Proceedings of the 17th World Congress
The International Federation of Automatic Control
Seoul, Korea, July 6-11, 2008

# Redundant Robot Kinematics Control with HCMAC Neural Network Manipulability Enhancement

**V. Řikovský*. Š. Kozák****

*Institute of Control and Industrial Informatics, Slovak University of Technology, Ilkovičova 3, 812 19 Bratislava, Slovakia (e-mail:vrikovsky@googlemail.com).
**Institute of Applied Informatics, Faculty of Informatics and Information Technology, Slovak University of Technology Ilkovičova 3, 812 19 Bratislava, Slovakia (e-mail: stefan.kozak@stuba.sk).

**Abstract:** The kinematics problems of redundant robots have been investigated for many years. A plenty of different robot redundancy usages were successfully implemented. In practice, e.g. redundant robot manipulability improvement, and robot obstacle avoidance are commonly used. Conventional methods use the manipulability gradient computation to improve the end-effector manipulability. However, the computational effort of this approach brings about many difficulties in real-time control when used with plenty of constraints. Recently, neural networks have found wide application in robotics, because of their feature to learn any complicated system model. This paper deals with new numerically efficient procedures and an application of HCMAC (Hierarchical Cerebellar Model Arithmetic Controller) neural network for manipulability gradient computation and consecutive robot control improvement. The conventional CMAC can be viewed as a basis function network (BFN) with supervised learning well-performing in the terms of its fast learning speed and local generalization capability for approximating nonlinear functions. Nevertheless, due to the redundant robot high-dimensional function approximation the conventional CMAC is enormously memory-demanding. HCMAC indeed has low memory requirements and very good learning ability. Furthermore, it shows a better performance compared with the conventional approach and conventional CMAC.

## 1. INTRODUCTION

Robotic manipulators are nowadays important devices in many different industries. The control of manipulator involves trajectory planning, inverse kinematics and inverse dynamics. This paper deals with the kinematics control of redundant manipulator. Kinematical redundant robots are of special interest due to their redundancy that can be used for additional tasks as avoiding obstacles, singularities etc.

The inverse kinematics of redundant robots can be solved in many ways. (Crane and Duffy, 1998) One of the main widely used algorithms is the Jacobian Pseudoinverse algorithm. This algorithm allows the manipulator to satisfy the additional constraints through mapping the corresponding velocities to the null space motions, while the end-effector tracks the desired trajectory. One of the additional constraints can be the manipulability. This manipulability measure and its improvement using HCMAC neural network is the focus of this paper.

CMAC neural-network was first proposed by J. Albus in 1975 (Albus, 1975). With its fast learning, good generalization capability and easy hardware implementation, CMAC has been applied in many real-world applications, e.g. robot control (Hu et al,1999), signal processing (Glany and Duffy, 1998), pattern recognition (Miller and Glanz, 1990) etc. Previous CMAC studies have focused mainly on how to develop CMAC learning algorithms (Cotter and Guillerm, 1991), improve the CMAC topology structure (Lane *et al*, 1992) and select learning parameters (Lin and Li, 2000). The convergence property has also received considerable attention, when it was proved that CMAC's learning always converges with arbitrary accuracy on any training data set. Further it was proved that CMAC learning results in a least square error if the number of iterations approaches infinity and the learning rate approaches zero. When the system requires derivatives of input and output variables, new nonconstant differentiable Gaussian basis function was applied to CMAC. Beside that (Lane *et al.*, 1992) developed a higher order CMAC neural network by using B-Spline receptive field function in conjunction with a general CMAC weight addressing scheme.

Nevertheless, the Albus's CMAC model has two major limitations: enormous memory requirements when dealing with high-dimensional problems, and difficulties in selecting the memory structure parameters (Lin and Kim, 1995). The fast memory size requirements limit the CMAC application fields for real-world implementations. Problem of memory requirements can be solved with hash-coding. It reduces the required memory in the CMAC technique and works well for some problems. This method associates several hypercubes to the same weight, when the complete input space is not

involved in learning. In such a case, although the memory requirements can be reduced, the convergence may be frustrated, i.e. it can lead to divergence. This may affect the speed of convergence and degenerate the behavior of convergence.

