Stable Neural-Adaptive Control of Activated Sludge Bioreactors

C.J.B. Macnab

Abstract—This paper proposes an adaptive neural network control for an activated sludge bioreactor used for wastewater treatment. The novel method prevents weight drift and associated bursting when a persistent disturbance affects the system, without sacrificing performance - unlike traditional e-modification. The neural adaptive method outperforms two types of PI controllers when tracking arbitrary set points, of organic substrate and dissolved oxygen, when appropriate feedforward terms are unknown. The method also outperforms a feedback linearizing controller using model parameter estimates when an observer is used to provide an estimate of unmeasured substrate concentration.

I. INTRODUCTION

Municipal sewage treatment plants commonly use biological waste water treatment systems. The treatment system ensures that the large inventory of microorganisms (activated sludge) consumes the pollutant inputs. Phosphorus, nitrogen, and ammonia can be removed by different kinds of bacteria (bio-P,nitrifying, and denitrifying) in a process involving a train of anoxic, anaerobic, and aerobic zones. On the other hand, the organic substrate is consumed only in the aerobic zone(s) by numerous species of oxygen-using bacteria. The control problem discussed in this paper involves regulating the (output) organic substrate by allowing the control system to decide on the (input) recycle rate and (input) dissolved oxygen (DO). In practice the recycle rate is regulated by changing the recycle flow, and the dissolved oxygen by controlling the aeration valves.

Maintaining the stability of a bioreactor poses a non-trivial control systems problem. Too much or too little substrate or too little DO can lead to die-off of the biomass. Thus, choosing an appropriate DO and substrate setpoint, and then maintaining the setpoint, define important objectives. Here appropriate setpoints are assumed to have been identified, whether manually or through computer algorithms, and address the problem of reaching and maintaining the set points. Changing operating conditions can make this a challenging task - as during a rain storm or when so-called toxic slugs move through the system.

For low-level feedback control systems, the industry currently uses proportional-integral-derivative (PID) or PI control [1]. PI and PID performance will degrade significantly away from the setpoint since the system dynamics contain significant nonlinearities - although auto-tuning can mitigate the difficulties [2], [3]. There are more advanced linear control techniques than PID, which can provide improved performance and stability (especially disturbances, and uncertainties). Some have proposed multivariable robust linear controls [4], [5], [6] and others model-predictive linear control [7], [8]. One different approach uses plant input-output directly in developing a model and control system, skipping the modelling step altogether [9]. Nonlinear adaptive control appears especially suitable for this type of plant [10], [11], [12], since large nonlinearities occur that are difficult to model and changes in plant configuration/parameters are common.

Achieving both adaptation and robustness to disturbances, while maintaining excellent performance, provides the goal in this research paper. The proposed control uses an adaptive CMAC neural network approach, or approximate adaptive control. Previous methods in the literature adapt to only certain parameters in the simplified model. Using neural networks allows us to estimate much more general nonlinearities, even those that are difficult to model. However, previous robust approximate adaptive approaches in the literature require a trade-off between performance and stability. The proposed method avoids this trade-off.

II. BACKGROUND

A model for a bioreactor, simple enough to allow control system design yet capturing the essential dynamics, uses four states

- biomass concentration \( X(t) \)
- substrate concentration \( S(t) \)
- dissolved oxygen concentration \( C(t) \)
- recycled biomass \( X_r(t) \)

and measured output typically consists of only dissolved oxygen \( C(t) \). The concentrations inside the bioreactor change due to the effects of cell mass production (with rate \( \mu(S,C,t) \)) and the inputs:

- recycle rate \( r(t) \)
- air flow rate \( W(t) \)

Four simple nonlinear dynamic equations, as written in [11], model these relationships as

\[
\begin{align*}
\dot{X} &= \mu(S,C,t)X - D(1+r)X + rDX_r, \\
\dot{S} &= -\mu(S,C,t)X/Y - D(1+r)S + DS_{in}, \\
\dot{C} &= -\mu(S,C,t)K_0X/Y - D(1+r)C + DC_{in} + (C_s - C)\alpha W, \\
\dot{X}_r &= D(1+r)X - D(b+r)X_r,
\end{align*}
\]

with (positive) parameters

- dilution rate \( D \)
(waste flow)/(influent flow) $b$
(oxygen transfer rate)/(air flow rate) $\alpha$
and measured quantities in the input flow
input substrate concentration $S_{in}$
input dissolved oxygen concentration $C_{in}$
and parameters include: cell mass $Y$, maximum dissolved oxygen $C_s$, and positive constant $K_0$. Using a Monod Law for $\mu$ gives a simple model

$$\mu = \mu_{max} \frac{S}{K_s + S} + \frac{C}{K_c + C}$$

(5)

where $\mu_{max}$, $K_s$ and $K_c$ are positive constants.

