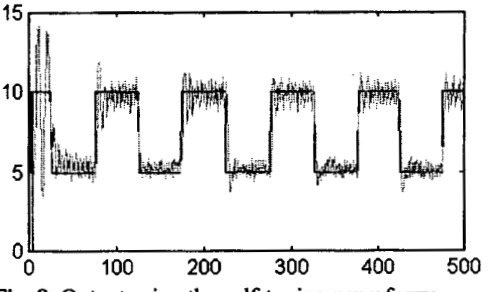

Fig. 8 Output using the self-tuning neurofuzzy integrating controller