UDC 621.391:681.3.06
Original scientific paper
Received: 21.03.2001.

CMAC neural network method with application to kinematics control of a redundant manipulator

Yangmin Li and **Sio Hong Leong**
Faculty of Science and Technology, University of Macau, P.O. Box 3001, MACAU
e-mail: YMLi@umac.mo

SUMMARY

The inverse kinematics problems of redundant manipulators have been investigated for many years. The conventional method of solving this problem is through applying the Jacobian Pseudoinverse Algorithm, which is effective and able to resolve the redundancy for a redundant manipulator. However, computational effort makes it not suitable for real time control. Recently, neural networks have been widely used in robotic control because they are fast, fault-tolerant and able to learn. In this paper, we will present the application of CMAC (Cerebellar Model Articulation Controller) neural network for solving the inverse kinematics problems in real time. Simulations have been carried out for a five-link manipulator in order to evaluate the performance of the CMAC neural network. Through computer simulation, we found CMAC NN method is especially suitable for real time control of robots and solving nonlinear function approximation problem.

Key words*: inverse kinematic problem, neural network method, redundant manipulator, real time control.*

1. INTRODUCTION

Robotic manipulators have been widely used in different industries in order to perform many kinds of specific work such as welding, painting, assembling, nuclear material handling, space and sea exploring etc. Control of a manipulator is extremely important, which involves trajectory planning, inverse kinematics and dynamics calculations as well as control algorithms and strategies. Among which, inverse kinematics of redundant manipulator is the interest of this paper. Kinematically redundant manipulators are of special interest due to their redundancy and can be utilized for avoiding singularities, obstacles, working in limited workspace and increasing fault-tolerance capability.

Conventionally, the Jacobian Pseudoinverse Algorithm [1] has been applied to solve the inverse kinematics problems due to its ability to satisfy additional constraints through mapping the velocities corresponding to the additional constraints into the null space of the Jacobian ($\boldsymbol{J}$) while tracking the desired workspace trajectory. However, the Jacobian Pseudoinverse Algorithm involves the inversion of the $\boldsymbol{J}$ matrix which can present a serious inconvenience not only at singularity but also in the neighborhood of a singularity, though the damped least squares (DLS) inverse can render the inversion better considered from numerical viewpoint.

Recent years, neural networks (NN) [2] have been successfully applied in robotic control. Their generalization capacities and structures make them robust and fault-tolerant in algorithms. They are able to solve a problem that they have not solved before. Also, NNs are composed of many neurons, even though some of them are damaged, the output of the network will not have been affected too much. Another advantage of NNs is their ability to solve highly nonlinear problems. The properties of NNs above make them so promising in application to the robot control problems.

There are many kinds of NNs [3]. Among which, CMAC (Cerebellar Model Articulation Controller) neural networks [4, 5] will be selected in this paper due to their speed of convergence as well as its simple δ-learning method. CMAC neural network was proposed by Albus in 1975 in order to simulate the function of our cerebellum. It is an associative neural network in which the inputs determine a small subset

of the network and that subset determines the outputs corresponding to the inputs. The associative mapping property of CMAC assures local generalization, that is, similar inputs produce similar outputs while distant inputs produce nearly independent outputs. CMAC is similar to perceptron, although it is a linear relationship on the neuron scope, in overall it is a nonlinear relationship.

Many researchers investigated the application of CMAC neural networks. W.T. Miller III etc. [5] developed a robot tracking system consisting of a manipulator attached with a video camera for tracking an object on a conveyor and having the image of the object specifically positioned and oriented on the screen without giving any robot kinematics information, height measurement or any camera-screen calibration. The CMAC was used to learn all these parameters. Moreover, CMAC was modified by C.J.B. Macnab etc. [6] to utilize radial basis functions for precision control of flexible-joint robots in order to deal with the elasticity. E. Oyama etc. [7] proposed a Modular Neural Net System for learning inverse kinematics in order to deal with the multi-valued and discontinuous function of the inverse kinematics system. J.F. Gardner etc. [8] investigated the application of neural network in a two-link manipulator trajectory tracking, in which the NN was used to compute the inverse of the Jacobian matrix. A. Paula etc. [9] applied Attentional Mode Neural network (AMNN) in leading a robot arm in 3-D space to a goal point in real time. PSOM Network was presented by J.A. Walter [10] to learn the inverse kinematics based on only 27 data points.