To reduce enormous memory requirements (Lin and Li, 2000) presented a unique learning structure composed of small two-dimensional CMACs to solve high-dimensional problems. However, their proposed structure has its limitations. To get a good learning capability for nonlinear functions, many parameters are to be obtained heuristically. This results to high dependence on parameter specification, and if the architecture is not properly considered the network does not work well.

In this paper, the HCMAC (Hierarchical CMAC) neural network for modeling the manipulability of redundant robot is proposed. This type of neural network can effectively overcome the problem of enormous memory requirement in the original CMAC model, because the new structure can partition high-dimensional problems into several manageable two-dimensional subproblems. The gradient-descent learning rule to train the proposed HCMAC model is presented as well.

The experiments with manipulability measure modeling using CMAC neural network showed that this neural network can faster compute the manipulability gradient. It means that the computation time for one robot configuration is much smaller than the conventional method based on nonlinear optimization. Thus, the CMAC neural network can substitute the original computation even sparing some computation time. The only disadvantage of CMAC neural network model is the need for hash-coding for the redundant robot high-dimensional function. Also the memory requirement to cover the whole input space would be too high for such a high-dimensional function. To solve these problems the new HCMAC neural network was chosen.

This paper is organized in 5 sections. Section 2 describes the inverse kinematics and manipulability measure. In Section 3 the structure of CMAC and HCMAC neural networks are shown. Section 4 presents verification results of modelling manipulability gradient on a 7 DOF redundant manipulator model using these neural networks. The last Section 5 summarizes the achieved results.

## 2. REDUNDANT ROBOT KINEMATICS

### 2.1 Pseudoinverse kinematics

For a kinematic redundant manipulator, the end-effector velocity is a function of the joint velocities which is expressed as follows

$$\dot{x}(t) = J(q(t))\dot{q}(t)\,, \tag{1}$$

where $\dot{x}(t)$ is the $m\times 1$ end-effector velocity, $\dot{q}(t)$ is the $n\times 1$ joint velocity vector, and $J(q(t))$ is the $m\times n$ Jacobian matrix. The main goal is to obtain the inverse kinematics equation:

$$\dot{q}(t) = J^{-1}(q(t))\dot{x}(t) \tag{2}$$

Since the manipulator is redundant $(n > m)$, the Jacobian matrix is not square. Equation (2) is solved by pseudoinverse of the Jacobian matrix that locally minimizes the norm of joint velocities. Equation (2) is then transformed into

$$\dot{q}(t) = J^{+}(q(t))\dot{x}(t) \tag{3}$$

with the pseudoinverse Jacobian matrix

$$J^{+} = J^{T}(JJ^{T})^{-1}\,. \tag{4}$$

Furthermore,

$$\dot{q}(t) = J^{+}(q(t))\dot{x}(t) + (I - J^{+}J)\dot{q}_a(t) \tag{5}$$

where $\dot{q}_a$ is a vector of arbitrary joint velocities projected in the null-space of $J$. The vector $\dot{q}_a$ specifies the additional redundancy constraints. For the Jacobian pseudoinverse computation the Moore-Penrose pseudoinverse method has been used.

### 2.2 Redundant robot manipulability

As mentioned before, robot joint variables are denoted by n-dimensional vector $q$. It is then considered that the set of all end-effector velocities that are realizable by joint velocities is such that the Euclidean norm of $\dot{q}$

$$\|\dot{q}\| = (\dot{q}_1^2 + \dot{q}_2^2 + ... + \dot{q}_n^2)^{1/2}\,, \tag{6}$$

satisfies $\|\dot{q}\| \le 1$. This set is an ellipsoid in the m-dimensional Euclidean space ($m$ is the dimension of end-effector position and orientation vector). In the direction of the ellipsoid major axis the end-effector can move at a high speed. On the other hand, in the direction of the minor axis it can move only at a low speed. If the ellipsoid is almost a sphere, the end-effector can move uniformly in all directions y. This ellipsoid is called the manipulability ellipsoid.

