2017 IEEE 56th Annual Conference on Decision and Control (CDC)
December 12-15, 2017, Melbourne, Australia

Control of a Quadrotor Helicopter using a Stable Introspective-CMAC Adaptive Control

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) neural network utilizes hypercube basis function domains, called cells. Because of the local nature of the cells the CMAC exhibits more weight drift (also called parameter drift or overlearning) compared to other types of neural networks. Typical robust weight update law modifications designed to prevent drift, like deadzone or e-modification, often sacrifice performance. This paper proposes a solution to this problem by using two CMACs in the control law, referred to as a performance CMAC and a robust CMAC. The performance CMAC reduces the state error to a small value during initial training, and then turns off its weight updates according to the output of a decision algorithm in order to capture the best performance. One proposed algorithm simply turns off the training of a weight after its cell has been activated for a certain period of time. A more advanced method, deemed the introspective algorithm, stops training in a cell when it appears that the average error measured over sequential cell domains is no longer being reduced by the weight update in a cell. The robust CMAC trains in parallel using a conservative e-modification algorithm, continuing to provide robustness after the performance CMAC has stopped training. Lyapunov methods show a guarantee of uniformly ultimately bounded signals. Simulations with a quadrotor helicopter demonstrate the superior performance of the proposed method over e-modification, deadzone, and a PID control.**

I. Introduction

Neural networks often serve as nonlinear approximators in Lyapunov-stable adaptive control schemes i.e. neural-adaptive control. The Cerebellar Model Articulation Controller (CMAC) [1] suits real-time control applications, training faster than a Multi-Layer Perceptron (MLP) and handling more inputs than a Radial Basis Function Network (RBFN). In all types of neural networks, if proper precautions are not taken, the weights can grow larger than desired during training. In static training this results in poor approximation between training points, usually referred to as *overlearning*. In traditional adaptive control *parameter drift* occurs, or *weight drift* in neural-adaptive control. Too much drift in the adaptive parameters or weights will cause *bursting* (where the state error suddenly diverges after a period of apparent convergence).

Some have modified CMAC training using standard adaptive-control robust methods to guarantee uniformly ultimately bounded (UUB) signals and thus prevent drift and bursting, including deadzone [2], adaptive-parameter (weight) projection [3], [4], leakage (σ-modification) [5], and e-modification (e-mod) [6],[7]. However, these methods tend to sacrifice performance in the case of CMAC, especially in the case of non-minimum phase or open-loop unstable systems. Another approach uses a switching control (discontinuous robust control) [8],[9],[10],[11], which can theoretically drive the state error to zero but results in control signal chatter.

Some researchers have made efforts to find novel solutions that do not sacrifice performance for stability. Methods using variable adaptation rates appear in [12],[13] without stability proofs. This author proposed methods that use an alternate set of weights which can guarantee UUB signals [14],[15]. A modified weight smoothing algorithm for on-line adaptation appears in [16] without stability proof. This author also proposed an introspective neural network algorithm in [17] without stability proof; the introspective algorithm is tested on an experimental quadrotor in [18]. This paper proposes a way to ensure UUB signals using the introspective algorithm; the introspective algorithm provides an advantage over the methods in [14],[15] because it does not require carefully chosen (tuned) parmeters to prevent weight drift.

The proposed control utilizes two CMACs in parallel. The *robust* CMAC uses a conservative e-mod design. The *performance* CMAC bounds its weights during initial training, and then eventually turns off adaptation (for each weight individually) according to some algorithm. This paper tests two such algorithms: the first simply turns off adaptation after a certain time period of cell activation, while the second uses the introspective algorithm from [17]. In the introspective algorithm, each cell votes on whether it *thinks* the total weight update during the last cell activation has reduced the average error or not, where the average error is measured over subsequent cell activations on the same CMAC array. In a quadrotor helicopter simulation the proposed methods outperform deadzone, e-modification and a tuned PID control when there is a significant sinusoidal torque disturbance.

II. Background

A. CMAC

Similar to an RBFN, a CMAC output consists of a weighted sum of basis functions

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

where $\mathbf{x}_{\text{in}} \in \mathcal{R}^n$ is the input vector, $\mathbf{\Gamma} \in \mathcal{R}^{1\times N}$ is a row vector of basis functions and $\hat{\mathbf{w}} \in \mathcal{R}^N$ is a vector of weight estimates (estimates of the unknown ideal weights

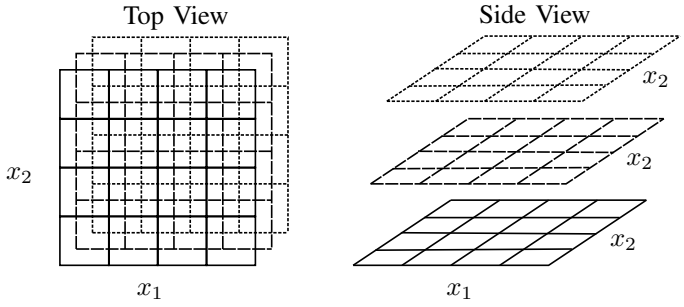

Fig. 1: A CMAC with $q = 4$ quantizations, $n = 2$ inputs, and $m = 3$ arrays

$\mathbf{w}$). Like an RBFN, given enough basis functions the CMAC can uniformly approximate a nonlinear function $f(\mathbf{x}_{\text{in}})$ in a region $\mathcal{D} \subset \mathcal{R}^n$ as in

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

where the approximation error is bounded by positive constant $\epsilon_{\max}$ i.e. $|\epsilon| < \epsilon_{\max}\ \forall\, \mathbf{x}_{\text{in}} \in \mathcal{D}$.

The CMAC differs from an RBFN by using local basis function domains, deemed *cells*. The cells consist of n-dimensional hypercubes arranged in m offset arrays, each array having q divisions on each input (Figure 1). The input indexes (*activates*) one cell per array, and the computational efficiency of the CMAC stems from only having to perform m (rather than $N = mq^n$) calculations to get the output. Rather than allocating N memory locations, the CMAC uses a hash-coding scheme to store (previously) activated weights in a small physical memory [19].

