2017 IEEE Conference on Control Technology and Applications (CCTA)
August 27-30, 2017. Kohala Coast, Hawai'i, USA

# CMAC Control of a Quadrotor Helicopter using a Stable Robust Weight-Smoothing Algorithm

C.J.B. Macnab
Dept. of Electrical and Computer Engineering, University of Calgary, Calgary, Alberta, Canada
*cmacnab@ucalgary.ca*

***Abstract*—The Cerebellar Model Articulation Controller (CMAC) is a type of neural network particularly suited to real-time control applications due to fast adaptation and the ability to handle many inputs. However, the CMAC is well-known to exhibit weight (adaptive-parameter) drift when used in adaptive control, and overlearning when applied in static learning situations. A weight smoothing algorithm originally proposed for halting the overlearning during static training has recently been modified and proposed for use in adaptive control. However, the method as proposed currently suffers from impractical computational complexity for more than one or two inputs, and does not have guarantees of stability. This paper proposes both a way to ensure computational efficiency for real-time operation and a Lyapunov-derived method of achieving uniformly ultimately bounded signals. Simulations with a quadrotor helicopter show the method significantly outperforms other robust direct adaptive control methods (deadzone and e-modification) as well as a tuned PID control.**

## I. Introduction

The Cerebellar Model Articulation Controller (CMAC) [1] has always held the potential to become the standard nonlinear approximator in nonlinear adaptive control schemes, since it adapts much faster than multilayer perceptrons (MLPs) and can handle many more inputs than radial basis function networks (RBFNs). However, the CMAC is prone to *weight drift*, which is also called *adaptive-parameter drift* or *overlearning*. Continued weight drift will result in increasing control chatter until *bursting* occurs, where the state error suddenly increases after a period of convergence at small errors. Typical robust weight updates normally used to prevent drift (parameter projection, deadzone, leakage, and $e$-modification) typically work well for MLPs or RBFNs, but sacrifice performance in the case of CMAC [2]. The reason weight drift is so severe in the CMAC is the local nature of the cells; a small oscillation about the origin (zero error) can oscillate between two different cells and two different basis functions. If the weight update is based on state error as in direct adaptive control, then the weight in one cell will drift to large positive values and the weight on the other side of the origin will drift to large negative values, creating an increasingly chattering control signal. Contrast this to MLPs and RBFNs where a small oscillation about the origin occurs entirely within each basis function, and if the basis functions have a small-enough slope the weights will not tend to grow in magnitude i.e. weight drift can easily be prevented simply by choosing basis functions wide enough (with slope small enough) in the case of MLPs and RBFNs.

In the CMAC control literature deadzone [3] and $e$-modification [4] have been explicitly proposed, however the issue of poor performance was not dealt with. Others have proposed variable adaptation rates to solve the problem, without providing stability proofs [5] and [6]. This author has proposed using a set of alternate weights [7] and an introspective method [8], both applied to quadrotor control. This paper explores the method of *weight smoothing*, which was originally designed to prevent overlearning during CMAC static training [9],[10] but has been recently proposed for on-line adaptation in [11]. Weight smoothing is an attractive algorithm: it makes intuitive sense and it is simple. The two drawbacks from the method in [11] were 1) great computational complexity even in the case of one-input CMACs with real-time implementation very unlikely for multi-input CMACs 2) no stability proof. This paper first proposes a method for overcoming the computational complexity problem (*nonrobust weight smoothing*) and then develops a scheme to ensure uniformly ultimately bounded (UUB) signals using Lyapunov methods (*robust weight smoothing*). The proposed method is tested with a quadrotor helicopter simulation. The simulations first demonstrate how both $e$-modification and deadzone exhibit a severe trade-off between stability and performance when using CMAC. Then simulations demonstrate the excellent performance, disturbance-rejection, and control-efficient properties of the weight smoothing algorithms (both nonrobust and robust versions) compared to a tuned PID control.

## II. Background

### A. CMAC

When used as a nonlinear approximator, the CMAC output provides an estimate $\hat{f}(\mathbf{x}_{\text{in}})$ of nonlinear function $f(\mathbf{x}_{\text{in}})$, with $\mathbf{x}_{\text{in}} \in \mathcal{R}^n$. Like RBFNs, the CMAC output is a weighted sum of basis functions

$$\hat{f}(\mathbf{x}_{\text{in}}) = \mathbf{\Gamma}(\mathbf{x}_{\text{in}})\mathbf{w}, \tag{1}$$

where $\mathbf{\Gamma} \in \mathcal{R}^{1\times N}$ is a row vector of basis functions and $\mathbf{w} \in \mathcal{R}^N$ is a column vector of weights. To define basis function domains, the CMAC uses $m$ stacked arrays which are offset from each other. Basis functions are typically rectangular (binary), triangular, or splines with a value and derivative of zero at the cell boundaries (Fig 1). With each input divided into $q$ discrete quantizations, each array contains $q^n$ hypercube *cells* and the total number of cells

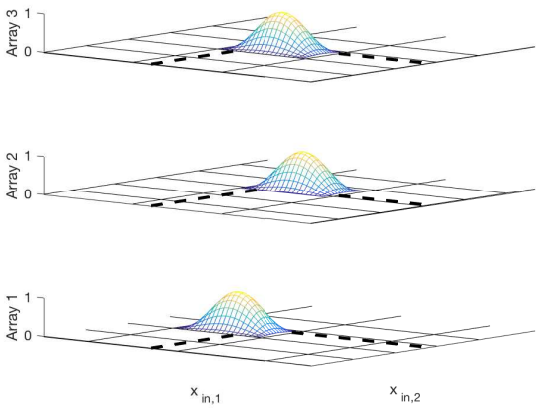


Fig. 1: An example spline CMAC with $n = 2$ inputs, $m = 3$ offset stacked arrays, and $q = 4$ discretizations (only the activated basis functions indexed by the dashed-line inputs are illustrated).

is $N = mq^n$. Computational efficiency stems from only having to calculate the basis functions and weights for the $m$ *activated* cells (since the input indexes one cell per array) rather than for all $N$ as is required in an RBFN. Instead of allocating $N$ memory locations, the CMAC uses a hash-coding scheme to store only (previously) activated weights [12].