The objective of the control is to follow a setpoint $\zeta^* = [S^* \ C^*]^T$ and the dynamics for the relevant states are represented

$$\dot{\zeta} = f(\zeta, x) + p_0 - D\zeta + g(\zeta)u$$

(6)

where

$$f(\zeta, x) = \begin{bmatrix} S \\ C \end{bmatrix} \quad x = \begin{bmatrix} X \\ X_r \end{bmatrix} \quad u = \begin{bmatrix} r \\ W \end{bmatrix} \quad p_0 = \begin{bmatrix} DS_{in} \\ DC_{in} \end{bmatrix}$$

and

$$g(\zeta) = \begin{bmatrix} -DS \\ -DC (C_{max} - C) \alpha \end{bmatrix}$$

Given perfect knowledge of the system a stabilizing control to force $\dot{\zeta} \to 0$ is given by

$$u = g^{-1}(-f - p_0 + \zeta^* + D\zeta - Kz)$$

(7)

where $K$ is a positive-definite matrix of gains and $z = \zeta - \zeta^*$

(8)

The stability of the zero dynamics $\dot{z}$ must be analyzed to ensure stability. The zero dynamics can be written

$$\begin{bmatrix} \dot{X} \\ \dot{X}_r \end{bmatrix} = \begin{bmatrix} \mu - D(1 + r) & Dr \\ D(1 + r) & -D(b + r) \end{bmatrix} \begin{bmatrix} X \\ X_r \end{bmatrix}$$

(9)

which has negative eigenvalues as long as $D_{min}(1 + r) > \mu$, which is how the value of $D_{min}$ must be established.

Our objective is to design a stabilizing control in spite of unknown nonlinear functions and parameters in the nonlinear model. Specifically, it is assumed that $f(\zeta, x)$ is difficult to model and thus remains unknown; although parameters $r$, $C_{max}$, $\alpha$ are known. The control algorithm will not require knowledge of $S_{in}$ or $C_{in}$. An adaptive design using CMAC neural networks will accomplish the objective.

III. CMAC ADAPTIVE CONTROL

The control is designed assuming knowledge of a nominal setpoint $\zeta_n$, $u_n$. The objective is to design a control to track any desired setpoint $\zeta^*$ without knowledge of the corresponding $u^*$.

The Cerebellar Model Arithmetic Computer is a type of neural network that consists of layers of offset look-up tables. The $n$ inputs activate one $n$-dimensional hypercube cell on each layer. Each cell has an associated weight. In a binary CMAC the activated weights are simply summed to get the output. In a more advanced scheme, each cell has a basis function associated with it, and a weighted sum of the activated basis functions provides the output. For the wastewater treatment system, consider the proposed control using CMAC

$$\Delta u = g^{-1}(-\Gamma(\zeta, x)\dot{w} - Kz)$$

(10)

where $\Delta u$ is the difference from the nominal set point control $u_n$, the activated CMAC weights are in column vector $w$ and

$$\Gamma = \begin{bmatrix} \gamma \\ 0 \\ 0 \end{bmatrix}$$

where each $\gamma$ is a row vector of activated CMAC basis functions. The ideal (unknown) weights in the CMAC will be denoted $w$ and the weight error is $\dot{w} = w - \bar{w}$. Consider a direct adaptive control Lyapunov function

$$V = \frac{1}{2}z^Tz + \frac{1}{2\beta}\dot{w}^T\dot{w}$$

(11)

The time derivative is

$$\dot{V} = z^T(f + p_0 - D\zeta + g(\Delta u + u_n) - \dot{\zeta}^*) + \frac{\dot{w}^T\dot{w}}{\beta}$$

(12)

Substituting (10) gives

$$\dot{V} = z^T(f + p_0 - D\zeta - \Gamma\dot{w} - Kz + gu_n - \dot{\zeta}^*) + \frac{\dot{w}^T\dot{w}}{\beta}$$

