Neural Processing Letters **19:** 49–61, 2004.
© 2004 *Kluwer Academic Publishers. Printed in the Netherlands.*

A Learning Algorithm of CMAC Based on RLS

TING QIN, ZONGHAI CHEN, HAITAO ZHANG, SIFU LI, WEI XIANG and MING LI
Department of Automation, University of Science and Technology of China, Hefei 230027, China. e-mail:dragon@mail.ustc.edu.cn

Abstract. Conventionally, least mean square rule which can be named CMAC-LMS is used to update the weights of CMAC. The convergence ability of CMAC-LMS is very sensitive to the learning rate. Applying recursive least squares (RLS) algorithm to update the weights of CMAC, we bring forward an algorithm named CMAC-RLS. And the convergence ability of this algorithm is proved and analyzed. Finally, the application of CMAC-RLS to control nonlinear plant is investigated. The simulation results show the good convergence performance of CMAC-RLS. The results also reveal that the proposed CMAC-PID controller can reject disturbance effectively, and control nonlinear time-varying plant adaptively.

Key words. CMAC, CMAC-PID controller, nonlinear plant, recursive least squares algorithm

Nomenclature

X	the input space
A_c	Conceptual Memory
$\|A_c\|$	the size of Conceptual Memory
A_p	Physical Memory
$\|A_p\|$	the size of Physical Memory
c	the generalization factor
A^*	the number of the activated neurons
Q	the levels of quantization
d	the dimension of the input vector
φ_k	the indicator of the kth input sample
$nzpos_k$	the set to indicate the positions of those nonzero elements in φ_k without hash-coding
$\hat{\theta}_k$	the estimated weight vector after the kth sample data is presented and finished learning
λ	the learning rate
RMSE	the root mean square errors
P_c, I_c, D_c	the proportional, integral and derivative parameters
fp	the filter parameter of the first order low-pass filter
u_{pid}	the control input of the PID controller
uf_{pid}	the value after the control input of the PID controller is filtered
u_{cmac}	the control input of CMAC inverse controller

uf_{cmac} the value after the control input of the CMAC inverse controller is filtered
uf_{total} the control input of the CMAC-PID controller

1. Introduction

CMAC, which stands for cerebellar model articulation controller, was first proposed based on the model of human cerebellum by Albus [5, 6] three decades ago. The higher-order CMAC with B-splines [13] or Gaussian [3] receptive field functions can further improve the accuracy of function approximation. In 1992, Parks and Miltizer [12] defined a Lyapunov function to prove that CMAC learning converges to a limited cycle given that the learning rate equals to one. In the same year, Wong and Siders [20] showed that CMAC learning is equivalent to solving a linear system with a Gauss–Seidel iteration scheme. Lin and Chiang [4] showed us the different convergence characteristics at different learning rate. Further research [2, 11] in theoretical development also promotes the application. Because CMAC is capable of fast learning and possesses good local generalization property, moreover, it is easy to be realized in hardware [8] and software. In resent years, it has been successfully applied in many fields, such as robotic control [15–18], machine control [9], pattern recognition [14], and signal processing [1], etc.

Traditionally, CMAC uses LMS rule to update its weights. However, using this learning algorithm, the convergence ability is very sensitive to the learning rate. Once the learning rate is provided improperly, the convergence will worse off, sometimes even to the extent of divergence. Based on RLS, we bring forward CMAC-RLS different from CMAC-LMS which eliminates this shortcoming, and so it provides a stable modeling algorithm for control adaptively on line.

The Letter is organized as follows. First, the mechanism of CMAC is briefly introduced in Section 2; in Section 3, we will prove and analyze the convergence characteristics of RLS even if the number of the sample is less than that of the unknowns. And then RLS is applied to update the weights of CMAC. In Section 4, implementing CMAC-RLS to control nonlinear plant on line is further studied. Conclusion is given in Section 6. While a large number of notations are used in the paper, we summarize them in the nomenclature for easier reference.

2. Mechanism of CMAC

A schematic sketch of CMAC is illustrated in Figure 1. It can be regarded including three mappings:

$$X \xrightarrow{M} A_c \tag{1}$$

$$A_c \xrightarrow{H} A_p \tag{2}$$

$$A_p \xrightarrow{S} Y \tag{3}$$

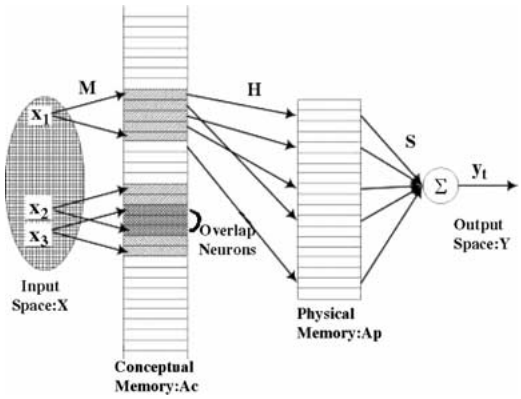

Figure 1. Basic skeleton of CMAC.

where X, A_c and A_p, Y are input space, the space of conceptual memory and the space of physical memory and output space, respectively.

