A CMAC with Content Addressable Memory

Yuan-Pao Hsu, Kao-Shing Hwang, and Jinn-Shyan Wang
Department of Electrical Engineering
National Chung Cheng University
Chia-Yi, Taiwan
E-mail : hsuyp@robot.ee.ccu.edu.tw

Abstract

A new design scheme of Cerebellar Model Articular Controller (CMAC) is presented in this article. The controller is designed with a content addressable memory (CAM) to replace the function of hash-coding method, which consumes more computation efforts, in the traditional CMAC for memory space consideration. Therefore, the address mapping of the proposed CMAC is comparatively different from the hash-coding method. The CAM, which is capable of fast comparison, can immediately determine in parallel if there is any akin data in memory as the input data presents. If a match doesn't occur after comparison, the activated address is then stored in a vacant cell of the CAM indexed by a circular incremental pointer. The memory collision should be able to be avoided unless the memory is all occupied. Thereby, memory utilization rate can be improved to 100%. Meanwhile, control noise can be suppressed significantly. In content addressing, the associated mask function, which is triggered while the CAM is full, can decrease the probability of collision and improve control performance. Simulation results of function approximation and truck backer-upper control indicate the proposed CMAC is explicitly superior to the conventional CMACs.

1. Introduction

Cerebella Model Articulation Controller (CMAC), originally proposed by James Albus [1], is the architecture of a neural network. A CMAC fundamentally estimates the desired output by taking input states as an index to refer to a look-up table where the synaptic weights are addressed and stored. The relationship between input and output can be represented approximately by a CMAC, if addressable weights are updated evolutionarily by a decent algorithm. Functionally, a CMAC is defined by a series of mappings as shown in Fig. 1.

$$S \to M \to C \to P \to O \quad ,$$

where S, M, C, P, and O stand for the input vector, encoding, the conceptual memory, the actual memory, and the output respectively. The overall mapping $S \to O$ can be presented in the form $O=h(S)$. The capability of learning and adaptability of a CMAC is provided by the memory addressing algorithm. It has been widely used in robot control [2], color correction [3], and system model approximation [4] as a result of the properties of local generalization, rapid computation, function approximation, and output superposition.

The hardware implementation of CMAC is hard to be realized since its conceptual memory theoretically needs a huge space to address the encoded input information. Hash coding algorithms [5] are generally applied to reduce the space into a more reasonable scale. However, this approach has some drawbacks [6]. First, collisions always occur frequently during mapping from a wide input domain to a relatively small field. For instance, the cells of c_3 and a_2, as illustrated in Fig. 1, project onto a memory cell p_1 simultaneously. Some algorithms [5] have been introduced to achieve low collision rate and high memory utilization. However, in these improved algorithms some cells are still never visited thoroughly. In other words, they hardly achieve 100% memory utilization rate. Second, while collisions occur, hash coding algorithms regard the weights stored in the clashed memories as a validated one that responds to the temporal input though. Therefore, most of CMACs with hash coding algorithms may retrieve, output, and update an irrelevant data in terms of mapping crash. From the viewpoint of system control, this manner could cause CMACs interfered by so-called control noises.

The objective of this article is to propose a hardware realization of the CMAC function. The proposed architecture of the CMAC is called CAM-based CMAC (CCMAC), which utilizes an embedded content addressable memory (CAM) to execute the functions of efficient encoding and memory association. The mechanism of CAM is mainly used to replace the function of hash coding in addressing. By the scheme, the new design should have better performance in memory utilization and control noise alleviation.

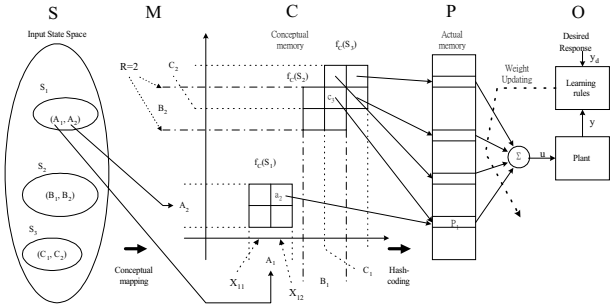

Fig. 1 Theoretical diagram of CMAC

The paper is organized as follows. Section 2 briefly introduces the theory of the conventional CMAC for the purpose of comparison with the proposed design. Section 3 illustrates the functional architecture of the proposed CCMAC. The simulations on function approximation of a sinusoidal function and the backer-upper control of a truck are conducted in section 4. The comparison between the proposed CCMAC and the conventional CMAC is discussed in this section also. Finally, discussion and conclusions are drawn in the last section.

