Buletinul Ştiinţific al Universităţii "Politehnica" din Timişoara

Seria ELECTRONICĂ şi TELECOMUNICAŢII
TRANSACTIONS on ELECTRONICS and COMMUNICATIONS

Tom 49(63), Fascicola 1, 2004

Hardware implementation of CMAC based artificial network with process control application

Tihamér Sándor BRASSAI[1][2], László DÁVID[3], László BAKÓ[4][5]

Abstract –The cerebellar model articulation controller (CMAC) is often used in learning control and has become especially popular in the areas of robotics and control where the real-time capabilities of the network are of particular importance. In this paper a CMAC based adaptive controller software implementation and simulation are discussed with application in process control. The implementation process of the controller on digital reconfigurable hardware is also mentioned. Experimental results are given with controller software simulation results in a trajectory following application.
Keywords: CMAC, Neural networks, FPGA, Hardware implementation

I. INTRODUCTION

Great interest has been manifested lately in the utilization of adaptive modeling and control, based on biological structures and learning algorithms.
Control systems need to have high dynamic performance and robust behavior. These controllers are expected to cope with complex [1], uncertain and nonlinear dynamic processes. It is difficult to obtain a mathematical representation of uncertain and nonlinear dynamic processes that impose an intelligent modeling and control. ICs are generally self-organizing or adaptive and are able to cope with significant changes in the plant and its environment, while satisfying the control requirements. Intelligent modeling and control are advantageous if the learning schemes generate local continuous nonlinear mappings so that the desired function can be represented over local regions. The network's output is represented as an overlapping local mapping, each of them allowing local adaptation. These locally generalizing networks are universal approximation schemes as they can approximate any continuous nonlinear function to any desired degree of accuracy. A CMAC type controller was experimented in software and implemented in hardware. To achieve high control performance the weights were updated using the principle of minimal disturbance and not the standard gradient descendent rules.

II MODEL ARCHITECTURE

The learning module (Fig. 1) is used to model the process directly, receiving the applied control signal. The error between the model and the measured plant output is used as a feedback signal and these are passed to the learning controller (LC).

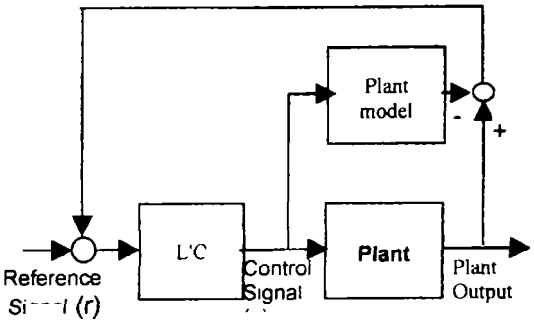


Fig. 1: Internal model control.

The used configurations represent an internal model control architecture as it is shown on Fig. 1 (LC-learning Controller, r-reference signal; u-control signal). Any associative type memory networks are generally used for functional approximation tasks. All of these networks can be represented as a three-layer system with:

- normalized input space
- basis functions
- weight vector and network output.

For the normalized input space a partitioning strategy was used (input lattice) based on optimal displacement tables from [2]. The basis functions are Gaussian type defined on normalized input space and their size and overlap determine how the network generalizes the input space.

[1] Teaching assistant at "SAPIENTIA" Hungarian University of Transylvania, P-ţa Trandafirilor nr. 61, Tg.-Mureş, Romania, tiha@ms.sapientia.ro
[2] PhD student at "TRANSILVANIA" University of Braşov, Romania
[3] Professor at "SAPIENTIA" Hungarian University of Transylvania, Tg.-Mureş, Romania
[4] Teaching assistant at "SAPIENTIA" Hungarian University of Transylvania, Tg.-Mureş, Romania
[5] PhD student at "TRANSILVANIA" University of Braşov, Romania

In the present paper a CMAC type AMN controller was used, built in an adaptive form. Three-dimensional normalized space is the receptive field of this ANN. The input space is with:

- *error of reference*, difference between the current position and reference position for the trajectory;
- *speed of current position*, difference in position in two neighbor time momentum;
- *acceleration for the current position* in order to define a lattice on the input space, a set of 3 known vectors is given, one known vector for each input axis.

These known values give the positions of the 2 dimensional planes. The set of all planes generates the lattice in the input space. It was incorporate a priori knowledge about this tacking problem into the network design with the position of knots for each axis.