(13)

The CMAC neural network can uniformly approximate a nonlinear function. Thus, one can approximate all the terms as

$$f + p_0 - D\zeta + gu_n - \dot{\zeta}^* = \Gamma(\zeta, x)\dot{w} + d(\zeta, x),$$

(14)

with $\|d(\zeta, x)\| \leq d_{max}$; positive constant $d_{max}$ bounds the approximation errors in $d$. Then

$$\dot{V} = z^T(\Gamma\dot{w} - \Gamma\dot{w} + d - Kz) - \frac{\dot{w}^T\dot{w}}{\beta}$$

(15)

$$= z^T(d - Kz) + \frac{\dot{w}^T\dot{w}}{\beta}$$

(16)

Consider choosing robust (e-modification) weight updates

$$\dot{w} = \beta(\Gamma^Tz - \nu\|z\|)$$

(17)

The time derivative becomes

$$\dot{V} = -z^TKz + z^TD + \|z\|\nu\dot{w}^T\dot{w}$$

(18)

$$= -z^TKz + z^TD + \|z\|\nu\dot{w}^T\dot{w} - \|z\|\nu\|\dot{w}\|^2$$

(19)

$$< \|z\|(-\lambda_{min}(K))\|z\| + d_{max} + \nu\|\dot{w}\|\|z\| - \nu\|\dot{w}\|^2$$

(20)

Setting (20) equal to zero on the $(\|z\|, \|\dot{w}\|)$ plane defines a compact set $B$, and since $\dot{V} < 0$ outside $B$ according to standard Lyapunov arguments all signals will be uniformly ultimately bounded.

Another method for choosing robust weight updates is projection (instead of e-modification), and the simplest form
is an element-by-element bounding assuming one knows a maximum bound on the magnitude of the weights $w_{\text{max}}$.

$$\dot{\hat{w}}_i = \text{Proj}_z(\beta \Gamma z) = \begin{cases} 0 & \text{if } \hat{w} \geq w_{\text{max}} \text{ and } z > 0 \\ \beta \Gamma z & \text{otherwise} \end{cases}$$

Note in the first two cases above $z^T \Gamma \hat{w}_i < 0$. Defining

$$V = \frac{1}{2} z^T z + \sum_{i=1}^{m} V_{w,i}$$

the time derivative becomes

$$\dot{V}_{w,i} = \begin{cases} z^T \Gamma \hat{w}_i & \text{if } \hat{w} \geq w_{\text{max}} \text{ and } z > 0 \\ z^T \Gamma \hat{w}_i & \text{if } \hat{w} \leq -w_{\text{max}} \text{ and } z < 0 \\ 0 & \text{otherwise} \end{cases}$$

Since

$$\frac{d}{dt} \left( \frac{1}{2} z^T z \right) \leq z^T K z + dz$$

each $\dot{V} < 0$, for $i = 1, \ldots, L$, when $\|z\| < \delta_z$ where

$$\delta_z = d / \lambda_{\text{min}} K$$

and a uniform ultimate bound in error, $z_b$, is found by solving the following

$$V(\delta_z, w_{\text{max}}) = V(z_b, 0)$$

for $z_b$.

IV. PROPOSED ROBUST MODIFICATION

The proposed algorithm takes advantage of the nature of the CMAC local basis functions. It can measure the average error over the local cell associated with weight $\hat{w}_i$ and the average error as measured over the $i$th cell and then next cell (on the same layer) activated in time. It can measure the changes in weight and average error between two activations of the cells (due to periodic motion), $\Delta \hat{w}_i$ and $\Delta \hat{z}$. Then it can estimate

$$\frac{\partial \hat{z}}{\partial \hat{w}_i} \approx \frac{1}{N} \sum_{j=1}^{L} \frac{\Delta \hat{z}_j}{\Delta \hat{w}_i}$$

where the average is taken over the $L$ activated cells (one on each layer). Then the weight update can be applied

$$\hat{w} = \begin{cases} \text{Proj}_x(\beta \Gamma^T z) & \text{if } \frac{\partial \hat{z}}{\partial \hat{w}_i} > \rho \\ 0 & \text{otherwise} \end{cases}$$