B. CMAC Adaptive Control

This section considers CMAC control of a second-order system

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

with output $y = x_1$ and desired trajectory $y_{\text{des}}(t), \dot{y}_{\text{des}}(t)$ and $\ddot{y}_{\text{des}}(t)$, where f contains both linear and nonlinear terms, d contains bounded disturbances, and b is a positive constant. For trajectory tracking one can define the auxiliary error

$$z = \Lambda e_1 + e_2 = \Lambda(x_1 - y_{\text{des}}) + (x_2 - \dot{y}_{\text{des}}), \tag{5}$$

where Λ is a positive constant. (Note when $z \equiv 0$ then $e_2 = -\Lambda e_1$ is a stable linear system). By defining the weight estimation error as $\tilde{\mathbf{w}} = \mathbf{w} - \hat{\mathbf{w}}$ the direct adaptive control Lyapunov function becomes

$$V_i = \frac{1}{2b}z^2 + \frac{1}{2\beta}\tilde{\mathbf{w}}^T\tilde{\mathbf{w}}, \tag{6}$$

where β is a constant adaptation rate. The time derivative is

$$\dot{V} = \frac{1}{b}z(f(x_1, x_2) + bu + d(t) - \ddot{y}_{\text{des}}) - \frac{1}{\beta}\tilde{\mathbf{w}}^T\dot{\hat{\mathbf{w}}}, \tag{7}$$
$$= z\left(\frac{f(x_1, x_2) - \ddot{y}_{\text{des}}}{b} + u + d(t)/b\right) - \frac{1}{\beta}\tilde{\mathbf{w}}^T\dot{\hat{\mathbf{w}}}. \tag{8}$$

A CMAC can uniformly approximate $\frac{f(x_1,x_2)-\ddot{y}_{\text{des}}}{b}$ giving

$$\dot{V} = z[\mathbf{\Gamma}(\mathbf{x}_{\text{in}})\mathbf{w} + \epsilon(\mathbf{x}_{\text{in}}) + u + d(t)/b] - \frac{1}{\beta}\tilde{\mathbf{w}}^T\dot{\hat{\mathbf{w}}}, \tag{9}$$

where $\mathbf{x}_{\text{in}} = [x_1\ x_2\ \ddot{y}_{\text{des}}]^T$. Using control

$$u = -\mathbf{\Gamma}(\mathbf{x}_{\text{in}})\hat{\mathbf{w}} - Gz, \tag{10}$$

where G is a positive constant control gain gives

$$\dot{V} = z(\mathbf{\Gamma}\tilde{\mathbf{w}} + \epsilon + d/b - Gz) - \frac{1}{\beta}\tilde{\mathbf{w}}^T\dot{\hat{\mathbf{w}}}, \tag{11}$$
$$= -Gz^2 + z(\epsilon + d/b) + \tilde{\mathbf{w}}^T(\mathbf{\Gamma}^T z - \dot{\hat{\mathbf{w}}}/\beta). \tag{12}$$

Assuming that positive constant $d_{\max}$ exists such that

$$|\epsilon + d/b| < d_{\max}, \tag{13}$$

then one possible robust modification for the weight update is deadzone where

$$\dot{\hat{\mathbf{w}}} = \begin{cases} \beta\mathbf{\Gamma}^T z & \text{if } |z| > \zeta, \\ 0 & \text{otherwise.} \end{cases} \tag{14}$$

If one chooses positive constant deadzone size $\zeta > d_{\max}/G$ then

$$\dot{V} < -Gz^2 + |z|d_{\max} < 0 \quad \forall\, |z| > \zeta > d_{\max}/G, \tag{15}$$

and the signals will be UUB. Specifically, if the initial condition is $z = 0$ (reasonable for trajectory tracking) then an ultimate bound is given by

$$V(|z|, \|\tilde{\mathbf{w}}\|) = V(\zeta, \tilde{\mathbf{w}}_0), \tag{16}$$

where $\tilde{\mathbf{w}}_0$ is the initial weight error.

Another possible robust weight update is e-mod

$$\dot{\hat{\mathbf{w}}} = \beta(\mathbf{\Gamma}^T z - \nu|z|\hat{\mathbf{w}}), \tag{17}$$

which results in Lyapunov derivative

$$\dot{V} = -Gz^2 + z(\epsilon + d/b) + \nu|z|\tilde{\mathbf{w}}^T\mathbf{w} - \nu|z|\tilde{\mathbf{w}}^T\tilde{\mathbf{w}}, \tag{18}$$
$$\dot{V} < |z|(-G|z| + zd_{\max} + \nu\|\tilde{\mathbf{w}}\|\|\mathbf{w}\| - \nu\|\tilde{\mathbf{w}}\|^2). \tag{19}$$

Thus $\dot{V} < 0$ when either $|z| > \delta_z$ or $\|\tilde{\mathbf{w}}\| > \delta_w$ where

$$\delta_z = \frac{d_{\max}}{G} + \frac{\nu\|\mathbf{w}\|^2}{4G}, \tag{20}$$
$$\delta_w = \frac{\|\mathbf{w}\|}{2} + \sqrt{\frac{d_{\max}}{\nu} + \frac{\|\mathbf{w}\|^2}{4}}. \tag{21}$$

The signals are UUB, with an ultimate bound given by Lyapunov surface

$$V(|z|, \|\tilde{\mathbf{w}}\|) = V(\delta_z, \delta_w). \tag{22}$$

An appropriate (conservative) value of ν must be established through testing.