The hidden layer is represented by a set of local basis functions wich are defined on the 3 dimensional rectangle bounded lattice. The number of non-zero basis function for an input vector (generalization parameter) is a constant and it was experimented to have the value from 4 to 10 (overlays) with 10-30 internal point for every input dimension. These values were experimented and the mentioned values gave enough accuracy in testing. The number of generalization parameter or number of overlays must be optimal because of on-line learning procedure. If the internal point is higher the learning is more local. For a less number of internal points the controller forgot more quickly the controlling information learnt in last phases.

The local generalization wich occurs in the CMAC is uniquely determined by the initial nonlinear mapping, as each basis function has associated with it a support, the number of overlays specifies the number of basis functions that contributes to the network output but determines the size of their supports. In order to achieve smooth local generalization the overlays are "space" relative to each other to have a uniform projection onto each of input axes. The sets of adaptive parameters (weight vector) are trained in order to achieve a desired behavior.

III. HARDWARE IMPLEMENTATION

The hardware implemented CMAC type controller was developed on a Spartan II XC2S50 FPGA reprogrammable digital hardware with 50kGate wich had to satisfy the following properties:

- p v y o r ce the nt ll. parameters, the basis function, the initial weight values, the trajectory reference signal generated on a PC
- to implement the learning algorithm
- to elaborate the control signal from the reference signal and the plant current position
- to offer the possibility to save the network weights for a later utilization
- We proposed the following schematic bloc diagram wich contains the necessary elements to satisfy the needs wich was mentioned before.

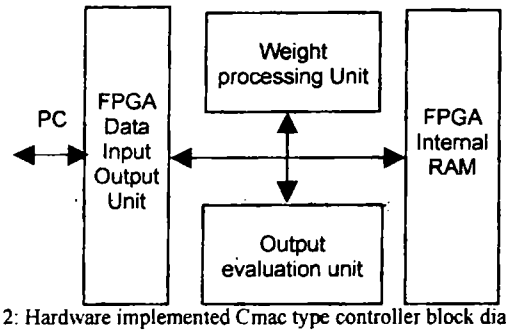


Fig. 2: Hardware implemented Cmac type controller block diagram

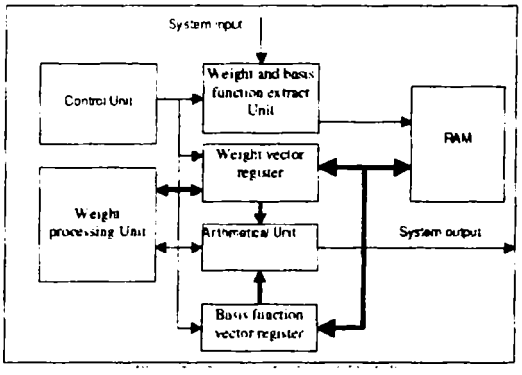
The Cmac type hardware implemented controller is composed of the following components:

- FPGA Data input-output unit which helps in data transfer from and to PC
- weight processing unit in which the learning algorithm is implemented (weight correcting algorithm)
- output evaluation unit. (Control evaluation unit) In this module the CMAC controller is implemented and this module elaborates the control signal.

Controller internal logic implementation:

The weight and basis function extraction unit extracts the weights vector and Basis function value vector from the RAM memory - according to the inputs – and stores them in the Weight vector and basis function value vector registers. The arithmetic unit has the role of multiplying the weights with the corresponding basis function values. It calculates the controller output by summing these intermediate values. The weight processing unit is responsible for weight correction (learning). All these units are synchronized by the Control Unit.

Fig 3. Controller internal logic implementation



At the other side, on the PC te following software modules were implemented:

- a communication module for data transfer from PC to FPGA
- Simulated process (which contains the system model , and trajectory generator)
- CMAC implementation initialization

The FPGA reprogrammable digital hardware is connected to the PC through the parallel port. To test

the hardware controller we used a simulated process implemented on PC in C++, using a discrete model identification.

IV. THE SIMULATION PHASES

When designing the CMAC based controller on the PC we have to define the following parameters:

- number of inputs,
- number of overlays,
- number of internal points,
- the inputs limit,
- the utilized basis function.

The basis function was built on the PC and calculated in a few points. The basis function values were transferred in the FPGA RAM. We chose to implement the basis function in this form because it gave us more possibilities to evaluate the controller behavior with different basis functions.