where $\rho$ is a positive constant. Thus, weight updates will stop when whether the weights reach their pre-determined maximum magnitudes or the effect of the weights on the average error becomes small enough. A uniform ultimate bound when they reach maximum is established through the projection algorithm. When $p$ weights are stopped short of the maximum magnitude (and $m - p$ are either stopped at the maximum amplitude or continue to update) the time derivative can be bounded

$$\dot{V} \leq -z^T K z + dz + \sum_{j=1}^{p} z^T \Gamma_j \hat{w}_j$$

$$\leq \|z\| (\lambda_{\text{min}}(K) \|z\| + d + \|\Gamma\| \|\hat{w}_p\|)$$

where $\hat{w}_p$ is a vector containing all (constant) weight errors that have been stopped short of the maximum magnitude. Assuming the basis functions have been normalized such that $\|\Gamma\| = 1$ then $\dot{V} < 0$ when

$$\|z\| > \frac{d + \|\hat{w}_p\|}{\lambda_{\text{min}}(K)}$$

and the system is uniformly ultimately bounded as before. Although the bounds are theoretically larger, improved performance is expected since the weights should be very close to the ideal weights when the weights stop updating.

A. Details

The following provides an overview of the algorithm for estimating $\frac{\partial \hat{z}}{\partial \hat{w}_i}$. More details are provided in [13].

The CMAC consists of $L$ layers of $N$-dimensional look-up tables. One hypercube cell is activated on each layer by the $N$-dimensional input. The layers are offset from one another so that the $L$ activated cells all have slightly different domains. For clarity, the following discussion and nomenclature is limited to a single layer, the $j$th layer, of the CMAC; keep in mind the same procedures occur for each layer in turn for $j = 1 \ldots L$. The activated cell on the $j$th layer is referred to as the current cell, which contains the current weight. The cell indexing scheme assumes that

Fig. 1. Showing cells in a two-input CMAC. Indexing activation/deactivation times at the end of one cycle: $n$ refers to total number of cell activations on the $m$th cycle, where $m$ is the number of activations of the current cell. For Case 2 trajectory $t_{n-2,m} = t_{n,m-1}$.
periodic cycles are occurring, referring to the $m$th cycle. The $n$th cell to be activated during the $m$th cycle contains the current weight denoted $\hat{w}_{n,m}$. When the cell on this layer is deactivated (when the input vector leaves the current cell) the cell becomes the last cell on this layer, and its weight the last weight $\hat{w}_{n-1,m}$. The 2nd-last cell has weight $\hat{w}_{n-2,m}$.

This is illustrated in Figure 1, where the two possible cases of trajectories are shown:

**Case 1** a trajectory that moves through (at least) three different cells continuously,

**Case 2** a trajectory that is oscillating between two cells.

Note that in Case 2 the 2nd-last cell and the current cell are actually the same cell, since the cycle is oscillating between two cells. The moment the current cell is activated is $t_{n,m}$. The last cell was activated at $t_{n-1,m}$, and 2nd-last cell at $t_{n-2,m}$. Note the indexing $n$ and $m$ are used to provide a clear mathematical description, but the computer algorithm merely needs to keep track of the last two cells activated in time.

### B. On-line weight update

The weight is divided into a permanent memory weight component $p_{n,m}$ and an on-line weight $\hat{w}_{n,m}$. The on-line weight has zero initial condition at time $t_{n,m}$ so that

$$\hat{w}_{n,m}(t) = \int_{t_{n,m}}^{t} \dot{\hat{w}}_{n,m}(t) \, dt \tag{30}$$

where $\dot{\hat{w}}_{n,m}(t)$ is a Lypunov-stable parameter update law like (17). The algorithm will decide whether to keep the on-line weight by adding it to the permanent memory, or discard it altogether. Thus, the CMAC output in the control consists of a weighted sum of on-line weights $\hat{w}$ and permanent weights $p$, and thus the control becomes

$$u = -\nabla \hat{z}(\hat{w} + \hat{w}) - Gz \tag{31}$$

### C. Recording average error

To evaluate the error associated with a weight, look at the average error in the weight’s cell domain as well as the average error in the next cell activated in time. In this way more information for Case 1 trajectories is obtained, and full information for Case 2 trajectories. The algorithm must wait for the next cell to be deactivated before making decisions i.e. the decision is about the 2nd last cell and whether to add $\hat{w}_{n-2,m}$ to $p_{n-2,m}$ (keep the on-line weight update) at time $t_{n,m}$ where

$$\hat{w}_{n-2,m} = \beta \int_{t_{n-2,m}}^{t_{n-1,m}} \hat{w}_{n-2,m}(t) \, dt \tag{32}$$

(Note $\hat{w}_{n-2,m} = \hat{w}_{n,m-1}$ for a Case 2 trajectory).