The ellipsoid volume is one of the representative measures for the manipulation ability derived from the manipulability ellipsoid. It is given by $c_m w$ ($c_m$ is a constant)

$$w = \sigma_1\sigma_2...\sigma_m \tag{7}$$

where the scalars $\sigma_1, \sigma_2, \ldots, \sigma_m$ are singular values of $J$ and the variable $w$ is called the manipulability measure for the manipulator configuration $q$. The manipulability measure $w$ has the following properties (Sciavicco and Siciliano, 1996)

$$w = \sqrt{\det(J(q)J^{T}(q))} \tag{8}$$

If $m = n$, i.e. when non-redundant manipulators are considered, the measure $w$ reduces to:

$$w = \left|\det(J(q))\right| \tag{9}$$

Usually $w \ge 0$ is satisfied; in such a case $w = 0$ if and only if $rank(J(q)) \le m$, i.e. when the manipulator is in a singular configuration. From this fact the manipulability measure can

be regarded as a kind of distance of the manipulator configuration from the singular one.

The increase in manipulability measure during the robot movement cannot influence the end-effector movement. The vector $\dot{q}_a$ is used, because it is projected with $(I - J^+ J)$ into the null space and doesn't affect the end-effector motion. If the null space motion of the robot is to be increased the manipulability measure the vector $\dot{q}_a$ has to satisfy

$$\dot{q}_a = \nabla H(q) \tag{10}$$

where $H(q)$ is the objective function in the optimization process and $\nabla H(q)$ is the gradient of the manipulability measure $w$ (Nakamura, 1991).

## 3. CMAC AND HCMAC NEURAL NETWORK

To derive the HCMAC neural network, we first briefly introduce the CMAC neural network based on different basis functions.

### 3.1 CMAC neural network

The CMAC, a look-up table neurocomputing technique, has fast learning ability and good local generalization capability for approximating nonlinear functions, and can be viewed as a basis function network (BFN). Several types of basis functions are used in CMAC neural networks. The basic type is the constant basis function that is usually implemented in association memory selection.

Before applying the basis function, the input data of each state variable must be quantized into discrete regions. Herein, the number of discrete regions is termed a resolution. Each input data can be mapped to several actual memory units via an association memory selection vector.

These mapped actual memory units are called hypercubes, i.e. each input data is distributively mapped on each hypercube.

For an input state, the actual output is obtained as the sum of stored contents for hypercubes covering the state, i.e. the actual output of a specific input state *s* can be mathematically expressed as follows:

$$y(s) = a^T(s)w = \sum_{j=1}^{N_h} a_j(s) w_j \tag{11}$$

where $w=[w_1, w_2,\ldots,w_{Nh}]^T$ is the vector of actual memory contents, $N_h$ is the entire memory size, $a^T=[a_1(s), a_2(s),\ldots, a_{Nh}(s)]$ is the memory selection vector.

To adjust the weight values during each learning cycle, the conventional CMAC uses a supervised learning approach. Its learning rule can be described as follows:

$$w_t = w_{t-1} + \frac{\alpha}{N_e} a(s)(\hat{y}(s) - a^T(s) w_{t-1}) \tag{12}$$

where $w_t$ is the vector of actual memory contents at time *t*, $w_{t-1}$ is the vector of actual memory contents at previous time *t-1*, $\alpha$ is the learning rate, $\hat{y}(s)$ is the desired output value, and $\hat{y}(s) - a^T(s) w_{t-1}$ is the error for the input training state *s*.