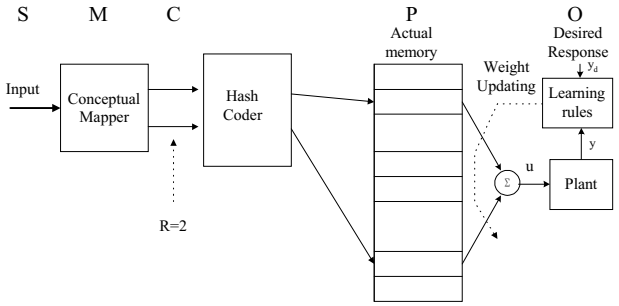

Fig. 2 The functional block of HCMAC

2. The Conventional CMAC

The input vector S is the state feedback from the plant or environment, and is mapped to the conceptual memory C by means of conceptual mapping mechanism M, as illustrated in Fig 1. This conceptual mapping mechanism can be described as Eq. (1)

$$f_c(S_j) = X_{ji} = [\text{int}(\frac{S_j + R - i - 1}{R})] \times R + i, \quad (1)$$

where S_j is the input, and R is the size of the receptive field (R = 2 in the case of Fig 1). i = 0, 1, 2,…, R-1 is an index number of the activated address, and X_{ji} is the activated address mapped from S to C. If the distance among the input vectors is relatively short, then there should be some addresses overlapping as the case of $f_c(S_2)$ and $f_c(S_3)$ indicated in Fig. 1. The situation is referred as the generalization problem. The problem can also be explained as: *if input vectors S_2 and S_3 are similar, and there have been adequately tuned weights stored in the memory with respect to S_2 already, then S_3 could refer to these overlapped addresses associated with S_2 to get more suitable weights for producing output before updating.* The functional block diagram of a conventional CMAC with hash coding function, called as HCMAC, is depicted in Fig. 2.

Without a decent encoding function, the size of conceptual memory would be unreasonably large in terms of hardware implementation. For example, if an input vector is encoded in 20 bits, then the size of the conceptual memory needed can be 1M bytes. Therefore, such conceptual memory is not practical to be realized in hardware. Since a hash coder can map the activated cells onto a physical memory in a reasonable size, the coder scheme is always adopted in various implementations

The output of the controller is computed by the summation of the values (weights) stored in the indexed physical memory cells. The learning process, where weights are adjusted based on the desired and actual outputs, is taken thereby. The weights of the indexed cells W_i is updated according to the delta-type learning rule [7] by the scale of

$$\Delta W_i = \gamma \frac{y_{di} - y_i}{R}, \quad (2)$$

where y_i is the actual output of the *ith* control cycle; y_{di} is the desired output of the *ith* control cycle; γ is the learning rate which is in the range of (0, 1]; R is the number of the activated memory cells.

3. CMAC with Embedded CAM

The most significant characteristic of Content Addressable Memory is of fast matching. It can determine whether the addressing content resides in the memory or not very quickly. This function is very useful for the mapping function of a CMAC. Therefore, the proposed architecture of a CMAC is designed to utilize the merit of CAM in memory addressing. The functional block diagram of a CCMAC is shown in Fig. 3. Each cell of the CAM stores the activated addresses and their associated weights are kept in SRAM, respectively. The input of each control cycle introduces some activated addresses through a conceptual mapping function similar to Eq. (1). These addresses are immediately compared with the ones stored in the CAM. If a match occurs, the corresponding weight is read out, otherwise, these activated addresses is stored in vacant cells of the CAM indexed by an circular incremental pointer. If there is vacancy no more, the mask function is invoked alternatively. The superset of the activated addresses bits of which will be ignored bit by bit from the least significant bit (LSB) to the most significant bit (MSB) in matching process until a match appears or the limited bits have been masked. In this scenario, matching tolerance and searching range become larger and larger along with the number of the searching cycles. In the worst case, there might be no match during the whole masking-matching process. The address is then pushed in an occupied cell pointed by the circular pointer. Memory utilization rate can, with this mechanism, be improved to 100%. The noise is attenuated since the weight is reset to zero instead of inheriting the obsolete value while