This paper is organized as follows. Section 2 presents the network method in solving the complex inverse kinematics problems. Section 3 presents the CMAC neural network in solving the complex inverse kinematics of manipulators. Simulation will be performed with a 5-link planar manipulator tracking a circle trajectory in Section 4. Finally some conclusions are presented in Section 5.

2. CONVENTIONAL MANIPULATOR INVERSE KINEMATICS

For a kinematic redundant manipulator, the end-effector position is a function of the joint variables which can be expressed as the following kinematics equation:

$$\boldsymbol{x} = f(\boldsymbol{e}) \tag{1}$$

where: $\boldsymbol{x} \in R^m$ denotes end-effector position vector and $\boldsymbol{q} \in R^n$ is the joint angle vector. Here $n>m$ in case of redundant manipulator and $r=n-m$ is defined as degree of redundancy.

The differential kinematics equation can be derived as:

$$\dot{\boldsymbol{x}} = \boldsymbol{J}(q)\dot{\boldsymbol{e}} \tag{2}$$

where $\dot{\boldsymbol{x}} \in R^m$ denotes end-effector velocity vector, $\dot{\boldsymbol{e}} \in R^n$ represents joint velocity vector, and $\boldsymbol{J}(q) \in R^{m \times n}$ is the Jacobian matrix.

The inverse kinematics problem is to determine the joint variables corresponding to a given end-effector's position and orientation. That is, given a desired workspace trajectory, how we can find out the corresponding joint space trajectory. This is a complex problem since redundant manipulators have more than necessary degrees of freedom (DOF), multiple or infinite solutions may exist. Moreover, the equations to solve are usually nonlinear and it is difficult to find a closed-form solution.

Thus, our main concern is to solve the following inverse kinematics equation:

$$\dot{\boldsymbol{e}} = \boldsymbol{J}^{-1}(\dot{e})\,\dot{\boldsymbol{x}} \tag{3}$$

Since the manipulator is redundant (n>m), the Jacobian matrix is not square. Usually we can solve Eq. (3) by using the pseudoinverse of the Jacobian that locally minimizes the norm of joint velocities [1]. Equation (3) now becomes:

$$\dot{\boldsymbol{e}} = \boldsymbol{J}^{+}(\dot{e})\dot{\boldsymbol{x}} \tag{4}$$

where matrix $\boldsymbol{J}^{+}$ is:

$$\boldsymbol{J}^{+} = \boldsymbol{J}^{T}(\boldsymbol{J}\boldsymbol{J}^{T})^{-1} \tag{5}$$

Furthermore, through:

$$\dot{\boldsymbol{e}} = \boldsymbol{J}^{+}(\boldsymbol{q})\dot{\boldsymbol{x}} + \left(\boldsymbol{I} - \boldsymbol{J}^{+}\boldsymbol{J}\right)\dot{\boldsymbol{q}}_a \tag{6}$$

where $\dot{\boldsymbol{q}}_a$ is a vector of arbitrary joint velocities projected in the null space of $\boldsymbol{J}$. We can resolve the redundancy by specifying $\dot{\boldsymbol{q}}_a$ so as to satisfy an additional constraint.

As we can see, the above pseudoinverse formulation involves the inverse of $\boldsymbol{J}$ and a lot of calculations are needed for solving the equation. Thus it is not suitable for real-time control.

3. CMAC NEURAL NETWORK METHOD

3.1 Principles

The structure of CMAC neural network [5] is shown in Figure 1.