After the CMAC type controller has been designed, the controller structure will be transferred to the FPGA over the PC parallel port, with special software package provided by hardware manufacturer. At this stage the Output evaluation module, and weight processing unit are built in the FPGA, while the weight vector also has to be transferred using the Data Input Output Unit and stored in RAM module on the FPGA prototyping board.

The trajectory generator program runs on the PC, program wich was utilized in software simulated CMAC based controller with minor modifications. In fact at this point the controller is initialized, and is read to start the simulation.

The trajectory generator calculates the next reference point. From this point and from the current position of the DC motor an error reference, the speed of current position, and the acceleration of the current position, are calculated and then transferred over the data output-input unit to the Output evaluation unit. The Output evaluation unit produces the next command wich is applied over the Data Input output units to the simulated Process. The weight processing unit, in wich is implemented the learning algotithm, corrects the weights corresponding to inputs in that moment, using the error reference.

The error reference will define the magnitude, and direction which will be used in weight correcting and learning process. It has to be mentioned, that VHDL language and the Webpack6.1 software were used in hardware implementation, provided by the FPGA manufacturer. As an error detection tool we used a 34 channel logic analyzer, which proved to be invaluable help throughout the hole design process.

One of the problems that had occurred during the hardware implementation was the fact, that we were running out of available logic gates when the number of overlays and internal points grew higher, forcing us to reoptimize the control unit and the weight processing unit. In the future we propose to change the simulated process to a real process, in which case the FPGA will be directly connected to the real process. The controller will be programmed from a PC, and the weight and controller structure can be saved in a FPGA internal FLASH memory, while the controller can be disconnected from PC.

In the following section we present several experimental results obtained with the software implemented CMAC type controller.

V. EXPERIMENTS

In these experiments three different type of generated trajectories were used: a random type, a sine-curve and a triangular type reference trajectory. The CMAC controllers for two different actuators were used in sequence (serial command), each with its prescribed component of final trajectory.

The used CMAC controller has an input vector with three components, five overlays for an accurate generalization, the universe of every input is devised into fifteen divisions, and the optimal overlay displacement vector was (1,2,4) [2]. The basis functions were of Gaussian [4,5] type in every experiment. The command signal is limited and the experiments were made with different learning rate (lr) and different discretization period for the control loop. In the following figures we present experimental results, where the *thick* line represents the difference between the prescribed trajectory and the learned following path. The *thin* line represents the prescribed trajectory and the *dashed* line is the learned following path.