III. Proposed Methods

The proposed methods uses two CMACs in parallel in control law

$$u = -\mathbf{\Gamma}\hat{\mathbf{w}} - \mathbf{\Gamma}\hat{\mathbf{p}} - Gz. \tag{23}$$

where $\mathbf{\Gamma}\hat{\mathbf{w}}$ is the robust CMAC and $\mathbf{\Gamma}\hat{\mathbf{p}}$ the performance CMAC . The robust CMAC trains using e-mod (17) with a conservative value of ν. The performance CMAC trains using a bounded weight update at first, but then makes decisions on when to turn off adaptation in each cell. The expression for the ith weight update is

$$\dot{\hat{p}}_i = \begin{cases} \beta\Gamma_i z & \text{if } \mathcal{A}_i \text{ and } \mathcal{B}_i, \\ 0 & \text{otherwise,} \end{cases} \tag{24}$$

where $\mathcal{A}_i$ bounds the weights using

$$\mathcal{A}_i = \begin{cases} 0 & \text{if } \hat{p}_i \geq p_{\max} \text{ and } z > 0, \\ 0 & \text{if } \hat{p}_i \leq p_{\min} \text{ and } z < 0, \\ 1 & \text{otherwise,} \end{cases} \tag{25}$$

and $\mathcal{B}_i$ is the output of a decision algorithm

$$\mathcal{B}_i = \begin{cases} 1 & \text{if decision is made to keep updating,} \\ 0 & \text{otherwise.} \end{cases} \tag{26}$$

Thus, the performance CMAC should be capable of reducing the errors to small values in the initial training, regardless of how conservative the value of ν is in the robust CMAC. Also, the robust CMAC training will result in a guarantee of UUB signals regardless of the choice of $p_{\min}$ and $p_{\max}$. The algorithm $\mathcal{B}$ then tries to capture the best performance by halting the training of the performance CMAC at the appropriate time. The robust CMAC will continue to train after the performance CMAC has become static, providing continued robustness and guarantees of UUB signals.

The new Lyapunov function and its time derivative are

$$V = \frac{1}{2b}z^2 + \frac{1}{\beta}\tilde{\mathbf{w}}^T\tilde{\mathbf{w}} + \frac{1}{\beta}\tilde{\mathbf{p}}^T\tilde{\mathbf{p}}, \tag{27}$$

$$\dot{V} = z\left(\frac{f(x_1,x_2) - \ddot{y}_d}{b} + d/b + u\right) - \frac{\tilde{\mathbf{w}}\dot{\hat{\mathbf{w}}} + \tilde{\mathbf{p}}_t\dot{\hat{\mathbf{p}}}_t}{\beta}, \tag{28}$$

where $\hat{\mathbf{p}}_t$ denotes the performance CMAC weights that still continue to be updated. Since $\mathbf{\Gamma}\mathbf{w}$ is capable of uniformly approximating the nonlinear terms, so is $\mathbf{\Gamma}\mathbf{w}+\mathbf{\Gamma}\mathbf{p}_t$, and thus

$$\dot{V} = z(\mathbf{\Gamma}\mathbf{w} + \mathbf{\Gamma}\mathbf{p}_t + \epsilon + d/b + u) - (\tilde{\mathbf{w}}\dot{\hat{\mathbf{w}}} + \tilde{\mathbf{p}}_t\dot{\hat{\mathbf{p}}}_t)/\beta. \tag{29}$$

Adding $\mathbf{\Gamma}\mathbf{p}_t$ does not change the size of ϵ. Thus using control (23) and (17) for the updates to $\hat{\mathbf{w}}$ and (24) for the $\hat{\mathbf{p}}$ updates give

$$\dot{V} = -Gz^2 + z(\epsilon + d) + \nu|z|\tilde{\mathbf{w}}^T\mathbf{w} - \nu|z|\tilde{\mathbf{w}}^T\tilde{\mathbf{w}}, \tag{30}$$

$$\dot{V} < |z|(-G|z| + zd_{\max} + \nu\|\tilde{\mathbf{w}}\|\|\mathbf{w}\| - \nu\|\tilde{\mathbf{w}}\|^2). \tag{31}$$

Thus δ_z and δ_w are the same size as (20) and (21) respectively, and a uniform ultimate bound on the signals is given by

$$V(|z|, \|\tilde{\mathbf{w}}\|, \|\tilde{\mathbf{p}}\|) = V(\delta_z, \delta_w, P_{\max}), \tag{32}$$

where $\|\hat{\mathbf{p}}\| < P_{\max}$ with $P_{\max}$ a positive constant due to (25).

One advantage of this algorithm over normal adaptive-parameter projection is that $p_{\max}$ does not have to be chosen larger than $\sup_{i=1}^{N}(p_i)$, and $p_{\min}$ does not have to be smaller than $\inf_{i=1}^{N}(p_i)$. Thus unlike adaptive-parameter projection neither *a-priori* knowledge of the system model nor extensive pre-training are needed, although reasonable choices of $p_{\max}$ and $p_{\min}$ will have to be established through some testing. Also, in adaptive-parameter projection a persistant disturbance will typically cause the weights to reach the imposed bounds on $\hat{\mathbf{p}}$, whereas in this algorithm $p_{\max}$ and $p_{\min}$ only serve as a safety measure and with proper design of $\mathcal{B}$ the weights should never reach these bounds.

In summary, the performance CMAC will first reduce the error as much as possible (for the given approximation ability), since it doesn't have the performance-limiting e-mod term. Then the performance CMAC stops training while the robust CMAC updates remain in operation in order to react to system disturbances.

A. Method 1 (Timing): Stop adaptation in a performance CMAC cell after a certain cut-off time

Consider defining the notation for a CMAC memory location as $L_{j,k}$ for the kth cell activated sequentially on array j. Furthermore, $L_{j,k(i)}$ will refer to the location of the cell on its ith activation i.e. an oscillation between exactly two cells will result in $L_{j,k(i)} = L_{j,k(i-1)+2} = L_{j,k(i)-2}$ (Fig. 2).