The conventional CMAC uses a constant basis function. The non-differentiability leads to some limitations when using conventional CMAC in real-world applications. Therefore, the constant basis function is replaced by various differentiable functions e.g. the Gaussian basis function. The mathematical formulation of one-dimensional Gaussian basis function $\phi$ can be described as follows :

$$\phi(s) = e^{-((s-m)/\sigma)^2} \tag{13}$$

where *m* is a hypercube centre, $\sigma$ is a hypercube radius, and *s* is a specific input state. The output of CMAC neural network with Gaussian basis function with $N_v$ dimensional problem is revised to become:

$$y(s) = \sum_{j=1}^{N_h} \left[ a_j(s).w_j.\left( \prod_{i=1}^{N_v} e^{-((s_i - m_{ji})/\sigma_{ji})^2} \right) \right] \tag{14}$$

where $a_j(s)$ is the $j^{th}$ element of association memory selection vector for a specific input space *s*, $w_j$ is the $j^{th}$ memory allocation of the actual memory, $s_i$ is the input value of $i^{th}$ dimension for a specific input state *s*, $m_{ji}$ is the corresponding hypercube centre, and $\sigma_{ji}$ is the corresponding hypercube radius. The CMAC with a Gaussian basis function as the nonconstant differentiable basis function is termed GCMAC.

### 3.2 HCMAC neural network

The HCMAC neural network consists at least of three differentiable GCMACs.

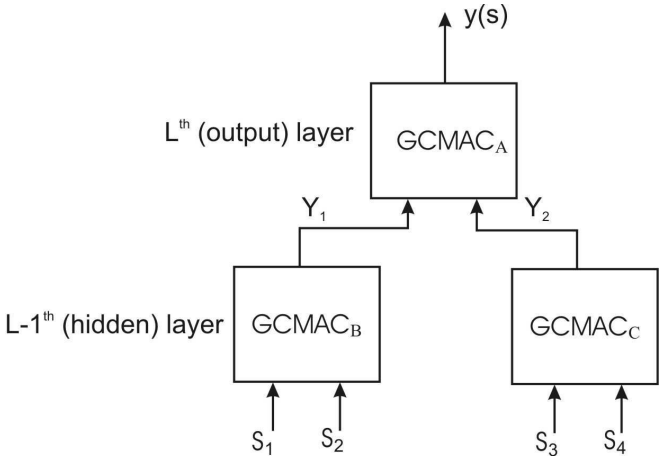


Fig. 1 Topology of the HCMAC neural network structure

Fig. 1 illustrates the smallest topology of the HCMAC neural network indicating that each GCMAC includes two input values, and the output values of the first-layer GCMACs serve as input values for the second-layer GCMACs.

The structure of HCMC in Fig. 1 comprises four inputs $s_i$ where $s_i(i=1,2,\ldots,4)$ represents the $i^{th}$ input value, $y_j(j=1,2)$

is the $j^{th}$ output of the hidden layer, and *y(s)* is the output of the whole HCMAC neural network for a specific input state *s*. Using the HCMAC neural network is a problem of four input values divided into two two-dimensional GCMACs.

The gradient steepest-descent method has been chosen as the learning method for the HCMAC neural network training. First the error cost function *E* is defined as follows

$$E = \frac{1}{2}(\hat{y}(s) - y(s))^2 \tag{15}$$

where $\hat{y}(s)$ and *y(s)* are the desired and actual HCMAC neural network output values for the input state *s*, respectively. In the GCMAC neural network equation there are three parameters to be tuned during the learning process: the weight *w*, the radius $\sigma$ and the center *m*. First the GCMAC output layer is trained, where the updates for the GCMAC parameters are as follows:

$$\begin{aligned}\Delta w_j &= -\frac{\alpha}{N_e}\frac{\partial E}{\partial w_j} \\ &= \frac{\alpha}{N_e}(\hat{y}(s) - y(s)).a_j(s).\prod_{i=1}^{2} e^{-((y_i - m_{ji})/\sigma_{ji})^2}\end{aligned} \tag{16}$$

where $\Delta w_j$ is the updated weight value of the $j^{th}$ actual memory in the $GCMAC_A$, $w_j$ is the weight value of the $j^{th}$ actual memory, $\alpha$ is a learning rate and $N_e$ is the number of mapped hypercubes for the input state *s*.