First, the input vector is quantized and then activates A^* association neurons in A_c (i.e., in Figure 1, $A^* = 5$) by (1). Without hash-coding, the weights attached to the activated neurons in A_c are summed to produce the output.

If two input vectors are close, there will be overlap between their corresponding activated neurons, such as x_2 and x_3. On the contrary, the neurons activated by input vectors far away do not overlap, such as x_1 and x_2.

In view of a practical system, if the quantization is very fine, the size of the conceptual memory may be too large (this is especially true for multidimensional inputs) to implement. Albus solved this problem by hash coding the conceptual memory into a small physical memory. The activated neurons in A_c will map the neurons in A_p by (2). Hence, we sum the weights of the mapped neurons in A_p to get the output.

A key parameter of CMAC is the generalization factor denoted by c. In general, all the neurons activated by one input sample are in a A^* hypercubic region and

$$A^* = (2 \times c + 1)^d \tag{4}$$

where d is the dimension of the input vector.

Given a set of sample data (x_k, y_k), where x_k is the input vector and y_k is the desired output vector. We represent the neurons that x_k activates by a vector of characteristic function a_{i_k} $(1 \leqslant i \leqslant |A_c|)$, where

$$a_{i_k} = \begin{cases} 1 & \text{the } i\text{th neuron in } A_c \text{ is activated by } x_k \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

Let us set $n = |A_c|$ and call the vector of characteristic function indicator and denote it as

$$\varphi_k = [a_{1_k}, a_{2_k}, \ldots, a_{n_k}], \tag{6}$$

$$\text{Form the matrix } A_k = \begin{bmatrix} \varphi_1^T & \varphi_2^T & \cdots & \varphi_k^T \end{bmatrix}_{k\times n}^T, \quad Y_k = \begin{bmatrix} y_1 & y_2 & \cdots & y_k \end{bmatrix}_{k\times 1}^T. \tag{7}$$

Where k is the number of training samples. The goal of CMAC learning is to find a weight vector θ_k such that

$$A_k\theta_k = Y_k \tag{8}$$

where $\theta_k^T = [w_{1k}, w_{2k}, \ldots, w_{nk}]$ and represents the contents in A_c after the kth set of sample data has been presented and the calculating is over.

Traditionally, the weight vector of CMAC is updated using the LMS rule. That is

$$\theta_k^{(i)} = \theta_{k-1}^{(i)} + \lambda(y_k - \varphi_k\theta_{k-1}^{(i)})/A^* \tag{9}$$

where θ_k^i is the weight vector when the kth sample is presented in the ith iteration (before updating), λ is the learning rate.

It has been proved in reference [20] that on condition that no two different inputs have the same indicators, θ_k can be calculated by $A_k^T(A_kA_k^T)^{-1}Y_k$, and the result is equal to the result calculated by CMAC-LMS, but there may be two inputs whose corresponding indicators are the same, and therefore $A_kA_k^T$ is not invertible. However, under this circumstance we still have

$$\theta_k = A_k^+ Y_k \tag{10}$$

where A^+ denote the Moore–Penrose inverse of A.

Hence we can see that if (8) represents compatible equations, θ_k is the minimum norm solution of (8); and if (8) represents contradictory equations, θ_k is the minimum norm least squares solution.

If hash-coding is used, $|A_c|$ in the description above should be changed to $|A_p|$.

3. CMAC-RLS Algorithm

3.1. CMAC-RLS WITHOUT HASH-CODING

The recursive least squares algorithm is described in reference [10], but it is restricted to the case that the number of samples is more than that of the unknowns at least. By mathematical analysis we will point out that the estimated values of the unknowns converge to the minimum norm solution if the equations composed by the sample data are compatible equations or minimum norm least squares solution in case that they are contradictory equations.

LEMMA 3.1.

$$\lim_{\beta\to+\infty}(1/\beta I_n + A^TA)^{-1}A^T = A^+. \tag{11}$$

Where I_n is an $n \times n$ unit matrix and β is a positive number.