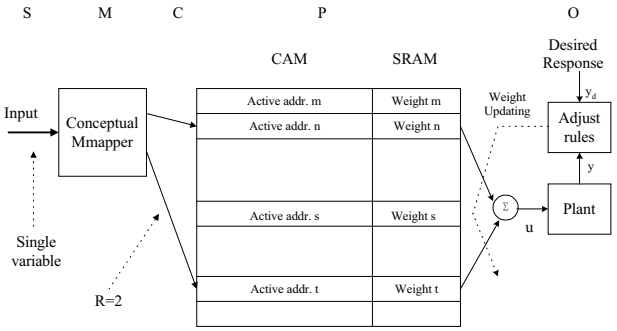

Fig. 3 CCMAC functional block diagram

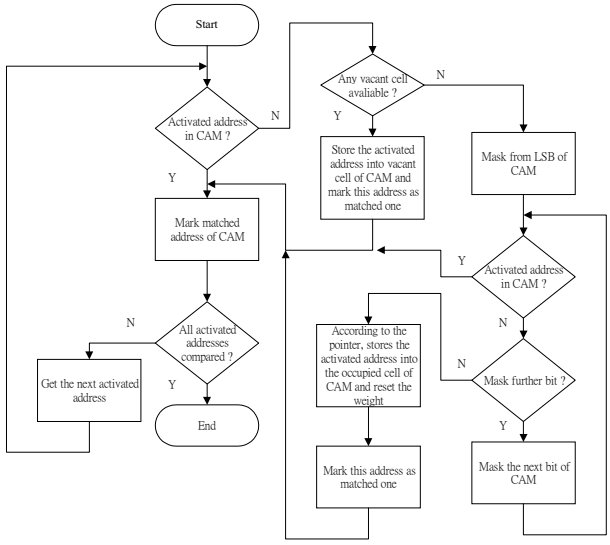

Fig 4. CCMAC mask function behavior flowchart

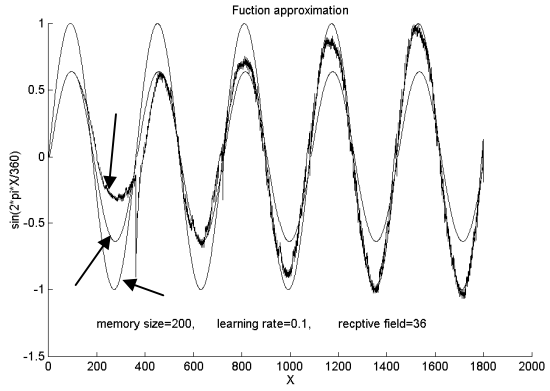

Fig. 5 CCMAC and HCMAC function approximation results, γ=0.1

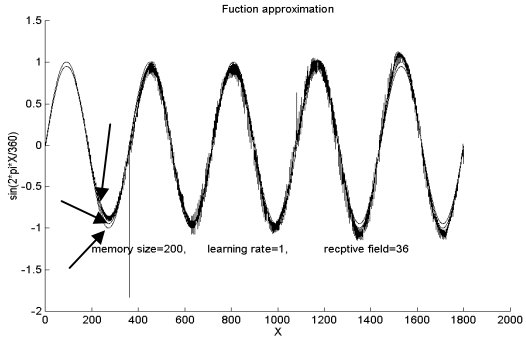

Fig. 6 CCMAC and HCMAC function approximation results, γ=1

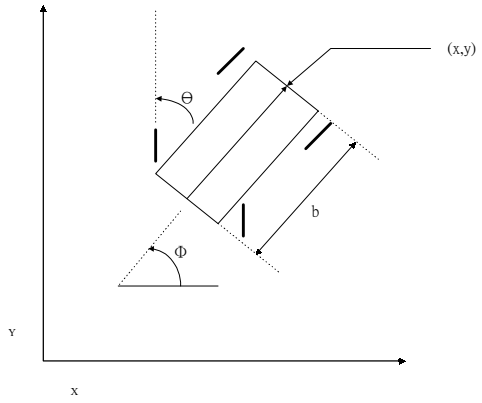

Fig. 7 Workspace for truck backer-upper control

the collision occurs. Fig 4 is the flowchart that the whole process be illustrated.