$$\begin{aligned}\Delta\sigma_{ji} &= -\frac{\alpha}{N_e}\frac{\partial E}{\partial \sigma_{ji}} \\ &= \frac{\alpha}{N_e}(\hat{y}(s) - y(s)).a_j(s).\left[\prod_{i=1}^{2} e^{-((y_i - m_{ji})/\sigma_{ji})^2}\right].\frac{2(y_i - m_{ji})^2}{\sigma_{ji}^3}\end{aligned} \tag{17}$$

where $\Delta\sigma_{ji}$ is the updated radius of the $i^{th}$ dimension for the $j^{th}$ mapped hypercube for the input state *s* in the $GCMAC_A$, $\sigma_{ji}$ is radius of the $i^{th}$ dimension for the $j^{th}$ mapped hypercube of the input state *s*.

$$\begin{aligned}\Delta m_{ji} &= -\frac{\alpha}{N_e}\frac{\partial E}{\partial m_{ji}} \\ &= \frac{\alpha}{N_e}(\hat{y}(s) - y(s)).a_j(s).\left[\prod_{i=1}^{2} e^{-((y_i - m_{ji})/\sigma_{ji})^2}\right].\frac{2(y_i - m_{ji})}{\sigma_{ji}^2}\end{aligned} \tag{18}$$

where $\Delta m_{ji}$ is the updated centre value of the $i^{th}$ dimension for the $j^{th}$ mapped hypercube of the input state *s* in the $GCMAC_A$, $m_{ji}$ is the radius of the $i^{th}$ dimension for $j^{th}$ mapped hypercube of the input state *s*. The parameters in $GCMAC_B$ and $GCMAC_C$ are updated according to the conventional backpropagation rule, where the derivatives $\partial y/\partial y_1$ and $\partial y/\partial y_2$ represent the error backpropagated to $GCMAC_B$ and $GCMAC_C$.

## 4. EXPERIMENTS

In this paper the objective of the neural network is to approximate the manipulability gradient computation function. In order to test the control system with the neural network, the kinematics of a 7-DOF redundant robotic system model was considered.

The neural network was tested on a position control system in Fig. 2. The control system input is the desired position and the output is the robot end-effector position.

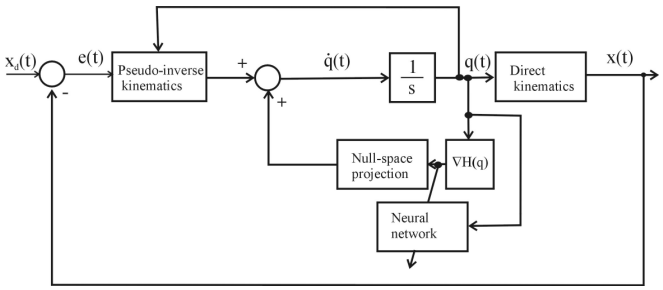


Fig. 2 Position control scheme with null-space motion and CMAC training

From Fig. 2 it is evident that the pseudoinverse kinematics (5) was used to compute the desired robot joint velocities. Apart this, the null-space motion has been added, whereby the manipulability measure is maximized. It learns the gradient values of the manipulability measure. The neural network needs 7 input values (robot joint positions) and 7 output values. The conventional CMAC neural network allows to use any number of inputs and outputs. Otherwise, HCMAC neural network needs to implement appropriate GCMACs according to the number of inputs. This is done using a full binary tree approach, or it can be optimized according to the input requirements. In this paper the structure of the full binary tree was optimized to get a minimal GCMACs structure.