Like an RBFN or MLP, the CMAC is capable of uniformly approximating nonlinear function $f(\mathbf{x}_{\text{in}})$ in a region $\mathcal{D} \subset \mathcal{R}^n$; that is

$$f(\mathbf{x}_{\text{in}}) = \mathbf{\Gamma}(\mathbf{x}_{\text{in}})\mathbf{w} + d(\mathbf{x}_{\text{in}}) \quad \forall \mathbf{x}_{\text{in}} \in D, \tag{2}$$

where $|\mathbf{x}_{\text{in}}| < d_{\max} \ \ \forall \ \mathbf{x}_{\text{in}} \in D$ with $d_{\max}$ a positive constant.

### B. Direct Neural-Adaptive Control

Consider a second-order nonlinear system in state-space form with $x_1 = \theta$ and $x_2 = \dot{\theta}$

$$\dot{x}_1 = x_2, \tag{3}$$
$$\dot{x}_2 = f(x_1, x_2) + gu, \tag{4}$$

where $g$ is a positive constant and $u$ a control input. Given desired trajectory $\theta_d, \dot{\theta}_d, \ddot{\theta}_d$ one can define the auxiliary error

$$z = \Lambda(x_1 - \theta_d) + (x_2 - \dot{\theta}_d), \tag{5}$$
$$= \Lambda e_1 + e_2, \tag{6}$$

where $\Lambda$ is a positive constant (note when $z \equiv 0$ this is a stable linear system). Choosing adaptive control Lyapunov function

$$V = \frac{1}{2g}z^2 + \frac{1}{2\beta}\tilde{\mathbf{w}}^T\tilde{\mathbf{w}}, \tag{7}$$

where the weight error has been defined $\tilde{\mathbf{w}} = \mathbf{w} - \hat{\mathbf{w}}$ with $\mathbf{w}$ denoting the ideal set of weights and $\hat{\mathbf{w}}$ denoting the weight estimates (actual weights in the CMAC). The time derivative is

$$\dot{V} = z[(\Lambda e_2 + f(x_1, x_2) - \ddot{\theta}_d)/g + u] - \tilde{\mathbf{w}}^T\dot{\hat{\mathbf{w}}}/\beta, \tag{8}$$
$$= z(\mathbf{\Gamma}(\mathbf{x}_{\text{in}})\mathbf{w} + d(\mathbf{x}_{\text{in}}) + u) - \tilde{\mathbf{w}}^T\dot{\hat{\mathbf{w}}}/\beta, \tag{9}$$
$$= z(\mathbf{\Gamma}(\mathbf{x}_{\text{in}})\hat{\mathbf{w}} + d(\mathbf{x}_{\text{in}}) + u) + \tilde{\mathbf{w}}^T(\mathbf{\Gamma}(\mathbf{x}_{\text{in}})z - \dot{\hat{\mathbf{w}}}/\beta), \tag{10}$$

where $\mathbf{x}_{\text{in}} = [x_1 \ x_2 \ \dot{\theta}_d \ \ddot{\theta}_d]^T$. Choosing control and robust $e$-modification [13] weight updates

$$u = -\mathbf{\Gamma}(\mathbf{x}_{\text{in}})\hat{\mathbf{w}} - Gz, \tag{11}$$
$$\dot{\hat{\mathbf{w}}} = \beta(\mathbf{\Gamma}(\mathbf{x}_{\text{in}})z - \nu|z|\hat{\mathbf{w}}), \tag{12}$$

where $G$ is a positive control gain and $\nu$ is a positive constant, results in

$$\dot{V} = d_{\max}z - Gz^2 - \nu|z|\tilde{\mathbf{w}}^T\tilde{\mathbf{w}} + \nu|z|\tilde{\mathbf{w}}^T\mathbf{w}, \tag{13}$$
$$\leq |z|(d_{\max} - G|z| - \nu\|\tilde{\mathbf{w}}\|^2 + \nu\|\tilde{\mathbf{w}}\|\|\mathbf{w}\|). \tag{14}$$

Since $\dot{V} < 0$ outside of a compact set on the $(|z|, \|\tilde{\mathbf{w}}\|)$ plane that includes the origin, all signals are UUB with an ultimate bound given by the smallest Lyapunov surface that encloses the compact set. An alternative robust weight update is deadzone [14]

$$\dot{\hat{\mathbf{w}}} = \begin{cases} \beta\mathbf{\Gamma}(\mathbf{x}_{\text{in}})z & \text{if } |z| > \zeta, \\ 0 & \text{otherwise,} \end{cases} \tag{15}$$

where positive constant $\zeta$ must be chosen such that $\zeta \geq d_{\max}/G$. In this case signals are also UUB: the Lyapunov surface that includes point $(\zeta, \|\tilde{\mathbf{w}}(0)\|)$ is an ultimate bound.