In this first method the total time that a cell has been activated for the ith time is indicated by $T(L_{j,k(i)})$ and

$$\mathcal{B}(L_{j,k(i)}) = \begin{cases} 1 & \text{if } T(L_{j,k(i)}) < T_c, \\ 0 & \text{otherwise,} \end{cases} \tag{33}$$

where T_c is the cut-off time. A test must establish an appropriate value of T_c.

B. Method 2 (Introspective): Stop adaptation in a performance CMAC cell according to the introspective algorithm from [17]

The introspective algorithm keeps updating a weight only when all the weight updates are clearly still reducing the error. It takes advantage of the local cell domains of the CMAC algorithm: the algorithm can measure the average error during a cell activation and decide whether a weight update appears to be reducing that error or not. Each cell then *votes* on whether to keep updating using a weighted vote. At the moment a cell is deactivated, the sum of all the total votes from all the activated cells decides whether or not to keep updating that cell's weight in the future.

The measurement of error associated with a particular cell will include S cells, i.e. including the $S-1$ subsequent cells on the same array, so calculations are made for cell

$$\mathcal{L} = L_{j,k(i)-S} \tag{34}$$

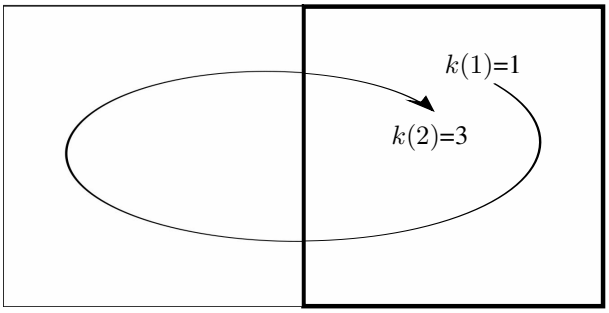

Fig. 2: For counter $k(i)$ for CMAC cell $L_{j,k(i)}$ (in bold) i indicates the number of activation times for this cell while k indicates the total number of sequential cell activations (on the jth array).

The average error calculated over these S cells is

$$Z_i(\mathcal{L}) = \frac{1}{\Delta T_{j,k(i)-S}} \int_{T_{j,k(i)-S}}^{T_{j,k(i)+1}} z \, dt, \tag{35}$$

where $T_{j,k(i)}$ indicates the time cell $L_{j,k(i)}$ becomes activated. The change in error between cell activations is the measurement of interest

$$\Delta Z_i(\mathcal{L}) = Z_i(\mathcal{L}) - Z_{i-1}(\mathcal{L}). \tag{36}$$

The total weight update made during a cell activation is

$$\Delta p_i(L_{j,k(i)}) = \int_{T_{j,k(i)}}^{T_{j,k(i)+1}} \dot{\hat{p}}_i(L_{j,k(i)}) \, dt. \tag{37}$$

The decisions behind the voting stem from the following logic:

1) The estimate of the change in average error with respect to the ith weight change should be negative during the training phase (if not training should stop),
2) Weight updates should also halt once the average error crosses the origin, indicating the error can no longer be reduced by increasing the weights,
3) If a weight has already stopped updating then its cell will vote to stop updating the other weights.

A positive vote is a vote to keep updating, and the vote is weighted by the weight magnitude so that

$$\text{vote}(\mathcal{L}) = \begin{cases} |\hat{p}_i(\mathcal{L})| & \text{if } \dfrac{\Delta Z_i(\mathcal{L})}{\Delta w_{i-1}(\mathcal{L})} < 0, \\ & \text{and } Z_i(\mathcal{L}) Z_{i-1}(\mathcal{L}) > 0, \\ & \text{and } \text{vote}_{i-1}(\mathcal{L}) > 0, \\ -|\hat{p}_i(\mathcal{L})| & \text{otherwise,} \end{cases} \tag{38}$$

The voting result for cell $\mathcal{L}$ stays in memory. Thus, a memory will decide the fate of the weight update for the cell that has just been deactivated

$$\mathcal{N} = L_{j,k(i)}. \tag{39}$$

If the votes are negative, then at the moment the cell is deactivated the algorithm discards the total weight update made during the activation

$$p(L_{j,k(i+1)}) = \begin{cases} p_i(\mathcal{N}) & \text{if } \Sigma_{J=1}^{m} \text{vote}(L_{J,k(i)}) > 0, \\ p_i(\mathcal{N}) - \Delta p_i(\mathcal{N}) & \text{otherwise.} \end{cases} \tag{40}$$

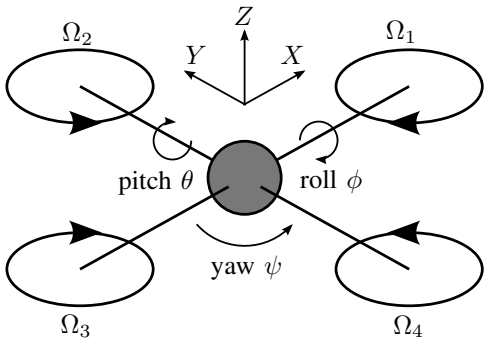

Fig. 3: Coordinates and assumption of rotor spin directions

TABLE I: Quadrotor parameters

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

IV. Results

A. Quadrotor helicopter

The quadrotor has four rotors with speeds $\Omega_1, \Omega_2, \Omega_3, \Omega_4$, with speed directions defined in Figure 3. Each rotor inertia is given by J_r and the inertia matrix of the system is diag(J_X, J_Y, J_Z). The basic angular coordinates are roll ϕ, pitch θ, and yaw ψ.