4. Performance Evaluation

Function approximation and truck backer-upper simulation are used to compare the control performance between HCMAC and CCMAC methods. The simulation results are investigated for verifying the practicability of CCMAC.

Case I. Function Approximation

Assuming that $y=sin(2\pi x/360)$ is the function to be approximated. x represents the input variable and varies integrally in the range [0, 360]. Receptive field R=36, conceptual memory C=395, actual memory P=200, and initial value of weight W=0 are the correlative parameters of the CMAC. Simulations are repeatedly performed five times while x varies from 0 to 359. Fibonacci hash-coding mapping algorithm [5] is, as listed in Eq. (3), adapted for the simulation of HCMAC.

$$H(x) = 1 + fix\left\{ P\left[\left[\frac{F}{G} x \right] \right] \mod 1 \right\}, \quad (3)$$

where F=663608941, G=2^{30}, x stands for activated address, actual memory P=200, mod 1 means only the fraction part of a value remained, *fix* means that only the integer part of a value is remained, and $H(x)$ is the final actual address mapped.

The approximation function of CCMAC seems worse than HCMAC when the learning rate is small, i.g. γ=0.1 as shown in Fig. 5. Whereas, the performance of CCMAC is apparently better than HCMAC if the learning rate increases, such as γ=1 the improved performance can be observed from Fig. 6. It is worthy to note that HCMAC produced obvious noises in both results. On the contrary, CCMAC demonstrates no noises, which appears as fluctuation along the trajectory of the simulated curve, in the results.

Case II. Truck Backer-Upper Control Simulation

The second simulation is a backer-upper truck [8]. Fig. 7 shows the configuration of the workspace. The angle ϕ between the truck tail and horizontal line and the distance x between the truck tail and the target line are the input variables to the neural network. θ the output variable is the angle between front wheel and the direction along the truck tail. Controlling the truck in a manner of backer-upper going toward the middle line until $\phi = 90^\circ$ is the goal of the control. The truck moving range is [0, 20]. Metrics y is not important here. So the controller has a sense of parking a truck along the roadside while there are no other trucks parking there. The dynamic equations are as follows [9]

$$x(t+1)=x(t)+d\cos(\phi(t)), \quad (4)$$

$$y(t+1)=y(t)+d\sin(\phi(t)), \quad (5)$$

$$\phi(t+1)=\phi(t)+\theta(t), \quad (6)$$

where d is moving distance between two successive samples. The simulated input variables ϕ and x are encoded in 17-bit and 9-bit wide respectively. Each input variable actives six addresses respectively. Then the addresses are combined to 36 active addresses. The size of the actual memory P is 1024. Output variable θ is represented in a format of 16-bit digital data. The system converts a digital output to an analog signal for controlling the turning angle of the truck's front wheel. A simple weight-updating algorithm similar to the delta-type learning rules [7] is applied to adjusting weight incrementally. The rules are listed as follows

$$w_i(k+1) = w_i(k) + \gamma \frac{(y_d - y) f_i(s)}{R}, \quad (7)$$

where y_d is the desired output; the term $f_i(s)$ determines which weights should be updated. Fig. 8, Fig. 9, and Fig. 10 are the simulation results that the truck started from different starting points, respectively. The moving trajectories of the control effect of CCMAC and HCMAC are plotted in the figures for comparison. In every control cycle, the state of the truck is marked as a short line with an arrow. The dot line with a solid arrow represents the desired state of the