### C. Weight Smoothing

In order to prevent overlearning during CMAC static training, Kraft and Pallotta [9] proposed a weight smoothing technique which penalizes the difference between adjacent weights in a CMAC layer: for a one-input CMAC with $q$ quantizations an appropriate cost-functional for the $i$th array is

$$J_{w,i} = (w_{1,i} - w_{2,i})^2 + (w_{2,i} - w_{3,i})^2 + \ldots + (w_{q-1,i} - w_{q,i})^2, \tag{16}$$
$$= \mathbf{w}_i^T\mathbf{Q}\mathbf{w}_i, \tag{17}$$

where $\mathbf{w}_i \in \mathcal{R}^q$ is a vector of weights for the $i$th array and $\mathbf{Q} \in \mathcal{R}^{q \times q}$ is

$$\mathbf{Q} = \begin{bmatrix} 1 & -1 & 0 & 0 & \ldots & 0 \\ -1 & 2 & -1 & 0 & \ldots & 0 \\ 0 & -1 & 2 & -1 & \ldots & 0 \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ 0 & \ldots & \ldots & -1 & 2 & -1 \\ 0 & \ldots & \ldots & 0 & -1 & 1 \end{bmatrix}. \tag{18}$$

The error for static training of nonlinearity $f$ at the $k$th input data point is

$$e_k = f(\mathbf{x}_k) - \mathbf{\Gamma}(\mathbf{x}_k)\mathbf{w}, \tag{19}$$

and a cost-functional for batch training on $\mathcal{N}$ data points with weight smoothing is

$$J = \mathbf{e}^T \mathbf{R} \mathbf{e} + \sum_{i=1}^{m} \mathbf{w}_i^T Q \mathbf{w}_i, \tag{20}$$

where $\mathbf{R} \in \mathcal{R}^{\mathcal{N}\times\mathcal{N}}$ is constant and positive definite. Using this cost-functional Kraft and Pallotta derived a batch-training weight update rule for both one-dimensional [9] and two-dimensional CMACs [10]. Their results confirmed that overlearning was prevented.

Yang and Han [11] used a similar cost-functional, but for a single data point

$$J = \frac{1}{2} e^2 + \frac{1}{2} K \sum_{i=1}^{m} \mathbf{w}_i^T \mathbf{Q} \mathbf{w}_i, \tag{21}$$

where $K$ is a positive constant, and applying gradient descent resulted in weight updates for the $i$th CMAC array

$$\Delta \mathbf{w}_i = \eta(\mathbf{\Gamma}_i^T e - K \mathbf{Q} \mathbf{w}_i), \tag{22}$$

where $\eta$ is a positive constant. Yang and Han [11] then went further and applied their gradient descent algorithm to the case of direct adaptive control

$$\Delta \mathbf{w}_i = \eta(\mathbf{\Gamma}_i^T (y_{\text{ref}} - y) - K \mathbf{Q} \mathbf{w}_i), \tag{23}$$

where $y_{\text{ref}}$ is the desired output (reference signal) and $y$ is the system output.

Looking at (23) it is difficult to see how such an algorithm can be applied beyond the one or two-input case due to the computational complexity; in the authors' own words "... the weight updating method of the CMAC updates all weights each sampling time. This means that the improved CMAC is no longer a local neural network. The matrix operation is very time consuming and influences the real-time performance of control" [15]. Also, applying a gradient descent rule such as (23) in direct adaptive control is not necessarily stable; normally a Lyapunov-based derivation of a weight update rule is used to ensure stability.

## III. Proposed Method

### A. Eliminating Computational Complexity - Nonrobust Weight Smoothing

In order to make the weight smoothing algorithm computationally feasible, this paper proposes minimizing the difference between activated weights. Since there are only $m$ activated weights there are only $m$ extra calculations. Compare this to the previously proposed method where for $n$ inputs there are $2^n$ adjacent weights to a hypercube cell and thus $2^n$ extra calculations. (The results section of this paper uses a CMAC with $m = 100$ arrays and $n = 10$ inputs to control a quadrotor helicopter). In the proposed method an appropriate cost-functional for the $i$th array's activated weight would be

$$J_{w_a,i} = \sum_{j=1}^{m} (\hat{w}_{a,j} - \hat{w}_{a,i}), \tag{24}$$

$$= \hat{\mathbf{w}}_a^T \mathbf{Q}_{\text{new}} \hat{\mathbf{w}}_a, \tag{25}$$

where $w_{a,j}$ indicates the activated weight in array $j$, while $\hat{\mathbf{w}}_a$ is the vector of activated weights and

$$\mathbf{Q}_{\text{new}} = \begin{bmatrix} (m-1) & -1 & -1 & \dots & -1 \\ -1 & (m-1) & -1 & \dots & -1 \\ \vdots & & \ddots & \ddots & \vdots \\ -1 & \dots & -1 & (m-1) & -1 \\ -1 & \dots & -1 & -1 & (m-1) \end{bmatrix}.$$

Thus, based on a gradient-descent derivation, the weight update for the activated weights becomes

$$\dot{\hat{\mathbf{w}}}_a = \beta(\mathbf{\Gamma}_a^T z - K \mathbf{Q}_{\text{new}} \hat{\mathbf{w}}_a), \tag{26}$$

and for the general weight vector this can be expressed

$$\dot{\hat{\mathbf{w}}} = \beta(\mathbf{\Gamma}^T z - K \mathbf{S}(\mathbf{x}_{\text{in}}) \hat{\mathbf{w}}), \tag{27}$$

where $\mathbf{S}(\mathbf{x}_{\text{in}})$ has zeros on the rows associated with unactivated cells and terms from $\mathbf{Q}_{\text{new}}$ on rows associated with active cells.

### B. Ensuring Stability - Robust Weight Smoothing

Neither the previously proposed update (23) nor the new proposed update (27) were derived within a Lyapunov framework and thus there is no mathematical guarantee of stability. In order to ensure UUB signals, this paper proposes using two CMACs in parallel: a *robust CMAC* with weights $\hat{\mathbf{w}}$ and a *performance CMAC* with weights $\hat{\mathbf{p}}$. The control becomes

$$u = -\mathbf{\Gamma} \hat{\mathbf{p}} - \mathbf{\Gamma} \hat{\mathbf{w}} - Gz. \tag{28}$$