The outer loop control in this work is designed for hovering so that desired Cartesian coordinates are $X_{\text{des}} = Y_{\text{des}} = Z_{\text{des}} = 0$. The appropriate outer loop output (desired states provided to the inner loop control) are derived in [15] as

$$\phi_{\text{des}} = \sin^{-1}\left(-K_1 Y - K_2 \dot{Y}\right), \tag{41}$$

$$\theta_{\text{des}} = -\sin^{-1}\left([-K_1 X - K_2 \dot{Y}]/\cos\phi\right), \tag{42}$$

$$\psi_{\text{des}} = 0, \tag{43}$$

$$Z_{\text{des}} = 0, \tag{44}$$

where K_1 and K_2 are positive control gains.

The states for the inner loop are $x_1 = Z$, $x_2 = \phi$, $x_3 = \theta, x_4 = \psi, x_5 = \dot{Z}, x_6 = \dot{\phi}, x_6 = \dot{\theta}$ and $x_8 = \dot{\psi}$.

In this model formulation Z is the vertical direction defined by the body frame rather than absolute vertical direction. The force acting in Z is

$$F_Z = b\Omega_1^2 + b\Omega_2^2 + b\Omega_3^2 + b\Omega_4^2, \tag{45}$$

TABLE II: PID gains

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

TABLE III: CMAC input normalization

Input	Minimum	Maximum
z_1	-10 m	10 m
z_2, z_3, ϕ, θ	-1 rad	1 rad
ψ	-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
Ω_R	-10 rad/s	10 rad/s

where b is a constant thrust coefficient of the propellers. The actuating torques are

$$\mathbf{T} = \begin{bmatrix} T_\phi \\ T_\theta \\ T_\psi \end{bmatrix} = \begin{bmatrix} Lb(\Omega_4^2 - \Omega_2^2) \\ Lb(\Omega_3^2 - \Omega_1^2) \\ D(\Omega_1^2 + \Omega_3^2 - \Omega_2^2 - \Omega_4^2) \end{bmatrix}, \quad (46)$$

where L is length from rotor to quadrotor center of mass, and D is a constant drag factor. Using an assumption of slowly-varying rotor speeds the quadrotor dynamics derived in [15] are

$$\dot{x}_i = x_{i+4}, \quad \text{for } i = 1, 2, 3, 4, \quad (47)$$
$$\dot{x}_5 = -g\cos(x_2)\cos(x_3) + b_1 u_1, \quad (48)$$
$$\dot{x}_6 = x_7 x_8 a_1 + x_7 a_2 \Omega_r + b_2 u_2, \quad (49)$$
$$\dot{x}_7 = x_6 x_8 a_3 + x_6 a_4 \Omega_r + b_3 u_3, \quad (50)$$
$$\dot{x}_8 = x_6 x_7 a_5 + b_4 u_4, \quad (51)$$

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

$$b_1 = \tfrac{b}{m}, \quad b_2 = \tfrac{Lb}{J_X}, \quad b_3 = \tfrac{Lb}{J_Y}, \quad b_4 = \tfrac{D}{J_Z},$$
$$a_1 = \tfrac{J_Y - J_Z}{J_X}, \quad a_2 = -\tfrac{J_r}{J_X}, \quad a_3 = \tfrac{J_Z - J_X}{J_Y}, \quad a_4 = \tfrac{J_r}{J_Y},$$
$$a_5 = \tfrac{J_X - J_Y}{J_Z}. \quad (52)$$

Numerical values for the parameters appear in Table I. The inner loop control is $\mathbf{u} = [u_1 \ u_2 \ u_3 \ u_4]^T$. Since $\Omega_r(\mathbf{u})$ the CMACs in roll and pitch have the signal $\Omega_r(t - \Delta t)$ as an input, where Δt is the discrete update time-step of the CMAC. Thus changes in Ω_r must be approximately zero over one time step in order to both maintain validity of the model and to maintain stability of the controller. In particular, the use of a switching control (discontinuous robust control) would be unsuitable.

Given desired states $x_{1,\text{des}} = Z_{\text{des}} = 0, x_{2,\text{des}} = \phi_{\text{des}}, x_{3,\text{des}} = \theta_{\text{des}}, x_{4,\text{des}} = \psi_{\text{des}}$ and auxiliary errors

$$z_j = \Lambda(x_j - x_{j,\text{des}}) + (x_{j+4} - x_{j+4,\text{des}}), \quad \text{for } j = 1, 2, 3, 4 \quad (53)$$

with $\mathbf{z} = [z_1 \ z_2 \ z_3 \ z_4]^T$ the dynamics (47) through (51) can be expressed using the slowly-varying rotor assumption

$$\dot{\mathbf{z}} = \mathbf{f}(\mathbf{x}, \dot{\phi}_{\text{des}}, \dot{\theta}_{\text{des}}, \ddot{\phi}_{\text{des}}, \ddot{\theta}_{\text{des}}, \Omega_r(t - \Delta t)) + \mathbf{d}(t) + \mathbf{B}\mathbf{u}. \quad (54)$$

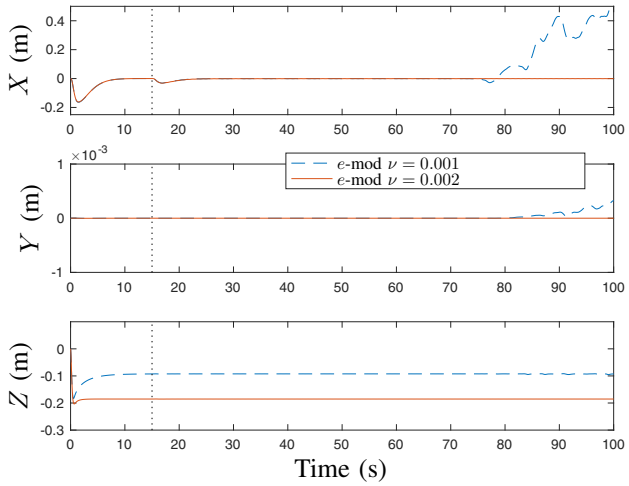

Fig. 4: The results using e-mod. Note $\nu = 0.001$ is too small, resulting in bursting behaviour, while $\nu = 0.002$ is too large resulting in a large steady-state error in Z of 19cm. (Start of disturbance is indicated by a vertical dotted line.)