For the 2nd last cell the total error over two sequential cell domain activations is

$$E_{n-2,m} = \frac{1}{t_{n,m} - t_{n-2,m}} \int_{t_{n-2,m}}^{t_{n,m}} z \, dt \tag{33}$$

Defining $T_{n-2,m} = t_{n,m} - t_{n-2,m}$ Table I defines all the nomenclature, with continuous time flowing left to right and subsequent activations of the same cells (i.e. cycles) going from top to bottom: For Case 2 trajectories cell$_n$, cell$_{n-2}$ and $E_{n-2,m} = E_{n-2,m-1}$.

### D. Weighted voting

The change in error is that between cycles

$$\Delta E_{n-2} = E_{n-2,m} - E_{n-2,m-1} \quad (34)$$

Each layer casts a vote as to whether it thinks the on-line weight update in the 2nd last cell $\hat{w}_{n-2,m}$ in that layer has reduced the error in that layer (over two cell activations) by evaluating the derivative $\partial E / \partial \hat{w}$ with $\partial E / \partial \hat{w}$. Specifically, choose three criteria as follows 1) a positive on-line weight should reduce the average error 2) the average error should remain the same sign between the two cycles and 3) the error should have changed by a significant amount. In addition, rather than just voting keep or toss in a binary manner, the vote is weighted using the magnitude of the previous on-line weight update:

$$\text{vote}_{n-2} = \begin{cases} \text{vote}_{n-2} = \text{vote}_{n-2} + \text{vote}_{n-2} & \text{if } \Delta E_{n-2} < 0 \\ \text{vote}_{n-2} & \text{if } \Delta E_{n-2} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{35}$$

where positive is a vote to keep and negative a vote to toss. To determine if $\hat{w}_{n-2,m}$ will be applied as a weight update to the permanent memory for cell$_{n-2}$, all the votes stored in memory from the last and 2nd-last cells on all layers are counted. A positive result means keep $\hat{w}_{n-2,m}$ and negative means toss:

$$\text{voting} = \sum_{j=1}^{m} \text{vote}_{n-1,j} + \text{vote}_{n-2,j} \tag{36}$$

where vote$_{n-1,j}$ refers to the vote of the last cell on the $j$th layer, and vote$_{n-2,j}$ the 2nd-last cell’s vote on this layer. By including both the last and 2nd last cell in the voting Case 2 trajectories then have information on the complete cycle when voting (and Case 1 trajectories have additional useful information).

The result is the permanent memory weight on the $j$th layer, $\hat{p}_{n-2,j}$, is updated at time $t_{n,m}$ according to

$$\Delta \hat{p}_{n-2,j} = \begin{cases} \hat{w}_{n-2,j} & \text{if voting} > 0 \\ \text{vote}(\hat{b}_{n-2,j} - \hat{p}_{n-2,j}) & \text{otherwise} \end{cases} \tag{37}$$

where $\nu$ is a small positive constant.

### V. Results

#### A. Simulation, Observer, and Controller Parameters

The simulations assume only dissolved oxygen $C$ can be measured and utilize a standard linear observer to estimate the other states. The parameters used in the simulation,
but unknown to the controller/observer (Table II first two columns) are those from [11]. The linear observer is designed using parameter estimates in Table II (third column) and then linearizing about a (known) nominal set point $\zeta_n = [S_n \ C_n]^T = [30 \ 3]^T$, $u_n = [r_n \ C_n]^T = [0.54 \ 32]^T$.

The observer gains are $K_{\text{observer}} = [10 \ 10 \ 10 \ 10]$. The neural network is provided with information of the state estimate $\hat{S}$ instead of the true (unmeasured) $S$, but provided with the real (measured) value of $C$.

Parameters used in the proposed control are $K = \text{diag}(0.1, 0.85)$, $\beta = 0.1$. The initial conditions for all simulations are $X_0 = 215, S_0 = 55, C_0 = 6, X_{r,0} = 400$.