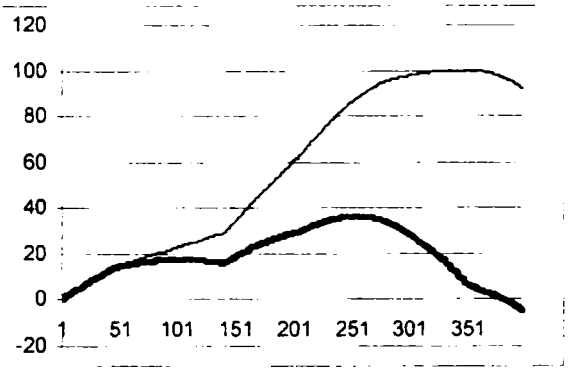


Fig. 4. Random type sine curve at the beginning of the simulation lr=0.01

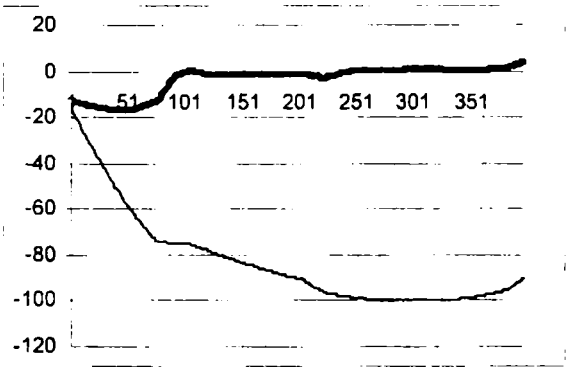


Fig 5. Random type sine curve after 1000 iteration of the simulation

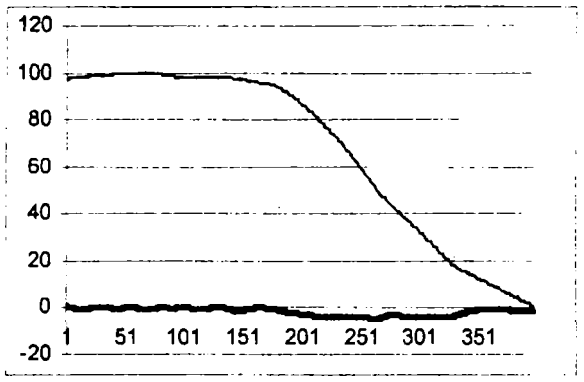


Fig 6. Random type sine curve after 4000 iteration of the simulation lr=0.01

Fig. 4,5,6 were extracted from the same simulation, and they show how the error between the reference trajectory and the followed trajectory decreases. Immediately after the start of the simulation (Fig. 4) the error is huge, because the CMAC type controller doesn't have an information about controlled process. After 1000 and 4000 iteration it is easily visible how the follow error decreases.

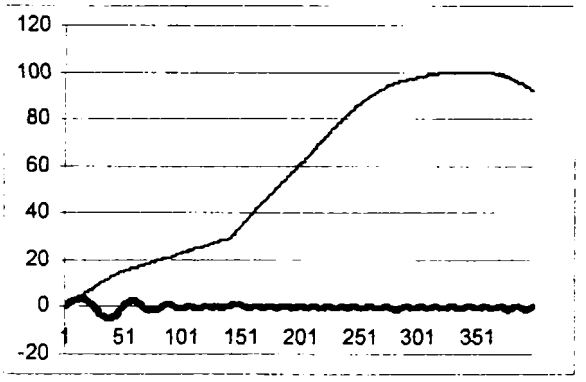


Fig 7. Random type sine curve at the beginning of the simulation lr=0.9

In both Fig. 4 and Fig. 7 the simulation it's presented from the start, the only difference between the two are the different learning rate values. One can easily observe, that in Fig. 7, where a bigger learning rate (lr=0.9) has been applied the trajectory following is better after a few iteration then in Fig. 4 with lr=0.01.

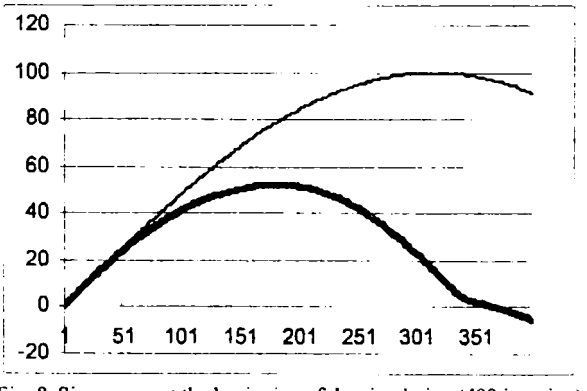


Fig 8. Sine curve at the beginning of the simulation (400 iteration) lr=0.01

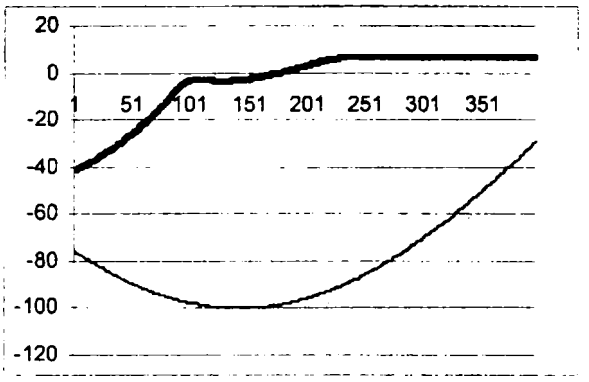


Fig. 9. Sine curve at the beginning of the simulation (2000 iteration) lr=0.01

Fig. 9. show a simulation result of 2000 iterations when the reference trajectory was a sine curve. Approximately after 1000 iterations the system can follow the reference signal with a good accuracy.

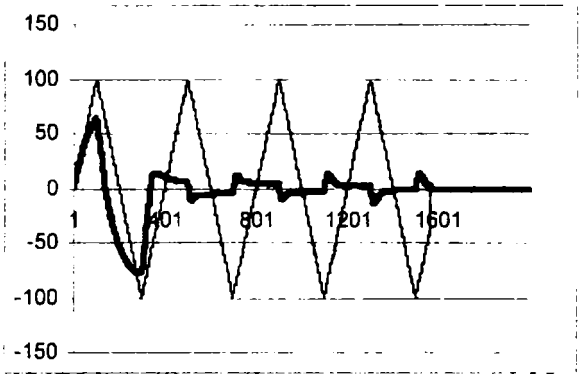


Fig 10. Triangular reference at the beginning of the simulation (2000 iteration) lr=0.01