Proof. The procedure of the proof can be referred to reference [19]. □

LEMMA 3.2. *Given an $n \times n$ matrix P with full rank and an $m \times n$ matrix A, if $P^{-1} + A^TA$ is invertible, then*

$$(P^{-1} + A^T A)^{-1} = P - PA^T(I_m + APA^T)^{-1}AP. \tag{12}$$

Proof. The procedure of the proof can be referred to reference [10].

THEOREM 3.1. *In using the* RLS *Algorithm, the estimated values of the unknowns converge to the minimum norm solution if the equations composed by sample data are compatible equations or minimum norm least squares solution in case that they are contradictory equations.*

Proof. We define

$$P_0 = \beta I_n \tag{13}$$

$$P_{k+1} = (P_0^{-1} + A_{k+1}^T A_{k+1})^{-1} \tag{14}$$

$$\hat{\theta}_{k+1} = P_{k+1} A_{k+1}^T Y_{k+1}. \tag{15}$$

Due to (11) and (10), we can conclude

$$\begin{aligned} \forall k, \lim_{\beta \to +\infty} \hat{\theta}_{k+1} &= \lim_{\beta \to +\infty} P_{k+1} A_{k+1}^T Y_{k+1} = \lim_{\beta \to +\infty} (P_0^{-1} + A_{k+1}^T A_{k+1})^{-1} A_{k+1}^T Y_{k+1} \\ &= A_{k+1}^+ Y_{k+1} = \theta_{k+1}. \end{aligned} \tag{16}$$

Hence, if β is set as some large constant in computer calculation, we can regard that $\forall k$, $\hat{\theta}_{k+1}$ converges to θ_{k+1} and call $\hat{\theta}_k$ the estimated vector of θ_k.

With (7) and (14), P_{k+1} can also be expressed recursively as

$$P_{k+1} = (P_k^{-1} + \varphi_{k+1}^T \times \varphi_{k+1})^{-1} \tag{17}$$

According to (12), one can have

$$P_{k+1} = P_k - P_k \varphi_{k+1}^T (1 + \varphi_{k+1} P_k \varphi_{k+1}^T)^{-1} \varphi_{k+1} P_k \tag{18}$$

$\hat{\theta}_{k+1}$ can also be expressed recursively as follows:

$$\begin{aligned} \hat{\theta}_{k+1} &= P_{k+1} A_{k+1}^T Y_{k+1} \\ &= P_{k+1}(A_k^T Y_k + \varphi_{k+1}^T y_{k+1}) \\ &= \{P_k - P_k \varphi_{k+1}^T (1 + \varphi_{k+1} P_k \varphi_{k+1}^T)^{-1} \varphi_{k+1} P_k\}(A_k^T Y_k + \varphi_{k+1}^T y_{k+1}) \\ &= P_k A_k^T Y_k - P_k \varphi_{k+1}^T \varphi_{k+1} (1 + \varphi_{k+1} P_k \varphi_{k+1}^T)^{-1} \varphi_{k+1} P_k A_k^T Y_k + \\ &\quad + P_k \varphi_{k+1}^T y_{k+1} - P_k \varphi_{k+1}^T (1 + \varphi_{k+1} P_k \varphi_{k+1}^T)^{-1} \varphi_{k+1} P_k \varphi_{k+1}^T y_{k+1} \\ &= P_k A_k^T Y_k - P_k \varphi_{k+1}^T (1 + \varphi_{k+1} P_k \varphi_{k+1}^T)^{-1} \varphi_{k+1} P_k A_k^T Y_k + \\ &\quad + P_k \varphi_{k+1}^T y_{k+1} (1 + \varphi_{k+1} P_k \varphi_{k+1}^T)^{-1} (1 + \varphi_{k+1} P_k \varphi_{k+1}^T - \varphi_{k+1} P_k \varphi_{k+1}^T) y_{k+1} \\ &= P_k A_k^T Y_k - P_k \varphi_{k+1}^T (1 + \varphi_{k+1} P_k \varphi_{k+1}^T)^{-1} \varphi_{k+1} P_k A_k^T Y_k + \\ &\quad + P_k \varphi_{k+1}^T y_{k+1} (1 + \varphi_{k+1} P_k \varphi_{k+1}^T)^{-1} y_{k+1} \\ &= \hat{\theta}_k + P_k \varphi_{k+1}^T (1 + \varphi_{k+1} P_k \varphi_{k+1}^T)^{-1} (y_{k+1} - \varphi_{k+1} \hat{\theta}_k). \end{aligned} \tag{19}$$

For convenience, we define

$$L_{k+1} = P_k \varphi_{k+1}^T / (1 + \varphi_{k+1} P_k \varphi_{k+1}^T). \tag{20}$$