Thus, the adaptive control Lyapunov function

$$V = \mathbf{z}^T \mathbf{B}^{-1} \mathbf{z} + \sum_{i=1}^{4} \tilde{\mathbf{w}}_i^T \tilde{\mathbf{w}}_i, \quad (55)$$

has the same form as (6) and the derivation of controls and weight updates follows as in Section II-B.

Note that in a physical system once the control signals have been calculated the motor controllers would receive the commanded rotor speeds calculated from

$$\begin{bmatrix} \Omega_1^2 \\ \Omega_2^2 \\ \Omega_3^2 \\ \Omega_4^2 \end{bmatrix} = \begin{bmatrix} 0.25 & 0 & -0.5 & 0.25 \\ 0.25 & -0.5 & 0 & -0.25 \\ 0.25 & 0 & 0.5 & 0.25 \\ 0.25 & 0.5 & 0 & -0.25 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix}. \quad (56)$$

B. Control Parameters

The outer loop control uses gains $K_1 = K_2 = 1$. The tuned PID control exhibits a settling time of 0.87 seconds and percent overshoot of 21% for each inner loop z_j (Table II). Note that PIDs with smaller overshoot result in larger transient errors in the Cartesian coordinates.

Each CMAC has $m = 100$ arrays, $Q = 10$ quantizations in each input, with minimum and maximum inputs given in Table III. The CMACs use adaptation gains $\beta_j = 4 \times 10^6$ for $j = 1, 2, 3, 4$. In order to have a fair comparison with the PID control, the feedback gains Λ_j in (53) and G_j in (23) are equivalent to the PID feedback gains i.e. $\Lambda_j = K_{P,j}/K_{D,j}$ and $G_j = K_{D,j}$. The CMACs receive inputs

$$\mathbf{\Gamma}_1() = \mathbf{\Gamma}_1(z_1, \phi, \theta), \quad (57)$$
$$\mathbf{\Gamma}_2() = \mathbf{\Gamma}_2(z_2, \dot{\phi}, \dot{\theta}, \dot{\psi}, \Omega_r(t - \Delta t)), \quad (58)$$
$$\mathbf{\Gamma}_3() = \mathbf{\Gamma}_3(z_3, \dot{\phi}, \dot{\theta}, \dot{\psi}, \Omega_r(t - \Delta t)), \quad (59)$$
$$\mathbf{\Gamma}_4() = \mathbf{\Gamma}_4(z_4, \dot{\phi}, \dot{\theta}). \quad (60)$$

The introspective algorithm uses $S = 2$ i.e. it measures average error over 2 sequential cell activations on the same

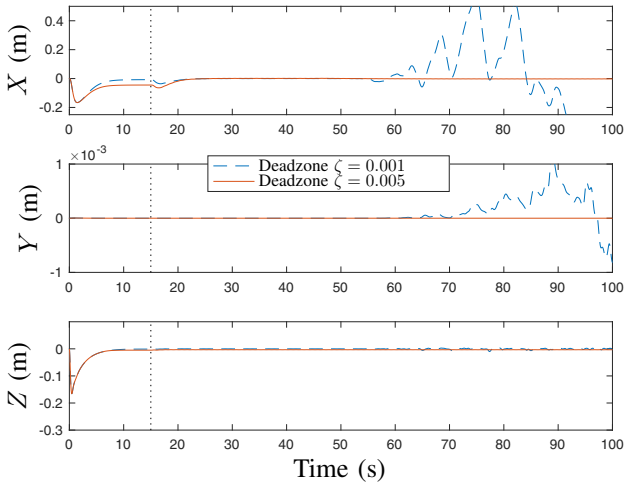

Fig. 5: The results using deadzone. Note $\zeta = 0.001$ is too small, resulting in bursting behaviour, while $\zeta = 0.005$ is too large resulting in large steady-state error in X of 4.5cm.

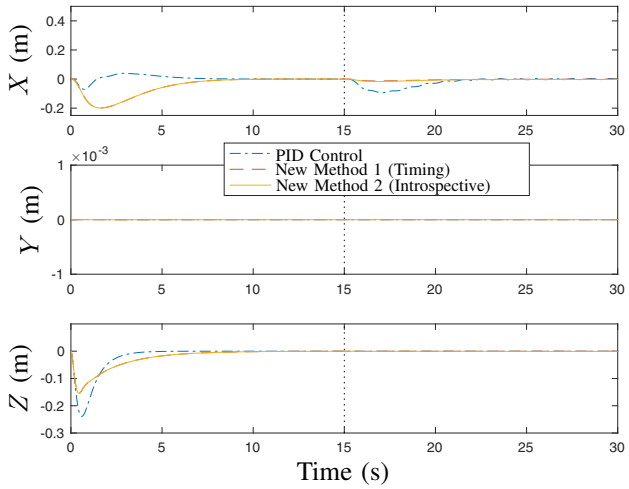

Fig. 6: Comparing the proposed methods to PID in Cartesian coordinates. Note PID outperforms the proposed methods during the initial training phase, but after training the proposed methods can reject the disturbance while PID has a 10 cm overshoot. (The two proposed methods have nearly identical performance.)

layer for making decision (and thus an oscillation between exactly two cells will result in a measurement over an entire period of oscillation).

The simulations start with zero speed for the propellers so that the quadrotor will drop in Z until the rotors have attained the required speed. The pitch offset starts at 5.7 degrees, while the roll and yaw angles and all angular velocities start at zero. This initial phase constitutes the training phase for the CMAC, as all weights start at zero. Note that there is no nominal (feedforward gravity compensation) control signal for the Z coordinate. At the 15 second mark an offset sinusoidal torque disturbs the system in the pitch coordinate to simulate the effect of wind

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

C. Simulations

The e-mod technique using $\nu = 0.001$ results in continued weight drift and eventual bursting behaviour after 60 seconds of sinusoidal disturbance (Figure 4, where the start of the disturbance is indicated with a vertical dotted line). Increasing ν to 0.002 results in a large steady-state error of 19 cm in Z. Note that the e-mod term multiplied by ν limits weight magnitudes, but in order to overcome the large gravitational term the CMAC for Z requires relatively large weight magnitudes.