The robust CMAC applies $e$-modification (12) to update the weights $\hat{\mathbf{w}}$ using a large (conservative) value of $\nu$. The performance CMAC applies the new weight smoothing term (27) for weights $\hat{\mathbf{p}}$ until a critical time $T_c$ is reached, while ensuring the weights remain bounded in the meantime. The rational for this approach is that, by observation, the practical effect of the original (23) is to make the weight drift extremely slow and the practical effect of the proposed (27) is that it appears to halt weigh drift altogether. Thus, allowing (27) to occur for some finite amount of time appears to be entirely safe. Although, to ensure that safety (stability) we must impose a bound on the weight norm, $p_{\max}$, during that finite time. Thus by defining

$$\dot{\mathbf{P}} = \beta(\mathbf{\Gamma}^T z - K|z|\mathbf{S}(\mathbf{x}_{\text{in}}) \hat{\mathbf{p}}), \tag{29}$$

the proposed weight update for the performance CMAC's $j$th weight is expressed

$$\dot{\hat{p}}_j = \begin{cases} 0 & \text{if } \mathcal{A}_j, \\ \dot{P}_j & \text{otherwise,} \end{cases} \tag{30}$$

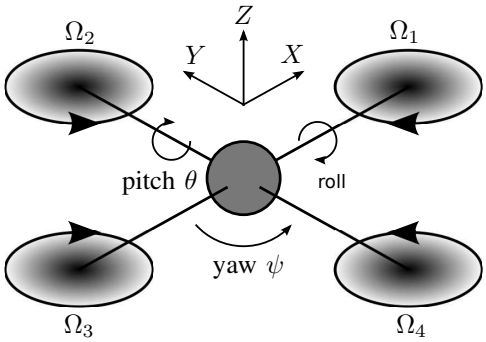

Fig. 2: Coordinates and assumption of rotor spin directions

where $\dot{P}_j$ is the $j$the element of $\dot{\mathbf{P}}$ and

$$\mathcal{A}_j = \begin{cases} 0 & \text{if } \|\hat{\mathbf{p}}\| = p_{\max} \text{ or } \sup_{j=1\ldots m}(T_{a,j}) > T_c, \\ 1 & \text{otherwise,} \end{cases} \quad (31)$$

where $T_{a,j}$ indicates the total time the $j$th activated cell has been activated.

To analyze stability consider the adaptive control Lyapunov function

$$V = \frac{1}{2b}z^2 + \frac{1}{\beta}\tilde{\mathbf{w}}^T\tilde{\mathbf{w}} + \frac{1}{\beta}\tilde{\mathbf{p}}^T\tilde{\mathbf{p}}. \quad (32)$$

Applying (28) for the control, (12) for the $\hat{\mathbf{w}}$ updates, and (30) for the $\hat{\mathbf{p}}$ updates, the time derivative becomes

$$\begin{aligned} \dot{V} =& -Gz^2 + z(\epsilon + d) + \nu|z|\tilde{\mathbf{w}}^T\mathbf{w} - \nu|z|\tilde{\mathbf{w}}^T\tilde{\mathbf{w}} \\ &+ K|z|\tilde{\mathbf{p}}^T Z(t)\hat{\mathbf{p}}, \quad (33) \\ \dot{V} <& |z|(-G|z| + zd_{\max} + \nu\|\tilde{\mathbf{w}}\|\|\mathbf{w}\| - \nu\|\tilde{\mathbf{w}}\|^2 \\ &+ (K+1)p_{\max}^2), \quad (34) \end{aligned}$$

where $Z(t)$ is a diagonal matrix, with values of 0 if the critical time has been reached and 1 otherwise. Thus, all signals are uniformly ultimately bounded since a bound on $\|\tilde{\mathbf{p}}\|$ is imposed and $\dot{V} < 0$ if either $|z|$ or $\|\tilde{\mathbf{w}}\|$ grows large enough.

Since bounds are imposed on the weights this algorithm may appear similar to traditional adaptive-parameter projection. However, unlike traditional adaptive-parameter projection one does not need knowledge of the model or extensive pretraining to identify maximum weight bounds i.e. one does not need to choose $p_{\max} \geq \|\mathbf{p}\|$. Also, unlike traditional adaptive-parameter projection one does not expect the weights to ever reach the maximum bounds when persistant sinusoidal disturbances affect the system i.e. $p_{\max}$ remains strictly a safety measure and we do not expect that the condition $\|\mathbf{p}\| \equiv p_{\max}$ will ever occur even when disturbances are present.

The purpose of the robust CMAC is twofold:

1) to ensure $p_{\max}$ can be chosen arbitrarily without knowledge of $\mathbf{p}$,
2) to ensure UUB signals even after the performance CMAC has stopped learning after each cell has reached total activation time $T_c$.

TABLE I: Quadrotor parameters

| Symbol | Definition | Value |
|---|---|---|
| $b$ | Thrust factor | $3.1 \times 10^{-5}$ (N s$^2$) |
| $d$ | Drag factor | $7.5 \times 10^{-7}$ (N m s$^2$) |
| $m$ | Mass | 0.468 (Kg) |
| $L$ | Distance | 0.232 (m) |
| $J_X$ | X Inertia | $6.2 \times 10^{-3}$ (Kg m$^2$) |
| $J_Y$ | Y Inertia | $6.2 \times 10^{-3}$ (Kg m$^2$) |
| $J_Z$ | Z Inertia | $1.1 \times 10^{-2}$ (Kg m$^2$) |
| $J_r$ | Rotor inertia | $6 \times 10^{-5}$ (Kg m$^2$) |

Note that the robust CMAC, on its own, would result in poor performance due to the large value of $\nu$, but when the two CMACs train in parallel the performance CMAC is fully capable of making up for the lack of accuracy in the robust CMAC (assuming $T_c$ is chosen large enough to allow convergence). Thus, the overall result is expected to be excellent performance.