Then taking (13), (18), (19) and (20) generates the following recursive learning algorithm: The initial values of P_k and $\hat{\theta}_k$ are set as

$$P_0 = \beta I_n \quad (\beta \text{ is set as some large constant}) \quad \text{and} \quad \hat{\theta}_0 = 0, \tag{21}$$

$$\forall k, \quad L_{k+1} = P_k \varphi_{k+1}^T / (1 + \varphi_{k+1} P_k \varphi_{k+1}^T), \tag{22}$$

$$P_{k+1} = P_k - L_{k+1} \varphi_{k+1} P_k, \tag{23}$$

$$\hat{\theta}_{k+1} = \hat{\theta}_k + L_{k+1}(y_{k+1} - \varphi_{k+1} \hat{\theta}_k), \tag{24}$$

Having said that, we are finally ready to propose the on-line CMAC-RLS learning as follows:

(1) Initialize the values of P_k and $\hat{\theta}_k$ by (21);
(2) Quantize x_k when a new sample (x_k, y_k) is presented;
(3) Compute φ_k by (5) and (6);
(4) Update the weight vector by (22), (23), (24);
(5) Go to (2) until the end.

It can be directly calculated that the time complexity of RLS for each iteration is $O(|A_c|^3)$.

Since the RLS derived above holds for any k, it implies that after the learning using the first sample is finished, $\hat{\theta}_1$ will converge to the minimum solution of the equation composed of the first set of sample data.

Continuing this procedure, when the kth sample is presented and the recursive learning is finished, $\hat{\theta}_k$ will converge to the minimum norm solution if the equations formed by all the preceding k sample data are compatible equations or minimum norm least squares solution in case that they are contradictory equations. And this establishes Theorem 3.1.

Note that θ_k equals to the result learned by traditional LMS rule, and for any k, $\hat{\theta}_k$ converges to θ_k, we can infer that the weight vector updated by CMAC-RLS converges to what is learned by CMAC-LMS at each step. However, CMAC-RLS is more suitable for on-line learning. For RLS, we can regard L_{k+1} in (24) as the learning rate which is obtained from the history data and varies at each step to guarantee the convergence of CMAC-RLS.

The neurons in A_c activated by x_k constitute a hypercube. Due to that characteristic, we can simplify CMAC-RLS in order to reduce the time complexity.

Let us define $nzpos_k$ as the set to indicate the positions of those nonzero elements in φ_k, then the simplified algorithm can be described as follows:

$$\hat{\theta}_{k+1} = \hat{\theta}_k + L_{k+1}[y_{k+1} - \varphi_{k+1}(nzpos_k)\hat{\theta}_k(nzpos_k)] \tag{25}$$

$$L_{k+1} = \frac{P_k(:, nzpos_k)\varphi_{k+1}^T(nzpos_k)}{1 + \varphi_{k+1}(nzpos_k)P_k(nzpos_k, nzpos_k)\varphi_{k+1}^T(nzpos_k)} \tag{26}$$

$$P_{k+1} = P_k - L_{k+1}\varphi_{k+1}(nzpos_k)P_k(nzpos_k, :)\,. \tag{27}$$

Remarks. If B is a matrix, $B(nzpos_k, :)$ represent the elements of B whose rows are the elements of $nzpos_k$; $B(:, nzpos_k)$ represents the elements of B whose columns are the elements of $nzpos_k$: $B(nzpos_k, nzpos_k)$ represents the elements of B whose rows are the elements of $nzpos_k$ and columns are the elements of $nzpos_k$.

If B is a vector, $B(nzpos_k)$ represent the elements of B whose positions are the elements of $nzpos_k$.

And it also can be easily derived that the time complexity of the simplified algorithm is $O(|A_c|^2+(A^*)^2|A_c|)$. Generally, $(A^*)^2$ is less than $|A_c|$, and thus the time complexity is nearly reduced $|A_c|$ times compared with the original CMAC-RLS algorithm.

To illustrate the simplified algorithm derived above, we will carry out a computer simulation of the new scheme. The function we approximate is,