The deadzone technique using a deadzone size of $\zeta = 0.001$ does not stop the weight drift and results in bursting (Figure 5). Increasing the deadzone size to $\zeta = 0.005$ results in a steady-state error of 4.5cm in X and 0.4 cm in Z before the disturbance is applied (Figure 5). The large steady-state error in X stems from the outer loop control (42), which can command very small desired changes in pitch (falling within the deadzone) even for relatively large values of error in X. (Note that the X error does go near zero after the disturbance is applied.)

The results from the e-mod and deadzone simulations suggest weight bounds for roll, pitch, and yaw as $p_{i,\max} = 4, p_{i,\min} = -4$ for $i = 2, 3, 4$ and for Z as $p_{1,\max} = 0, p_{1,\min} = -80$. However, both Method 1 and Method 2 stop the weight drift before these maximum/minimum bounds are reached; the bounds remain solely a matter of guaranteeing stability and do not affect the performance.

Based on the results from the e-mod simulations, the proposed methods use a conservative value of $\nu = 0.01$ for the e-mod term in the robust CMAC. Also based on previous simulations, the proposed Method 1 (Timing) cuts off training at time $T_c = 30$ s for each cell. The PID outperforms the two proposed methods during the initial phase (Figure 6 and Figure 7). This is not surprising as this constitutes the training phase for the CMAC (the weights start at zero) and one expects neural network controls to outperform a tuned PID only after some training has occurred. At the 15 second mark the step disturbance appears, and then the PID results in an overshoot in X of 10 cm, whereas Method 1 and Method 2 reject the disturbance with only a 1.4 cm and 1.8 cm overshoot respectively (Figure 6, top graph). The continued sinusoidal disturbance causes the PID control to oscillate in pitch with an amplitude in z_3 of 4 deg/s, whereas Method 1 and Method 2 oscillate in z_3 with an amplitude of only 0.3 deg/s and 0.5 deg/s respectively (Figure 7, second graph from top). With the sinusoidal disturbance, PID control causes rotor speeds Ω_1 and Ω_3 to oscillate with an amplitude of 0.77 rad/s (7.4 rpm), whereas Method 1 and Method 2 both cause a 0.60 rad/s (5.7 rpm) oscillation (Figure 8). Thus, the control uses less effort with the proposed methods. The speed variations remain smoother with the proposed methods than with PID control.

Method 1 (Timing) slightly outperforms Method 2 (Introspective) because the chosen value of T_c is large enough that

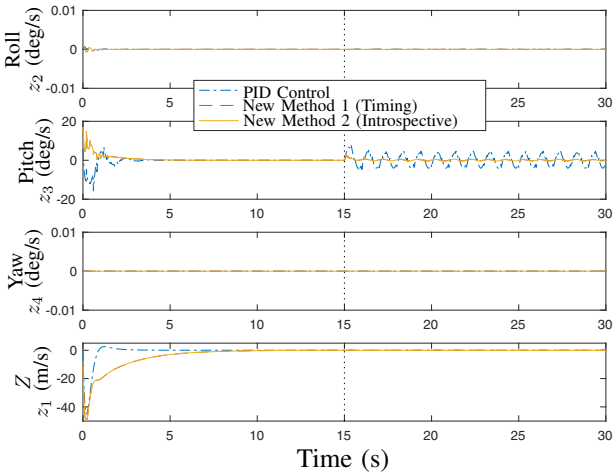

Fig. 7: Comparing the auxiliary errors. Note the proposed methods can reject the disturbance in pitch while PID results in a significant wobble.

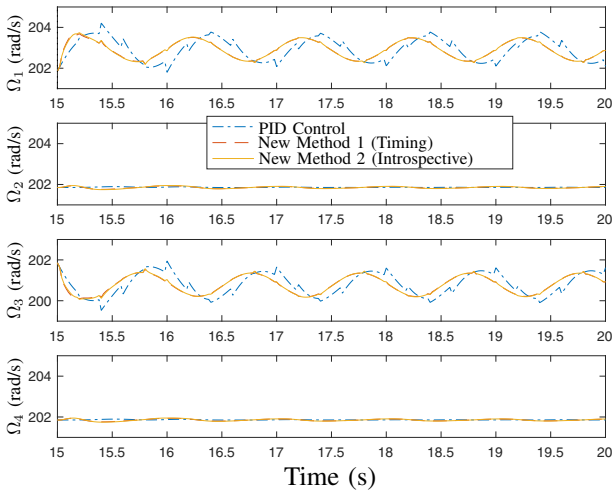

Fig. 8: Comparing rotor speeds. Note the proposed methods result in less control effort and smoother control signals than PID

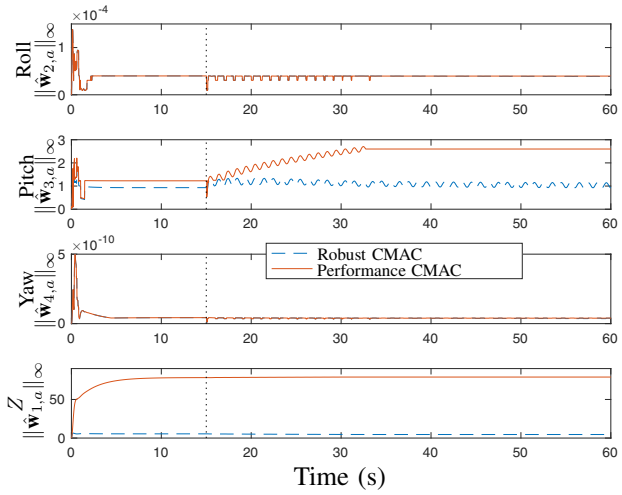

Fig. 9: New Method 1 (Timing) maximum weights. (The maximum weight magnitude of all the currently activated weights is shown).