<table>
<thead>
<tr>
<th>Known Parameters</th>
<th>Unknown parameters</th>
<th>Parameter Estimates</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D = 0.05 \text{ per h}$</td>
<td>$Y = 0.65$</td>
<td>$Y = 0.5$</td>
</tr>
<tr>
<td>$b = 0.2$</td>
<td>$K_0 = 0.5$</td>
<td>$K_0 = 0.4$</td>
</tr>
<tr>
<td>$\alpha = 0.018m^{-3}$</td>
<td>$K_S = 100mg/L$</td>
<td>$K_S = 80mg/L$</td>
</tr>
<tr>
<td>$C_{\text{max}} = 10 mg/L$</td>
<td>$K_c = 2mg/L$</td>
<td>$K_c = 2.4mg/L$</td>
</tr>
<tr>
<td>$C_{\text{in}} = 0.5mg/L$</td>
<td>$\mu_{\text{max}} = 0.15 mg/L$</td>
<td>$\mu_{\text{max}} = 0.12 mg/L$</td>
</tr>
<tr>
<td>$S_{\text{in}} = 200mg/L$</td>
<td>$S_{\text{in}} = 160mg/L$</td>
<td></td>
</tr>
</tbody>
</table>

The initial conditions for all simulations are $X_0 = 215, S_0 = 55, C_0 = 6, X_{r,0} = 400$.

B. Testing robust parameter updates

The first scenario is designed to justify the use of the proposed neural network update method, as opposed to using $e$-modification, by testing the ability to maintain the nominal setpoint $\zeta_n$ when there is a constant sinusoidal disturbance in $C$, namely $C_{\text{disturbance}} = 0.1 \sin[(2\pi)/(T_d)]$ where $T_d$ is 15 minutes. The $e$-modification method (17) with $\nu = 0.001$ results in (repeated) bursting occurring near the 2.5 hour mark (Figure 2) while the proposed method can maintain the set point without bursting. The mechanism for the bursting by looking at the magnitude of the weight vector (Figure 3), as it keeps drifting when using $e$-modification but not when using the proposed robust weight update method.

C. Comparing to other controls

The second simulation uses a scenario which switches between two set points every 30 minutes, between $S^* = 25, C^* = 2$ and $S^* = 35, C^* = 4$. It compares the proposed neural network method to two types of PI control (with feedforward) and the feedback linearizing nonlinear control in (7). The first PI control uses a static $r = r_n = 0.5$ and
measured $C$

$$W = W_n + 0.85(C^* - C) + 0.01 \int (C^* - C) dt \quad \text{PI 1}$$

The second PI control that utilizes the estimated states $\hat{S}, \hat{C}$ from the observer

$$r = r_n - 0.1(S^* - \hat{S}) - 0.01 \int (S^* - \hat{S})$$

$$W = W_n + 0.85(C^* - \hat{C}) + 0.01 \int (C^* - \hat{C}) dt \quad \text{PI 2}$$

The feedback linearizing control uses the same parameter estimates used in designing the observer.

The first PI control cannot track the setpoint for $S$ (Figure 4 dotted line), the second PI control has relatively slow transient response (Figure 4 dash-dot line). The feedback linearizing controller results in oscillations due to coupled interactions with the observer dynamics (Figure 4 dashed line). The proposed method outperforms these three, and steady state errors in $S$ are caused entirely by the observer error (Figure 5). Note the error in $C$ can be driven exactly to zero by using the actual error in $C$ in the neural network training signal, whereas the other control methods are affected by observer error in $C$ (Figure 4). Note that the interaction with observer dynamics is avoided simply by picking $K$ and $\beta$ small enough that the controller dynamics are slow relative to the observer dynamics (a technique which is not possible with the feedback linearizing control). The control signals remain smooth and do not saturate (Figure 6).

VI. CONCLUSIONS

A robust neural adaptive control method is proposed for controlling an activated sludge bioreactor for use in wastewater treatment. The proposed method self-evaluates the effect of weight updates upon the average error in a local domain, and ceases updating the weights when updates are deemed to have no beneficial, or insignificant, effect on the error. It does this by taking advantage of the local nature of the basis functions in a CMAC i.e. the average error during a cell activation is easily measured. The method outperforms the robust method of $e$-modification by avoiding bursting in the state error. The method also outperforms PI controls applied to the plant.

REFERENCES