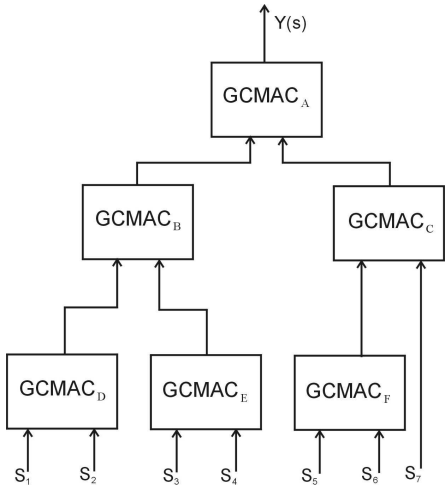


Fig. 3 Optimized HCMAC neural network structure for 7 inputs and 7 outputs.

To implement 7 outputs to the HCMAC structure the output $GCMAC_A$ was extended with 6 additional memories to store the other 6 output values. Fig. 3 shows the optimized HCMAC neural network structure with 7 inputs and 7 outputs.

Fig.3 shows that for a 7 inputs system at least 6 GCMACs are needed.

The testing simulation was the robot task to move from one point in space to another one; during the motion the manipulability of the robot configurations was increased. During simulation, the exact robot motion started at the position (0.1455, 0.6588, 1.3896) and stopped at the position (0.25, 0.15, 0.5). The whole simulation movement took 5 seconds.

4.1 Simulation results

The first neural network tested for manipulability modeling was the conventional CMAC neural network with the Gaussian basis function. The learning process of this neural network was performed online during the robot motion from the starting to the end point. After 10 training cycles, CMAC replaced the gradient computation and the system was tested with this neural network. Fig. 4 shows the comparison of manipulability measures obtained with CMAC neural network and the original gradient computation. It is evident that the CMAC neural network very well approximates the gradient computation and manipulability measure.

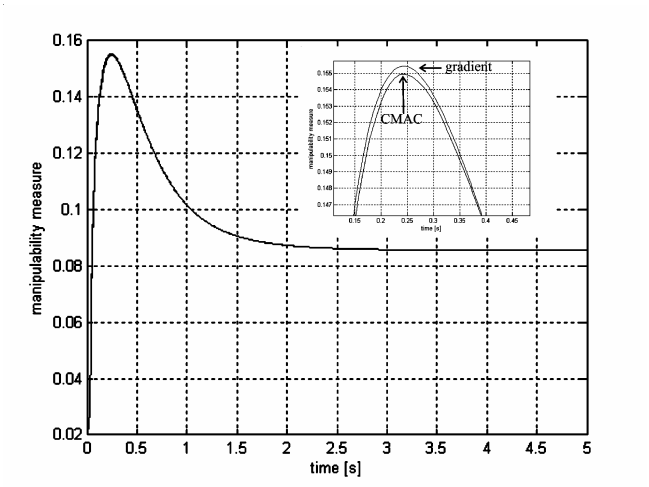


Fig. 4 Manipulability measure obtained with CMAC and with gradient computation after 10 trainings

To measure the approximation performance the integral absolute error (IAE) was used, defined as follows

$$IAE = \int_0^T |w_1(t) - w_2(t)| dt \tag{19}$$

where $w_1$ and $w_2$ are manipulability measures for the CMAC and the original gradient computation.

In our case the IAE values obtained for the CMAC and the gradient computation were 0.0744 and $7.44*10^{-5}$ per one training sample, respectively. Time responses of the CMAC and the gradient computation outputs are in Fig. 5 which shows that the CMAC very well approximates all elements of the manipulability gradient vector. CMAC outputs follow all the desired gradient values. Each curve represents one gradient vector element.

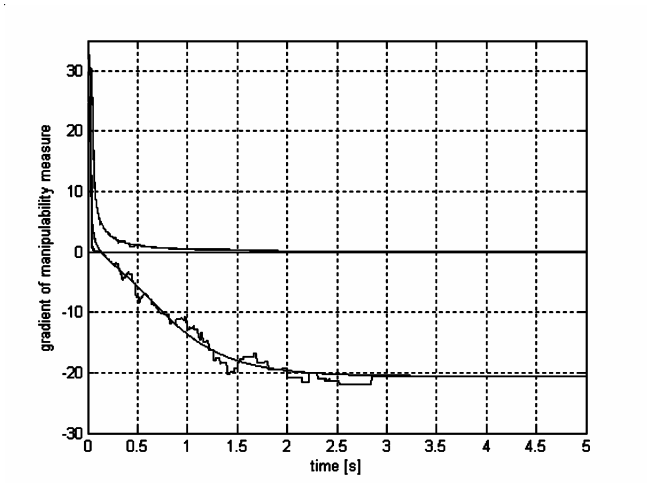


Fig. 5 Comparison between conventional CMAC and analytical gradient computation after 10 training cycles