## IV. Results

### A. Quadrotor Model

The quadrotor has rotor speeds $\Omega_i$ for $i = 1 \ldots 4$ (Figure 2 defines the directions) with each rotor inertia given by $J_r$. Using the inertia matrix $\text{diag}(J_X, J_Y, J_Z)$, body-frame vertical position $Z$, angular coordinates (roll $\phi$, pitch $\theta$, yaw $\psi$), and states $x_1 = Z$ , $x_2 = \phi$, $x_3 = \theta, x_4 = \psi, x_5 = \dot{Z}, x_6 = \dot{\phi}, x_6 = \dot{\theta}$ the equations of motion can be written as in [7] with an assumption of slowly-varying rotor speeds

$$\dot{x}_i = x_{i+4}, \quad \text{for } i = 1, 2, 3, 4, \quad (35)$$
$$\dot{x}_5 = -g\cos(x_2)\cos(x_3) + b_1 u_1, \quad (36)$$
$$\dot{x}_6 = x_7 x_8 a_1 + x_7 a_2 \Omega_r + b_2 u_2, \quad (37)$$
$$\dot{x}_7 = x_6 x_8 a_3 + x_6 a_4 \Omega_r + b_3 u_3, \quad (38)$$
$$\dot{x}_8 = x_6 x_7 a_5 + b_4 u_4, \quad (39)$$

where $\Omega_r = -\Omega_1 + \Omega_2 - \Omega_3 + \Omega_4$, $u_1 = F_Z/b$, $u_2 = T_\phi/(Lb)$, $u_3 = T_\theta/(Lb)$, $u_4 = T_\psi/D$, $g$ is the gravitational acceleration constant, and

$$\begin{aligned} &b_1 = \tfrac{b}{m}, \quad b_2 = \tfrac{Lb}{J_X}, \quad b_3 = \tfrac{Lb}{J_Y}, \quad b_4 = \tfrac{D}{J_Z}, \\ &a_1 = \tfrac{J_Y - J_Z}{J_X}, \quad a_2 = -\tfrac{J_r}{J_X}, \quad a_3 = \tfrac{J_Z - J_X}{J_Y}, \quad a_4 = \tfrac{J_r}{J_Y}, \\ &a_5 = \tfrac{J_X - J_Y}{J_Z}. \end{aligned} \quad (40)$$

Numerical values for the parameters appear in Table I.

The control signals are $\mathbf{u} = [u_1\ u_2\ u_3\ u_4]^T$. The term $\Omega_r(\mathbf{u})$ cannot technically be accounted for in a control of type (11); here the CMAC is provided with the value at the previous control time step $\Omega_r(t - \Delta t)$, and small changes in $\Omega_r$ over that time step provide a small disturbance to the system which is assumed to be zero in the control design.

With desired states $x_{1,\text{des}} = Z_{\text{des}} = 0, x_{2,\text{des}} = \phi_{\text{des}}, x_{3,\text{des}} = \theta_{\text{des}}, x_{4,\text{des}} = \psi_{\text{des}}$ the auxiliary errors can be defined

$$z_j = \Lambda(x_j - x_{j,\text{des}}) + (x_{j+4} - x_{j+4,\text{des}}), \quad \text{for } j = 1, 2, 3, 4 \quad (41)$$

TABLE II: CMAC input normalization

| Input | Minimum | Maximum |
|---|---|---|
| $z_1$ | -10 m | 10 m |
| $z_2, z_3, \phi, \theta$ | -1 rad | 1 rad |
| $\psi$ | -0.1 rad | 0.1 rad |
| $\dot{\phi}, \dot{\theta}$ | -1 rad/s | 1 rad/s |
| $\dot{\psi}$ | -0.1 rad/s | 0.1 rad/s |
| $\Omega_R$ | -10 rad/s | 10 rad/s |

TABLE III: PID gains

| Coordinate | Auxiliary Error $z_j$ | $K_P$ | $K_D$ | $K_I$ |
|---|---|---|---|---|
| Roll | $z_2$ | $3.01 \times 10^4$ | $2.72 \times 10^4$ | $8.3 \times 10^3$ |
| Pitch | $z_3$ | $3.01 \times 10^4$ | $2.72 \times 10^4$ | $8.3 \times 10^3$ |
| Yaw | $z_4$ | $5.24 \times 10^5$ | $4.75 \times 10^5$ | $1.45 \times 10^5$ |
| $Z$ | $z_1$ | $5.83 \times 10^5$ | $5.28 \times 10^5$ | $1.61 \times 10^5$ |

Defining $\mathbf{z} = [z_1\ z_2\ z_3\ z_4]^T$ the dynamics (35) through (39) become

$$\dot{\mathbf{z}} = \mathbf{f}(\mathbf{x}, \dot{\phi}_{\text{des}}, \dot{\theta}_{\text{des}}, \ddot{\phi}_{\text{des}}, \ddot{\theta}_{\text{des}}, \Omega_r(t-\Delta t)) + \mathbf{d}(t) + \mathbf{B}\mathbf{u}. \quad (42)$$

The adaptive control Lyapunov function

$$V = \mathbf{z}^T \mathbf{B}^{-1} \mathbf{z} + \sum_{k=1}^{4} \tilde{\mathbf{w}}_k^T \tilde{\mathbf{w}}_k, \quad (43)$$

is similar to (7) and thus the techniques described in Section II and Section III are appropriate.

Note that in the physical system the commanded rotor speeds are calculated using

$$\begin{bmatrix} \Omega_1^2 \\ \Omega_2^2 \\ \Omega_3^2 \\ \Omega_4^2 \end{bmatrix} = \begin{bmatrix} 0.25 & 0 & -0.5 & 0.25 \\ 0.25 & -0.5 & 0 & -0.25 \\ 0.25 & 0 & 0.5 & 0.25 \\ 0.25 & 0.5 & 0 & -0.25 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix}. \quad (44)$$

The Cartesian outer-loop control attempts to achieve hovering at $X_{\text{des}} = Y_{\text{des}} = Z_{\text{des}} = 0$ by providing desired angular and vertical coordinates as derived in [7]

$$\phi_{\text{des}} = \sin^{-1}\left(-G_{o,1} Y - G_{o,2} \dot{Y}\right), \quad (45)$$

$$\theta_{\text{des}} = -\sin^{-1}\left([-G_{o,1} X - G_{o,2} \dot{Y}]/\cos\phi\right), \quad (46)$$

$$\psi_{\text{des}} = 0, \quad (47)$$

$$Z_{\text{des}} = 0, \quad (48)$$

where $G_{o,1}$ and $G_{o,2}$ are positive control gains.