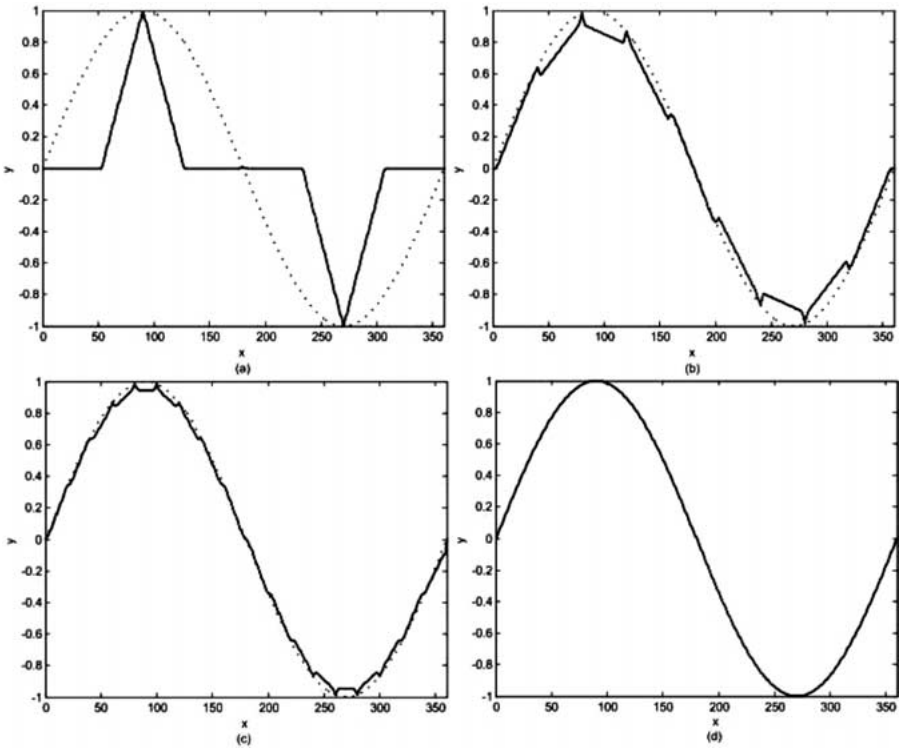

Figure 2. Local generalization ability of CMAC-RLS (a) $t = 90$, $RMSE = 0.489$, (b) $t = 40$, $RMSE = 0.082$, (c) $t = 20$, $RMSE = 0.033$, (d) $t = 1$, $RMSE = 1.202 \times 10^{-7}$.

$$y = \sin(2\pi x/360) \quad x \in [0, 360] \tag{28}$$

The parameters of CMAC are $Q = 360$; $|A_c| = 396$; $c = 18$.

Define t as the sampling interval. The simulation results are illustrated below:

From the simulations above, we can see that even if the number of samples is less than that of unknowns, CMAC still has good local generalization ability around the sample data. And it is obvious that the more evenly the samples scatter, the better the generalization ability is.

3.2. CMAC-RLS WITH HASH-CODING

It is mentioned in reference [21] that different hash-coding will affect the accuracy of modeling differently. Next we will show the effect of CMAC-RLS combined with hash-coding.

Similarly, the time complexity of CMAC-RLS with hash-coding is $O(|A_p|^3)$.

The address of conceptual memory ranges from 1 to $|A_c|$, and the address of physical memory ranges from 1 to $|A_p|$, then the hash-coding function $H(k)$ must satisfy

$$1 \leqslant H(k) \leqslant |A_p|, \qquad \forall 1 \leqslant k \leqslant |A_c|. \tag{29}$$

Hash-coding function is chosen the same as one in reference [21]:

$$H(k) = 1 + fix\left\{|A_p|\left[\left(\frac{F}{w}k\right) \bmod 1\right]\right\}, \quad w = 2^{30}, \quad F = 663608941 \tag{30}$$

The parameters are adopted as: $Q = 360$; $c = 18$; $|A_c| = 396$; $|A_p|$ is 107 or 200.

Compared with the simulations in reference [21], it is concluded that hash-coding affects the same on CMAC-RLS as on traditional CMAC-LMS. And hash-coding will reduce CMAC's approximation ability. Also, the smaller $|A_p|$ is, the worse the approximation ability is.

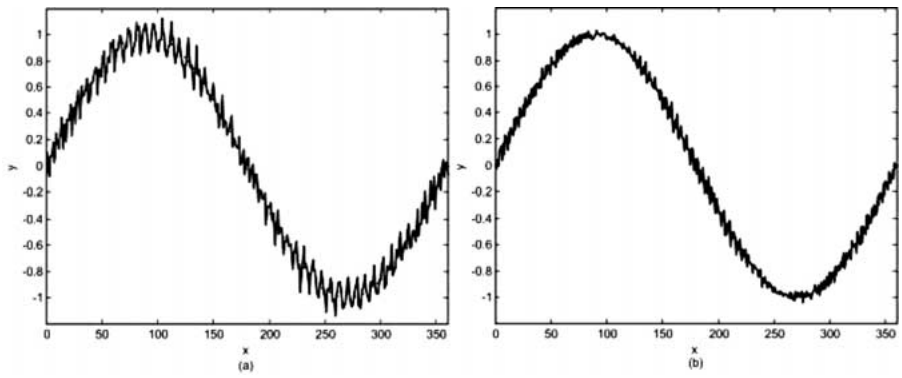

Figure 3. Effect of CMAC-RLS with hash-coding: (a) $|A_p| = 107$, $RMSE = 0.077$; (b) $|A_p| = 200$; $RMSE = 0.041$.