The second tested neural network was the HCMAC. The same training samples were taken as for the CMAC neural network. After the training, HCMAC neural network also replaced the original gradient computation and was compared with the conventional approach. Fig. 6 shows the results.

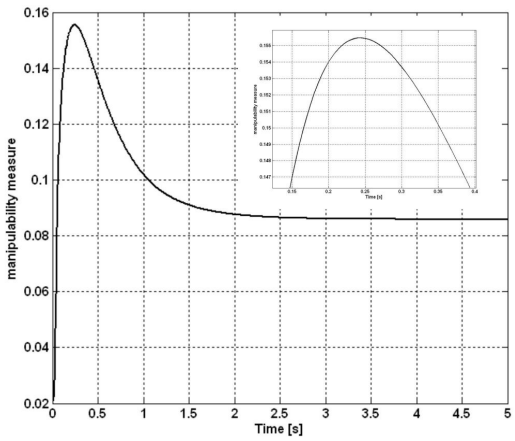


Fig. 6 Manipulability measure obtained with HCMAC and with gradient computation

It can be seen that the HCMAC neural network approximates the manipulability measure better than the CMAC neural network. For this approximation the total IAE computed was 0.0028, what is $2.8*10^{-6}$ per one training sample. The exact approximation of the manipulability gradient vector is shown in Fig. 7; it can be seen that all gradient vector elements were approximated better than in case of the CMAC neural network and the HCMAC neural network output function is much smoother than the CMAC neural network. The reason for this result is a better distribution of input elements among 6 GCMACs. Beside the approximation improvement, also the speed of computation for the HCMAC, CMAC and the Gradient method were tested.

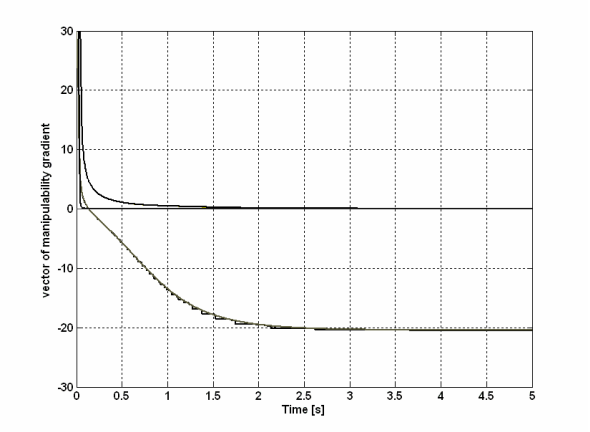


Fig. 7 Gradient values obtained using HCMAC and conventional gradient computation

For this experiment, the gradient was computed for 100 and 1000 randomly chosen robot configurations. The task function for measuring speed was the manipulability measure function. In Table 1 the results of this testing are presented.

**Table 1. HCMAC, CMAC and conventional gradient method computation speeds**

| Number of configurations | Gradient in [s] | CMAC in [s] | HCMAC in [s] |
|---|---|---|---|
| 100 | 0.1250 | 0.0310 | 0.09852 |
| 1000 | 1.0935 | 0.2580 | 0.86217 |

The results prove that CMAC and HCMAC neural networks compute the gradient faster than the original gradient method. This is the advantage of using HCMAC or CMAC neural networks.

The CMAC or the HCMAC neural networks can be used as an alternative to the commonly used computation method and can compute the outputs even faster than the original one. This is an advantage in the redundant robot kinematics control, where any saved computation time allows more time to be used for additional tasks. Compared with CMAC, the HCMAC neural network has better approximation abilities. The purpose of using it is the spare of memory without loss of precision. This was fully achieved using HCMAC instead of CMAC neural network.