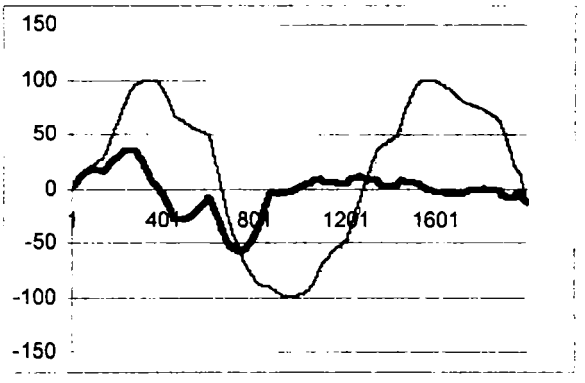


Fig. 11. Random type sine curve at the beginning of the simulation (4000 iteration) lr=0.9

Fig. 12 presents a sine curve reference. Because the learning rate was small, the follow error decreases slowly. In the next figure, the difference is, that we have a triangular reference signal a small learning rate at beginning, then - during the time of the simulation - the learning rate was modified from 0.01 to 0.9, fact wich can be seen in the last part of simulation.

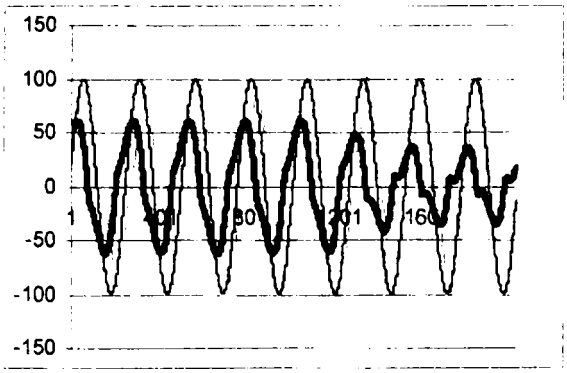


Fig 12. Sine curve reference simulation (2000 iteration) lr=0.01

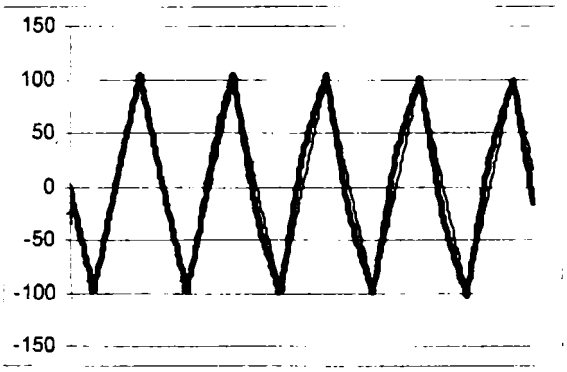


Fig. 13. Triangular reference simulation (2000 iteration) (lr=0.01 at beginning, lr=0.09 at finish)

Fig. 3-12 show different measured situations, comparing the prescribed trajectory and the actual trajectory with CMAC controller. It is necessary to mention that the CMAC system had no any initial information and the on-line training process assured the following of the prescribed path with an acceptable dynamic.

The weights of CM C ne works were ini ialized to random values at the beginning of every test. As it is known, the learning process is local (not every weight is modified at each step), and in this experiment the learning rate coefficient was modified and an extra module type coefficient was introduced in relation of weight update. The time axis for Fig. 4-9 represent a 400 iteration interval in simulations and Fig 10-13. represent 2000 iterations. The trajectory following performance is strongly dependent on the learning rate *(lr)* and the weight updating. The system has good performance in the situation of load disturbance. The whole software package was elaborated in C++.

REFERENCES

[1] Junhong, N. Derek, A.L., *Fuzzy-Neural Control, Prentice Hall*, 1995

[2] Brown, M. Harris C, *Neurofuzzy Adaptive Modeling and Control*, Prentice Hall, 1994

[3] Astrom, K.J. Wittenmark, B. *Computer Controlled Systems*, Prentice Hall, 1990

[4] Horváth Gábor, *Neuralis hálozatok és műszaki alkalmazások*, Műegye..i ..iadó, _udap..., 1.. 8

[5] Ching-Tsan Chiang, *CMAC with General Basis Functions*, Neural Networks, Vol.9 No.7, 1996, pp.1199-1211

[6] T.P. Trappenberg, *Fundamentals of Computational Neuroscience*, Oxford University Press, 2002.