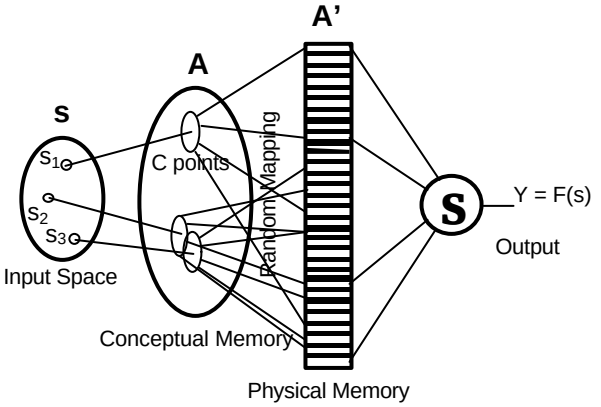


Fig. 1 Block diagram of CMAC

The input vectors in the input space *s* are a number of sensors in real world. Input space consists of all possible input vectors. CMAC maps the input vector into *C* points in the conceptual memory *A*. As shown in Figure 1, two "close" inputs will have overlaps in *A*, the closer the more overlapped, and two "far" inputs will have no overlap.

Since practical input space is extremely large, in order to reduce the memory requirement, *A* is mapped onto a much smaller physical memory *A'* through hash coding. So, any input presented to CMAC will generate *C* physical memory locations. The output *Y* will be the summation of the content of the *C* locations in *A'*.

From the above, we can see that the associative mapping within the CMAC network assures that nearby points in the input space generalize while distant points do not generalize. Moreover, since from *A'* to *Y* is a linear relationship but from *s* to *A'* is a nonlinear relationship, the nonlinear nature of the CMAC network perform a fixed nonlinear mapping from the input vector to a many-dimensional output vector.

3.2 Learning

Network training is typically based on observed training data pairs *s* and Y_d, where Y_d is the desired network output corresponding to the input *s*, using the least mean square (LMS) training rule. The weight can be calculated by:

$$w(t+1) = w(t) + \eta \frac{Y_d - Y(s)}{C} \tag{7}$$

where η is the learning step length.

Thus, if we define an acceptable error *e*, no changes needed for the weights when $[Y_d - Y(s)] \le e$. The training can be done after a set of training samples being tested or after each training sample being tested.

3.3 Solving inverse kinematics problems of kinematically redundant manipulators

Here we present the application of CMAC neural network in the inverse kinematics control of the manipulator as shown in Figure 2. By analyzing the block diagram, we can obtain:

$$\dot{X}^{t+1} = \dot{X}_d^{t+1} + \boldsymbol{K}e \tag{8}$$

which is equivalent to:

$$\dot{e} + \boldsymbol{K}e = 0 \tag{9}$$

If $\boldsymbol{K}$ is a positive definite matrix, the system is asymptotically stable [1].

The shadowed box is our main concern since it solves that inverse kinematics problem. The inputs are the current manipulator configuration at time *t* (q^t) and the desired end-effector position increments at time *t+1* (DX_d^{t+1}). The output is the corresponding joint angle increments that are fed together with the current joint angles to the robot manipulator in order to have it moved to the desired configuration.

Here "NN" is the CMAC neural network and "Opt" is the optimization process which is used to find the desired joint angle increments corresponding to the desired end-effector position increments as well as satisfying specified additional constraints.

In fact, the inverse kinematics problems can be solved by optimization method. However, optimization process usually takes quite a long time and thus is not suitable for real time control. Here we use CMAC neural network to generate an initial solution Dq^{t+1} for the optimization process. The difference between the initial solution Dq^{t+1} and the output of the optimization process Dq_d^{t+1} is feedback to modify the weights of the CMAC neural network, that is, CMAC neural network is learning to produce an output as close as possible to the optimization output. Thus the time spent in optimization can be reduced as the CMAC neural network learns more and more. Eventually, the output of the CMAC neural network can replace that of the optimization process.

As for the optimization process, we apply the gradient method to minimize the following objective function:

$$\boldsymbol{U} = \frac{a}{2}\boldsymbol{e}^T\boldsymbol{e} + \frac{b}{2}D\dot{\boldsymbol{e}}^T D\dot{\boldsymbol{e}} \tag{10}$$

where $\boldsymbol{e}$ is the position error vector, $D\boldsymbol{q}$ is joint angle changing vector, a and b are the weight constants.