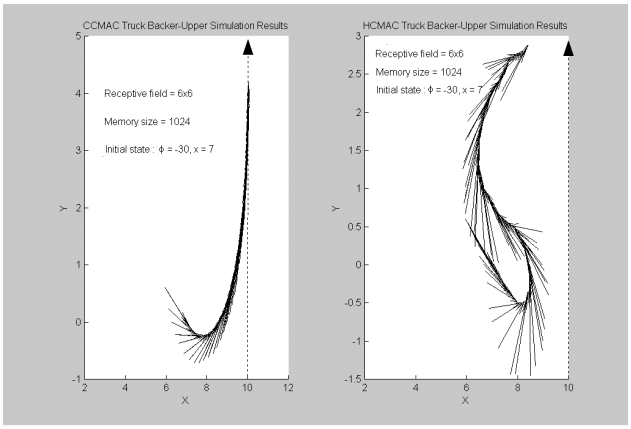

Fig. 8 Truck backer-upper simulation results, the truck starts from the point $\phi = -30^{\circ}$, $x = 7$. CCMAC controls the truck to reach the desired condition more smoothly than that of HCMAC does. The dot line with upward arrow stands for the desired truck state.

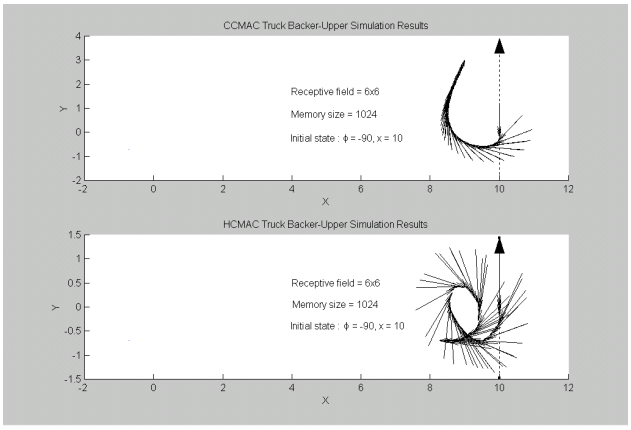

Fig. 9 Truck backer-upper simulation results from the starting point $\phi = -90^{\circ}$, $x = 10$, HCMAC controls the truck with a surfeit of output value to make the truck moving unstably, as indicated at the bottom figure. The dot line with upward arrow stands for the desired truck state.

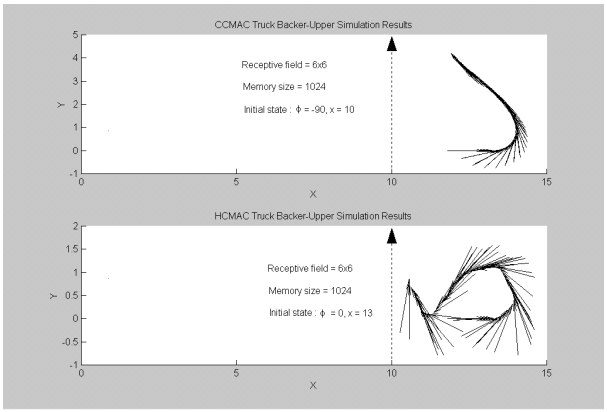

Fig. 10 Truck backer-upper simulation results from the starting point $\phi = 0^{\circ}$, $x = 13$, CCMAC controls the truck incrementally toward the desired state, conversely, HCMAC steers the truck by adopting inadequately weights to make the truck towards the undetermined direction. The dot line with upward arrow stands for the desired truck state.

truck under control.

In Fig. 8 the truck can get to the goal under both control methods, however, CCMAC gets smoother trajectory than HCMAC. The HCMAC controls the truck with fairly unsuitable output value as depicted in Fig. 9 and Fig. 10, so the truck is over forced towards the undetermined direction after only just tens of control signal had emitted. The successive control results are worse and are not depicted on the figures for the clarity of the moving trajectories. In addition with many other results are not shown here reveal obviously that the control effect of CCMACs is superior to HCMACs.

5. Discussion and Conclusion

The function approximation is a repetitive control job. Although the approximation effect of HCMAC is better than CCMAC with small learning rate, CCMAC still excels HCMAC apparently if the learning rate increases accordingly. Moreover, the control noise of HCMAC always appears but CCMAC does not have this drawback. CCMAC adopts circular incremental pointer, so the activated addresses and weights are stored where this pointer indexes. The pointer points back to the starting address when the CAM is filled. In such circumstance, if more activated addresses should be stored, CCMAC replaces the old address directly according to the pointer. The weight stored in this new address would be reset to zero. Consequently, the control value becomes slightly small if the number of new activated addresses is too many as indicated in Fig.5. This weakness is, as depicted in Fig. 6, improved by means of the adjustment of the learning rate. On the contrary, HCMAC refers to the weight stored previously while collision occurs. These values are not always so suitable for the new situation. Unavoidably, HCMAC causes control noise.