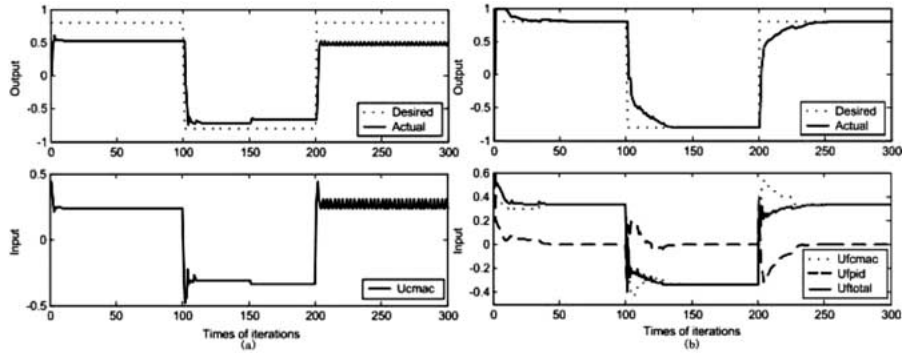

Figure 4. Comparison of the two controllers: (a) Performance of the CMAC inverse controller; (b) Performance of the CMAC-PID controller.

4. Implement CMAC-RLS on Nonlinear Control

In the previous sections, we have stated that CMAC-RLS is very suitable for modeling on line. Next we will examine this ability by applying it to control nonlinear plant. The plant is chosen as:

$$y(k+1) = 0.5y(k)^2u(k) + 0.5\sin[y(k)] + u(k) \tag{31}$$

$$\text{s.t.} \quad -1 \leqslant u \leqslant 1; \quad -1 \leqslant y \leqslant 1.$$

First, we conduct an experiment to generate 100 training input/output pairs with excitation signal which is random with uniform amplitude distribution in the interval $[-0.5\ 0.5]$ and at the same time train CMAC using CMAC-RLS on line. Subsequently we will begin to model and control simultaneously on line.

Because the coarse quantization will result in quantization errors, model inaccuracy is supposed to exist. It is shown in the left panel of Figure 4 that if only the CMAC inverse controller is used, it merely adjusts the output near the reference and the static error exists. Sometimes the output may have small-amplitude oscillation near the reference and the controller also exhibits a oscillatory state.

It is known that feedback control can reduce the model inaccuracy, so a fixed parameter proportional-integral-derivative (PID) feedback controller is incorporated with the CMAC inverse controller in parallel, which is designed as a CMAC-PID [7] controller. And in order to reduce the small-amplitude oscillation, first order low-pass filters are added to filter both the control input of the CMAC inverse controller and the control input of the PID controller. The final controller design is depicted in Figure 5.

The first order low-pass filters are described as

$$uf_{\text{cmac}}(k) = (1 - fp) \times u_{\text{cmac}}(k) + fp \times uf_{\text{cmac}}(k-1) \tag{32}$$

$$uf_{\text{pid}}(k) = (1 - fp) \times u_{\text{pid}}(k) + fp \times uf_{\text{pid}}(k-1) \tag{33}$$

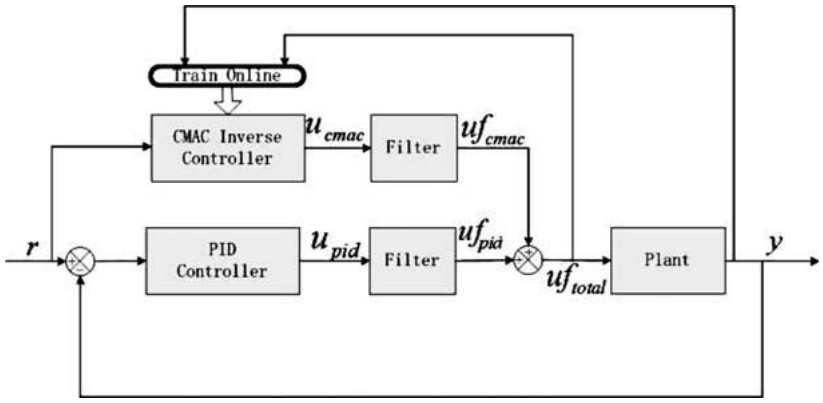

Figure 5. Design of CMAC-PID controller.