### B. Simulations

*1) Parameters:* The outer loop control uses gains $G_{o,1} = G_{o,2} = 1$. The tuned PID control has a settling time of 0.87 seconds and percent overshoot of 21% for each inner loop $z_j$ (gains given in Table III). Note that PIDs with smaller overshoot result in larger transient errors in the Cartesian coordinates.

The CMAC uses spline basis functions, has $m = 100$ layers, has $Q = 10$ quantizations in each input, and uses the minimum and maximum inputs listed in Table II. The CMAC uses adaptation gains $\beta_j = 4 \times 10^6$ for $j = 1, 2, 3, 4$.

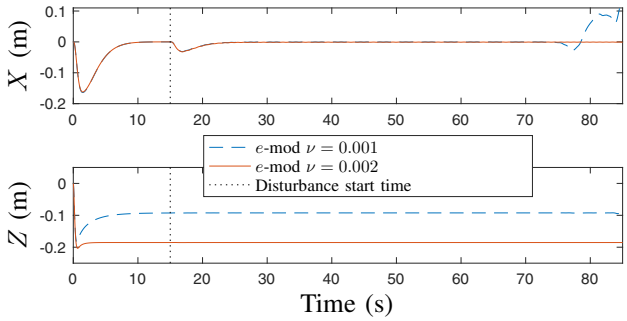


Fig. 3: **e-modification.** A too-small value of $\nu = 0.001$ results in bursting at 75s. Bursting is prevented when $\nu = 0.002$ at the cost of a large steady-state error in $Z$ of 19cm.

In order to have a fair comparison with the PID control, the feedback gains $\Lambda_j$ in (41) and $G_j$ in (28) are equivalent to the PID feedback gains i.e. $\Lambda_j = K_{P,j}/K_{D,j}$ and $G_j = K_{D,j}$.

In the simulations weights start at zero, so the quadrotor will initially drop in $Z$. The initial pitch is 5.7 degrees with other angular coordinates starting at zero. A wind-like offset sinusoid torques the system in the pitch coordinate starting at 15s

$$T_{d,\text{pitch}} = 1 + 0.5\sin(2\pi t) \text{ Nm}.$$

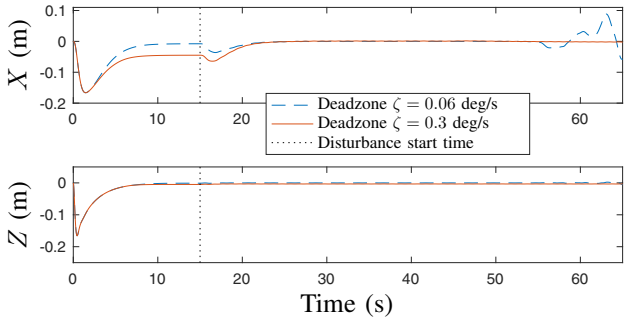


Fig. 4: **Deadzone.** A too-small $\zeta = 0.06$ results in bursting at 55s. Choosing $\zeta = 0.3$ prevents bursting but results in a large steady-state error in $X$ of 4.5cm in the disturbance-free case before 15s.

*2) e-modification:* Using a single CMAC in control (11) with standard $e$-modification (12) results in a trade-off between performance and stability. Choosing $\nu = 0.001$ results in excellent performance at first, but bursting occurs after 75 seconds (Fig. 3, top graph). The bursting occurs due to too much weight drift; specifically when the maximum weight magnitude in the pitch CMAC reaches 4.3 bursting occurs (Fig. 5, dashed line). If one increases $\nu$ to 0.002 in an effort to halt bursting, the performance in $Z$ is poor with a steady-state error of 19cm (Fig. 3, bottom graph).

*3) Deadzone:* Using a single CMAC in control (11) with standard deadzone (15) also results in a trade-off between performance and stability. Choosing the deadzone to be $\zeta = 0.06$ deg/s results in excellent performance at first, but bursting occurs after 55s (Fig. 4, top graph). The bursting

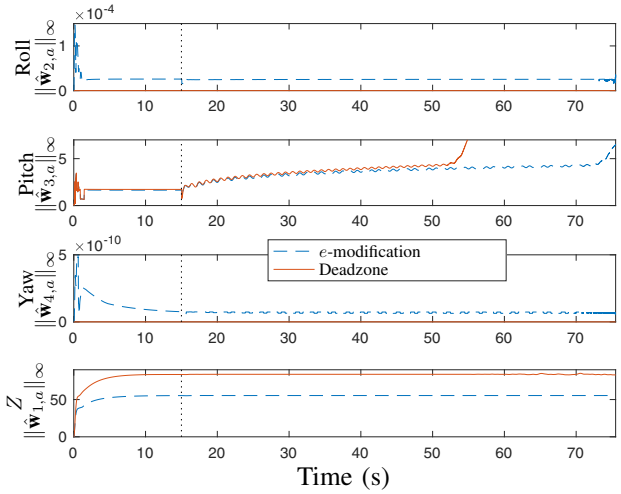


Fig. 5: **Weights for deadzone and e-modification.** In both cases bursting occurs when the maximum pitch weight magnitude reaches a value of 4.3.