Nevertheless, the truck backer-upper control is a nonrepetitive control job. The relevant weights are adjusted gradually so as to approximately manipulate the truck toward the goal with correct direction. In the whole control process, the controller must emit the correct control value to the truck in each control step. CCMAC can achieve this requirement due to its less control noise generated in the control process. HCMAC, with high SNR, will introduce too large or too small signals to control the truck. The truck spends too much time to reach the goal even it is not out of control. In the viewpoint of hardware realization, the major differences between CCMAC and HCMAC are listed in Table 1. CCMAC need an extra CAM to store the activated address, while HCMAC needs multiplier and adder to map the activated address to the actual memory. CCMAC spends some gate delay time to process the mapping from the activated address to the CAM. As well, HCMAC consumes some time consumption on the hash-code mapping by the multiplier and adder. The more accuracy is the HCMAC performs the calculation, the more complexity of the multiplier is needed and the more time is consumed. In summary, CCMAC is faster than HAMAC. Under reasonable estimation, the cost of the hardware is not much difference between CCMAC and HCMAC.

As a whole, in this article we propose a new neural network architecture, CCMAC, where a content addressable memory (CAM) is embedded into the CMAC. The scheme

Table 1 Comparison between CCMAC and HCMAC

	CCMAC	HCMAC
Active address mapping	Via index pointer	Hash-coding
Hardware requirement	SRAM storing weight, CAM storing active address	SRAM storing weight, Multiplier Adder
Mapping latency	Depend on CAM matching time (fast)	Depend on multiplier operation time (slow)
Memory usability	100%	According to hash-coding algorithm (<100%)
Control noise	Hardly observed	High
Learning ability	Yes	Yes
Applications	Jobs need that low control noise	Can not handle jobs that need low control noise (e.g. truck backer-upper control)

can be used to replace the HCMAC with traditional hash-coding method. 100% of memory utilization and less control noise are the major advantages of CCMAC. CCMAC can handle repetitive and nonrepetitive control job as superior control speed.

6. References

[1] J. S. Albus, “A new approach to manipulator control: The cerebellar model articulation controller (CMAC),” Trans. ASME, J. Dynamic Syst. Meas., Contr., vol. 97, pp. 220-227, Sept. 1975.

[2] W. T. Miller, III, "Sensor-Based Control of Robotic Manipulators Using a General Learning Algorithm," IEEE Journal of Robotics and Automation, Vol. RA-s, No. 2, April 1987.

[3] R.-C. Wen, J.-S. Ker, Y.-H. Kuo, B.-D. Liu, and G.-W. Chang," A CMAC Neural Network Chip for Color Correction," Neural Networks, IEEE World Congress on Computational Intelligence, 1994 IEEE International Conference, Vol. 3 , Pp. 1943 -1948, 1994.

[4] W. T. Miller, III, R. T. Hewes, F. H. Glanz, and L. G. Kraft, III, "Real-Time Dynamic Control of an Industrial Manipulator Using a Neural-Network-Based Learning Controller," IEEE Transactions on Robotics and Automation, Vol. 1, February. 1990.

[5] Z. –Q. Wang, J. Schiano and M. Ginsberg, ”Hash-coding in CMAC Neural Networks,” proceedings of International Conference on Neural Networks, pp. 1698-1703, Washington, D. C., June 3-6, 1996.

[6] Y.-P. Hsu, K.-S. Hwang, C.-Y. Pao, and J.-S. Wang, "A New CMAC Neural Network Architecture and Its ASIC Realization," proceedings of Asia and South Pacific Design Automation Conference 2000, Yokohama, Japan, pp. 481-484, January 2000.

[7] J. M. Zurada, Introduction to Artificial Neural Systems, West Publishing Company, Singapore, 1992.

[8] L. –X. Wang and J. M. Mendel, “Generating Fuzzy Rules by Learning from Examples,” IEEE Transactions on systems, man, and cybernetics, vol. 22, no. 6, Nov./Dec. 1992.