The parameters are adopted as: $Q = 30$; $c = 6$; $P_c = 2$, $I_c = 0.3$; $D_c = 0$; $fp = 0.7$.

The performance of the CMAC-PID controller is illustrated in the right panel of Figure 4, and we can see that it tracks the square-wave reference very well. The static error is eliminated and the oscillation vanishes. In the steady process we can see that the control signal of the PID controller almost reaches zero, whereas it can not be removed, because the CMAC inverse controller merely roughly adjust the output as displayed in the left panel of Figure 4, and the PID controller will adjust the output precisely. The CMAC-PID controller demonstrates a significantly better performance than the CMAC inverse controller alone.

In industrial engineering, the constant disturbance is often used to act on a plant to examine the controller's robustness. Consequently, at the 200th step we add a constant disturbance which is 0.1 to the output of the plant. It is displayed in Figure 6 that once the disturbance is added, the PID controller responses immediately. At the beginning the individual contribution to the total control signal from PID controller

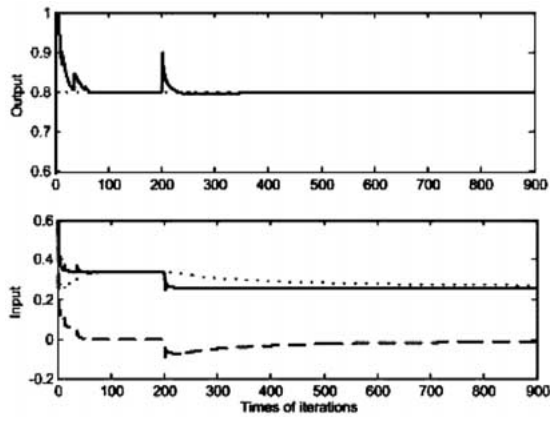

Figure 6. Effect if the constant disturbance is added.

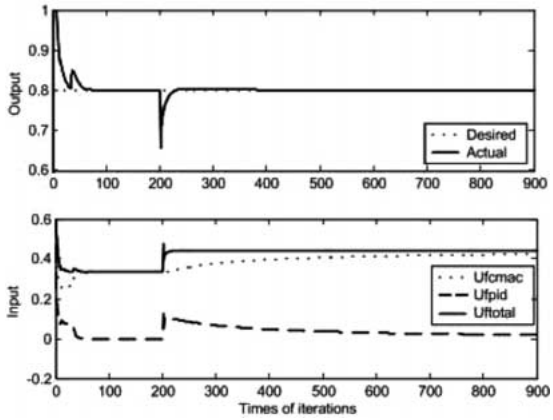

Figure 7. Effect if the plant varies with time.

is more. After a period of 50 seconds or so, the reference tracking is nearly achieved. But subsequently, CMAC is still trained on line to adapt to the new model. It can be seen that uf_{cmac} still decreases and uf_{pid} increases gradually, whereas their sum does not change so that the output remain unchanged. Generally speaking, this controller has good capability of rejecting external disturbances.

In order to examine the conroller's adaptive ability, we apply the neruo-PID controller to control nonlinear time-varying plant. At the 200^{th} step the plant model changes from (31) to

$$y(k+1) = 0.5y(k)^2u(k) + 0.3\sin[y(k)] + u(k) \tag{34}$$

Figure 7 shows the similar occurrence as illustrated in Figure 6. At the 200th step, uf_{pid} changes first. And after that moment, the weights of CMAC are updated continuously on line to adapt to the new plant model gradually. So we can see that uf_{cmac} increases and uf_{pid} decreases. But after the reference is tracked, their sum, uf_{total} does not change and the output still track the reference very well. This further demonstrates the good adaptive ability of the CMAC-PID controller.

It is shown both in Figures 6 and 7, when the plant model varies due to something unpredicted, the CMAC-PID controller can respond quickly to track the desired output. Just considering the internal interaction of the CMAC-PID controller, we can find that the PID controller provide time for CMAC to be trained on line. And accompanying with the training of CMAC, CMAC continuously approximates the new plant model; the effect of CMAC inverse controller is enhancing and the effect of the PID controller is reduced gradually. This also shows the advantage of the integration of the PID controller and the CMAC inverse controller.

In addition, this controller has another advantage that CMAC only requires coarse quantization so that the time complexity will be reduced.

We carry out all the simulations in Matlab on a PC whose CPU is Pentium 4 and RAM is 256M, and find out that the period for each iteration of RLS is about 1.5 seconds.