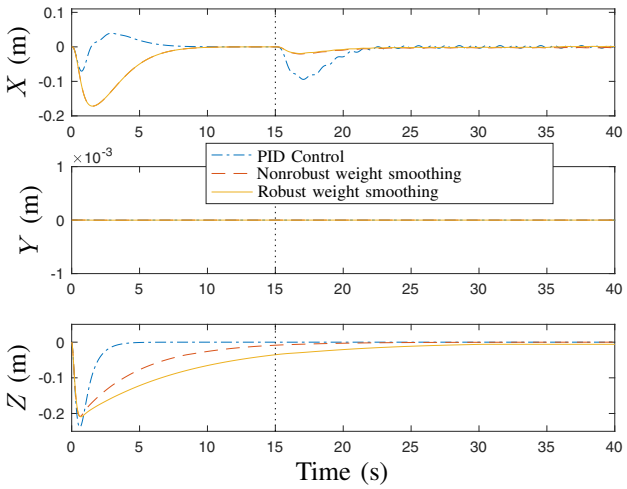


Fig. 6: **Proposed methods vs PID in Cartesian coordinates.** PID does better before the CMACs are trained, but after training the CMAC methods both outperform PID in disturbance rejection by a factor of 5 in $X$ position.

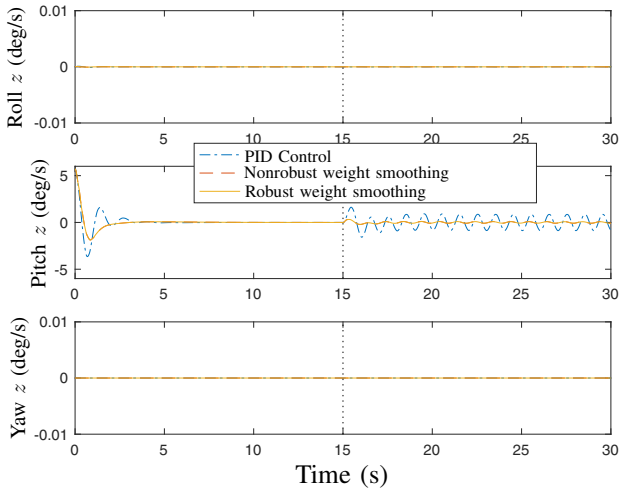


Fig. 7: **Proposed methods vs PID in angular coordinates.** After the disturbance is applied the proposed methods outperform PID by a factor of 9.

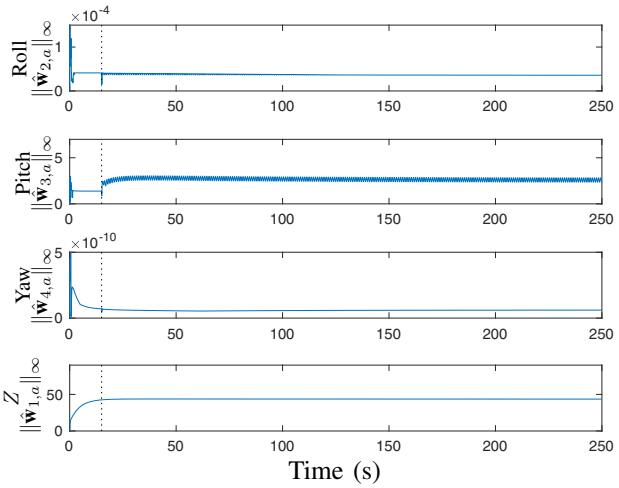


Fig. 8: **Nonrobust method of weight smoothing.** The weights are well behaved even though bounded weights are not guaranteed with this method, and remain far short of the value of 4.3 in pitch that resulted in bursting for $e$-mod and deadzone.

occurs due to too much weight drift; specifically when the maximum weight magnitude in the pitch CMAC reaches 4.3 bursting occurs (Fig. 5, solid line). If one increases $\zeta$ to 0.3 deg/s in an effort to halt bursting, the disturbance-free performance (before 15s) in $X$ is poor with a steady-state error of 4.5cm (Fig. 4, top graph).

*4) Nonrobust weight smoothing:* Using a single CMAC in control (11) with the proposed nonrobust weight smoothing algorithm (27), with $K = 0.0001$, does slightly worse that PID in $X$ and $Z$ during the first 15 seconds (Fig. 6): however the weights start at an initial condition of zero and the first 15 seconds is purely a training phase, where it is not surprising that PID does better. After 15 seconds the sinusoidal disturbance is applied and the performance in $X$ has an overshoot of 0.02m, whereas PID has an overshoot of 0.095m (Fig. 6, top graph). Looking at the value for the auxiliary error in the steady-state, the CMAC continues to reject the disturbance with a pitch oscillation of only 0.1 deg/s amplitude, compared to a 0.9 deg/s amplitude oscillation for PID (Fig. 7, middle graph). The weights are well behaved and stay below a value of 3 for maximum magnitude in the pitch CMAC, well below the value of 4.3 that caused bursting for $e$-mod and deadzone (Fig. 8). However, even though the weights appear well behaved there is no mathematical guarantee of bounded weights.

*5) Robust weight smoothing:* The robust weight smoothing uses both a robust and performance CMAC in parallel in control (28), in order to have mathematical guarantees of UUB errors and weights. The robust CMAC trains using $e$-

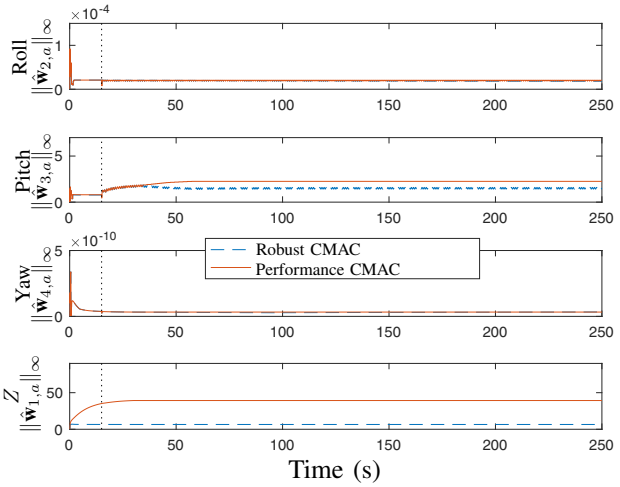


Fig. 9: **Robust method of weight smoothing.** This method results in a guarantee of UUB errors and weights, and the weights remain far short of the value of 4.3 in pitch that resulted in bursting for $e$-mod and deadzone.