To reduce the memory requirements, CMAC neural network with 7 inputs needs hashing algorithm because it is impossible to implement a memory able to cover the whole hyperspace $R^7$; however the hashing algorithm brings more difficulties as hash collisions. The HCMAC neural network covers the whole input space, but the computation is more complex than in CMAC neural network.

## 5. CONCLUSION

In this paper a new HCMAC neural network for manipulability gradient modelling was proposed and possible replacements of the original gradient computation method were tested. Simulation results obtained with the inverse kinematics control scheme for a 7 DOF redundant robot show that the HCMAC can learn this gradient values replace the original computation method with very good results. This neural network was also compared with the conventional CMAC neural network with Gaussian basis function. Testing results proved that the HCMAC can better approximate gradient computation, but the computation time for one output value is higher than in case of CMAC. Both neural networks showed than they are able compute the gradient faster than the original method. Moreover, in cases when the redundant robot control system is applied the designer always needs some computation time savings and this might be the way to succeed.

## REFERENCES

Albus J. S. (1975), A new approach to manipulator control: The Cerebellar model articulation controller (CMAC) *Trans. ASME J. Dynam. Syst., Meas. Contr.*, vol. 97, no. 8, pp 220-227, 1975.

Cotter N.E. and Guillerm T. J.(1991), The CMAC and a theorem of Komogorov,. *Neural Networks,* vol. 5, pp. 221-228

Crane D., Duffy J. (1998), *Kinematic Analysis of Robot Manipulators,* Cambridge University Press.

Glanz F. H., Miller W.T., and Kraft L.G. (1991), An overview of the CMAC neural network, *in Proc. 1991 IEEE Neural Networks Ocean Eng.,* pp. 301-308

Hu J. and Pratt F. 1 (1999), Self-organizing CMAC neural networs and adaptive dynamic control, in *Proc. 1999 IEEE Intel. Contr/Intell. Syst. Semiotics,* 1999, pp. 259-265.

Hu J., Pratt J., Pratt G. (1999), Stable adaptive control of bipedal walking; Robot with CMAC neural networks, in *Proc. 1999 IEEE Int. Conf. Robot. Automat.,* vol. 2

Miller W. T. and Glanz F.H.(1990) , CMAC: And associative neural network alternative to backpropagation, *in Proc. IEEE,* vol. 78, pp. 1561-1567

Lane S.H., Handelman D. A., and Gelfand J. J. (1992), Theory and development of higher-order CMAC neural networks, *IEEE Contr. Syst. Mag.,* vol. 12,pp. 23-30.

Li Y., Leong S.H., (2000) Kinematics Control of Redundant Manipulators Using CMAC Neural Network, *The 5th World Multiconference on Systemics, Cybernetics and Informatics (SCI2001).* pages 274-279., Orlando, USA.

Lin C.S. and Li C.K. (2000), A sum-of-product neural network (SOPNN), *Neurocomput.,* vol. 30, pp. 273-291.

Lin C.S. and Li C.K. (1999), A memory based self generated basis function neural network, *Int. J. Neural Syst.,* vol. 9, no. 1,pp. 41-59.

Lin C.S. and Kim H., (1995) Selection of learning parameters for CMAC-based adaptive critic learning, *IEEE Trans. Neural Networks,* vol. 6, pp. 642-647.

Nakamura Y., (1991) *Advanced Robotics: Redundancy and Optimization,* Addison-Wesley Publishing Company, Inc., Reading, Massachusetts.

Sciavicco L., Siciliano B. (1996), *Modeling And Control of Robot Manipulators*, The McGraw-Hill Companies, Inc.

Zhang K. and Qian F.(2000), Fuzzy CMAC and its application, *in Proc. 3th World Congr. Intell. Contr. Automat.,* Hefei, China, pp. 944-947.