5. Conclusion

CMAC is a neural network that has good ability in local generalization, and RLS is an algorithm which can guarantee global optimum. Implementing RLS to update the weights of CMAC, we present a convergent and adaptive algorithm, CMAC-RLS. And the results obtained also give us a better understanding of CMAC. Then we use CMAC inverse controller as a rough adjustment controller, PID as a refined adjustment controller to control nonlinear system. The simulations results testify that the proposed CMAC-PID controller is a robust and adaptive controller.

Acknowledgement

This work is supported by the national '985'high-level university fund, the youth fund of USTC and Hefei key science and technology plan.

References

1. Kolcz, A. and Allinson, N. M.: Application of the CMAC input encoding scheme in the N-tuple approximation network, *IEE Proc. Comput. Digital Techniques*, **141** (1994), 177–183.
2. He, C. Xu, L. and Zhang, Y.: Learning convergence of CMAC algorithm, *Neural Processing Letters* **14** (2001), 61–74.
3. Chiang, C.-T. and Lin, C.-S.: CMAC with general basis functions, *Neural Networks*, **9** (1996), 1199–1211.
4. Lin, C.-S. and Chiang, C.-T.: Learning convergence of CMAC technique, *IEEE Trans. on Neural Networks*, **8** (1997), 1281–1292.
5. Albus, J. S.: A new approach to manipulator control: The cerebellar model articulation controller (CMAC), *Trans. ASME, J. Dyanmic Syst., Meas. Contr.*, **97** (1975), 220–227.
6. Albus, J. S.: Data storage in the cerebellar model articulation controller(CMAC), *Trans. ASME J. Dynamics Syst. Means. Contr.*, **97** (1975), 228–233.
7. Chang, G.-C., Lub, J.-J., Liao, G.-D., Lai, J.-S., Cheng, C.-K., Kuo, B.-L. and Kuo, T.-S.: A neuro-control system for the knee joint position control with quadriceps stimulation, *IEEE Trans. Rehab. Eng.*, **5** (1997), 2–11.
8. Ker, J.-S., Kuo, Y.-H. and Liu, B.-D.: Hardware realization of higher-order CMAC model for color calibration, *Proc. of IEEE International Conference on Neural Networks*, Perth, WA, Austrialia (11/27/1995–12/01/1995), (1656–1661).
9. Koo, K.-M. and Kim, J.-H.: CMAC based control of nonlinear mechanical system, *Proc. of the 1996 IEEE IECON 22nd International Conference on Industrial Electronics, Control, and Instrumentation, Taipei*, Taiwan, Aug 5–10, (1996), 1954–1959.
10. Ljung, L. and Soderstrom, T.: *Theory and Practice of Recursive Identification*, The MIT Press, 1983, pp 16–21.
11. Liu, H., Xu, X. and Zhang, Z.: An improved CMAC neural network algorithm, *Acta Automatica Sinica*, **23** (1997), 482–488.

12. Parks, P. C. and Militzer, J.: Convergence properties of associative memory storage for learning control system, *Automation and Remote Control*, **50** (1989), 254–286.
13. Lane, S. H. Handelman, D. A. and Gelfand, J. J.: Theory and development of higher-order CMAC neural networks, *IEEE Control Systems Magazine*, **12** (1992), 23–30.
14. Manglevhedakar, S.: An adaptive hierarchical model for computer vision, Thesis, Louisiana State UNiv. 1986.
15. Miller, W. T., Latham, P. J. and Scalera, S. M.: Bipedal Gait Adaptation for Walking with Dynamic Balance, *Proc of the 1991 American Controls Conference, Boston, MA*, **2** (1991), 1603–1608.
16. Miller, W. T., Glanz, F. H. and Kraft, L. G.: Application of a general learning algorithm to the control of robotic manipulators, *Int. J. of Robotics Research*, **6** (1987), 84–98.
17. Miller, W. T.: Real-time application of neural networks for sensor-based control of robots with vision, *IEEE Transactions on Systems, Man and Cybernetics*, **19** (1989), 825–831.
18. Miller, W. T., Hewes, R. P., Glanz, F. H. and Kraft, L. G.: Real-time dynamic control of an industrial manipulator using a neural-network-based learning controller, *IEEE Trans. on robotics and automation*, **6** (1990), 1–9.
19. Wu, X.: *Matrix Theory*, Tongji Universitiy Press, 1994.
20. Wong, Y.-F. and Siders, A.: Learning convergence in the cerebellar model articulation controller, *IEEE Trans. on Neural Networks*, **3**, (1992), 115–121.
21. Wang, Z.-Q. Schiano, J. L. and Ginsberg, M.: Hash-Coding in CMAC Neural Networks, *Proc. of IEEE International Conference on Neural Networks*, Washington, DC, USA, June 3–6 (1996), 1698–1703.