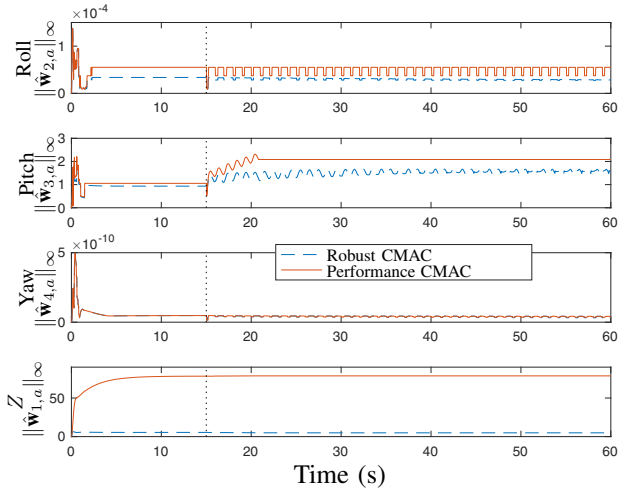

Fig. 10: New Method 2 (Introspective) maximum weights.

it allows the weights to grow in the pitch control long after the introspective Method 2 has stopped the weight growth (Figure 9 and Figure 10, second graph from top). If T_c had been chosen more conservatively then Method 2 would have the better performance. The disadvantage of Method 1 is that some information about the system (based on testing) is required to identify a suitable value of T_c; choosing T_c too high in a quest for performance risks bursting while choosing T_c conservatively low sacrifices performance. In contrast, the introspective Method 2 automatically finds the correct cut-off time.

V. Conclusions

This paper proposes a method to prevent overlearning and parameter/weight drift when using CMACs in a direct adaptive control scheme, since unchecked weight drift can lead to bursting. The method uses two CMACs in the control, the first deemed a robust CMAC and the second a performance CMAC. The robust CMAC trains using a conservative robust weight update (e-modification) at all times, ensuring uniformly ultimately bounded signals. The performance CMAC trains without a robust update at first, then turns its adaptation off after a certain period of time. One way to decide when to turn the update off is using the introspective algorithm from previous work, where the CMAC can make its own decision on when a weight update is no longer reducing the state error. Simulations of a quadrotor helicopter show how bursting can occur (due to weight drift) when using e-modification and deadzone if the parameters are not chosen appropriately. The proposed method outperforms PID control while eliminating weight drift, and results in less control effort and smoother control signals than PID.

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

[3] C.-F. Hsu, "Adaptive dynamic CMAC neural control of nonlinear chaotic systems with l 2 tracking performance," *Engineering Applications of Artificial Intelligence*, vol. 25, no. 5, pp. 997–1008, 2012.

[4] F. O. Rodríguez, J. de Jesús Rubio, C. R. M. Gaspar, J. C. Tovar, and M. A. M. Armendáriz, "Hierarchical fuzzy CMAC control for nonlinear systems," *Neural Computing and Applications*, vol. 23, no. 1, pp. 323–331, 2013.

[5] T.-F. Wu, P.-S. Tsai, F.-R. Chang, and L.-S. Wang, "Adaptive fuzzy CMAC control for a class of nonlinear systems with smooth compensation," *IEE Proceedings-Control Theory and Applications*, vol. 153, no. 6, pp. 647–657, 2006.

[6] S. Commuri, S. Jagannathan, and F. L. Lewis, "CMAC neural network control of robot manipulators," *Journal of Robotic Systems*, vol. 14, pp. 465–482, Dec. 1997.

[7] T. Mai and Y. Wang, "Adaptive force/motion control system based on recurrent fuzzy wavelet CMAC neural networks for condenser cleaning crawler-type mobile manipulator robot," *IEEE Transactions on Control Systems Technology*, vol. 22, no. 5, pp. 1973–1982, 2014.

[8] T. F. Wu, H. C. Huang, P. S. Tsai, N. T. Hu, and Z. Q. Yang, "Adaptive fuzzy CMAC design for an omni-directional mobile robot," in *Intl. Conf. Intelligent Information Hiding and Multimedia Signal Processing*, (Kitakyushu), pp. 839–843, 2014.

[9] C.-C. Tsai, Y.-P. Ciou, F.-C. Tai, S.-F. Su, and C.-M. Lin, "Backstepping station-keeping control using recurrent wavelet fuzzy CMAC for ball-driven chairs," in *Intl. Conf. on Machine Learning and Cybernetics*, vol. 2, (Lanzhou), pp. 721–727, IEEE, 2014.

[10] T. Ngo and T. Phuong, "Robust adaptive self-organizing wavelet fuzzy CMAC tracking control for de-icing robot manipulator," *Intl. J. of Computers, Communication, and Control*, vol. 10, no. 4, pp. 567–578, 2015.

[11] S. Y. Wang, C. L. Tseng, and C. J. Chiu, "Online speed controller scheme using adaptive supervisory tsk-fuzzy CMAC for vector controlled induction motor drive," *Asian Journal of Control,*, vol. 17, no. 2, pp. 569–581, 2015.

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

[13] 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.

[14] C. Macnab, "Neural-adaptive control using alternate weights," *Neural Computing Applications*, vol. 20, no. 2, pp. 211–231, 2011.

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

[16] 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.

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

[18] T. Clark and C. Macnab, "Cerebellar model articulation controller with introspective voting weight updates for quadrotor application," in *IEEE Information Technology, Electronics, and Mobile Communication Conference*, (Vancouver,Canada), pp. 1–7, 2016.

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