mod (12) at all times, but with a value of $\nu = 0.01$ ( which is conservative by a factor of 5 based on the results with $\nu = 0.002$ in Fig. 3). The performance CMAC trains using robust weight smoothing (30) with $K = 0.0001$, with training stopped after a cell has been activated for a total of $T_c = 30$ seconds overall. The robust weight smoothing performs as well as the nonrobust algorithm, but trains slightly slower (Fig 6). The performance weights converge and stay below a maximum magnitude of 2.25 (Fig. 9) In addition, the proposed robust weight smoothing produces controls with 10% smaller RMS values than PID control, with significantly smoother signals (Fig. 10).

## V. Conclusions

The method of weight smoothing is a technique that has previously been proposed for preventing overlearning in CMAC during static training and preventing weight drift in CMAC adaptive control (without sacrificing performance). However, the technique as previously proposed was computationally complex when there are more than one or two inputs, and did not guarantee stability when used in adaptive control. This paper proposes a method to achieve computational efficiency and a way to guarantee UUB signals using Lyapunov techniques. The results with a simulated quadrotor helicopter subject to sinusoidal disturbance shows the proposed method does prevent weight drift without sacrificing performance, unlike the traditional robust methods of $e$-modification and deadzone. Compared to a tuned PID control, the proposed method (after 15 seconds of training) has 5 times less overshoot in Cartesian coordinates, 9 times less oscillation amplitude in angular velocity, 10% less control effort, and noticeably smoother control signals.

## References


[1] J. Albus, "A new approach to manipulator control: the cerebellar model articulation controller (CMAC)," *J. Dyn. Sys. Meas. Contr.*, vol. 97, pp. 220–227, 1975.

[2] C. Macnab, "Preventing bursting in approximate-adaptive control when using local basis functions," *Fuzzy Sets Syst.*, vol. 160, pp. 439–462, February 2009.

[3] F. Chen and C. Chang, "Practical stability issues in CMAC neural network control systems," *IEEE Trans. Control Syst. Technol.*, vol. 4, pp. 86–91, 1996.

[4] S. Commuri, S. Jagannathan, and F. L. Lewis, "CMAC neural network control of robot manipulators," *Journal of Robotic Systems*, vol. 14, p. 465482, 1997.

[5] M. Abdelhameed, U. Pinspon, and S. Cetinkunt, "Adaptive learning algorithm for CMAC," *Mechatronics*, vol. 12, pp. 859–873, 2002.

[6] S.-K. Wang, J.-Z. Wang, and D.-W. Shi, "CMAC-based compound control of hydraulically driven 6-DOF parallel manipulator," *Journal of Mechanical Science and Technology*, vol. 25, no. 6, pp. 1595–1602, 2011.

[7] C. Nicol, C. Macnab, and A. Ramirez-Serrano, "Robust adaptive control of a quadrotor helicopter," *Mechatronics*, vol. 21, no. 6, pp. 927–938, 2011.

[8] K. Masaud and C. Macnab, "Preventing bursting in adaptive control using an introspective neural network algorithm," *Neurocomputing*, vol. 136, p. 300314, 2014.

[9] L. Kraft and J. Pallotta, "Real-time vibration control using CMAC neural networks with weight smoothing," in *American Control Conference*, (Chicago), pp. 3939–3943, 2000.

[10] J. Pallotta and L. Kraft, "Two dimensional function learning using CMAC neural network with optimized weight smoothing," in *American Control Conference*, (San Diego), pp. 373–377, 1999.

[11] B. Yang and H. Han, "A CMAC-PD compound torque controller with fast learning capacity and improved output smoothness for electric load simulator," *International Journal of Control, Automation, and Systems*, vol. 12, no. 4, pp. 805–812, 2014.

[12] J. Albus, "Data storage in the cerebellar model articulation controller (CMAC)," *J. Dyn. Sys. Meas. Contr.*, vol. 97, pp. 228–233, 1975.

[13] K. Narendra and A. Annaswamy, "A new adpative law for robust adaptation without persistant excitation," *IEEE Trans. Autom. Control*, vol. AC-32, pp. 134–145, Feb. 1987.

[14] B. Egardt, *Stability of Adaptive Controllers*. New York: Springer-Verlag, 1979.

[15] B. Yang, H. Han, and R. Bao, "An intelligent CMAC-PD torque controller with anti-over-learning scheme for electric load simulator," *Transactions of the Institute of Measurement and Control*, vol. 38, no. 2, pp. 192–200, 2016.


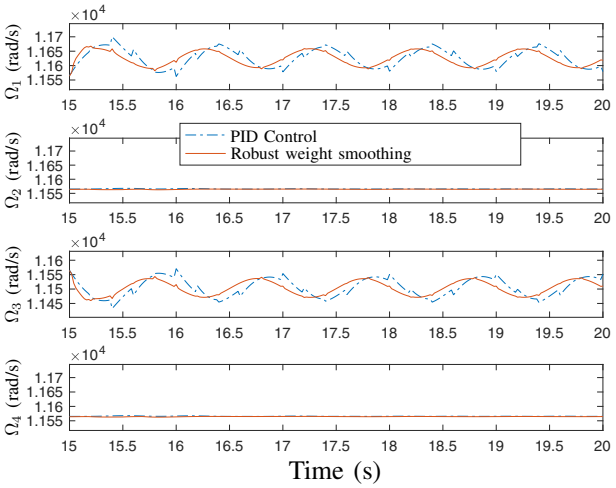


Fig. 10: **Robust weight smoothing vs PID rotor speeds.** The proposed method produces less control effort and smoother control signals than PID.