The second term on the right is for minimizing the joint angle changes, thus smoothing the motion as well as minimizing the energy consumption. Then we have:

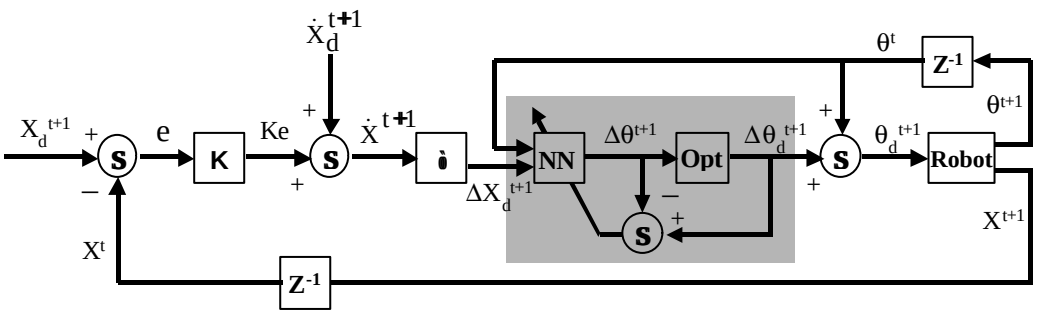


Fig. 2 Block diagram of CMAC NN for solving inverse kinematics problems

$$\frac{\partial U}{\partial D\dot{\theta}} = -a\, e^T J + b\, D\dot{\theta}^T \qquad (11)$$

where J is the Jacobian matrix and thus:

$$D\dot{\theta}_{k+1} = D\dot{\theta}_k - h\frac{\partial U}{\partial D\dot{\theta}} \qquad (12)$$

where, k is the number of iteration and h is the length of learning step.

4. SIMULATION

In order to evaluate the performance of CMAC, we have developed a simulation software under Microsoft Windows 98 system using Borland C++ Builder [11, 12]. The UNH_CMAC code [13] written in *C* was referenced in this paper, which includes multiple designs for the receptive field lattice and the receptive field sensitivity functions.

During simulation, parameters have been chosen as follows: the sampling time $Dt=0.02s$, constant $K=1/Dt=50$ and the parameters for the optimization are: $a=300.0$; $b=1.0$; $h=0.003$.

4.1 Manipulator kinematics

The manipulator for simulation is a 5-link planar manipulator as shown in Figure 3. Each link is connected with revolute joints (5-DOF, $m=2$, $n=5$, $r=n-m=3$) as shown in Figure 4. No joint angle limits are imposed on. The total length of the manipulator is 1.55 m with the first link of 0.5 m, second and third link of 0.25 m, fourth and fifth link of 0.275 m. The height of the base support is 0.35 m. The Denavit-Hartenberg defined parameters of the manipulator are listed in Table 1.

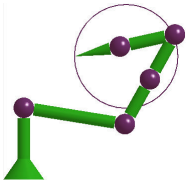


Fig. 3 Initial configuration of the manipulator

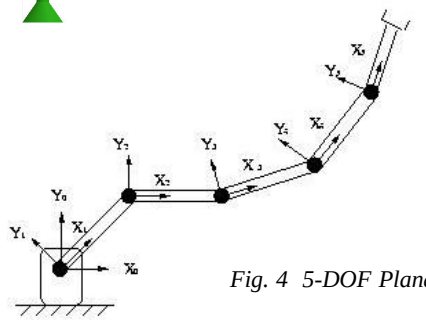


Fig. 4 5-DOF Planar Manipulator

Table 1 Link Parameters

Link	a_{i-1}	a_{i-1}	d_i	q_i
1	*0*	*0*	*0*	q_1
2	*L1*	*0*	*0*	q_2
3	*L2*	*0*	*0*	q_3
4	*L3*	*0*	*0*	q_4
5	*L4*	*0*	*0*	q_5
6	*L5*	*0*	*0*	*0*

The kinematics equations of the 5-DOF planar manipulator can be obtained as:

$$\begin{bmatrix} X \\ Z \end{bmatrix} = \begin{bmatrix} l_1S_1 + l_2S_{12} + l_3S_{123} + l_4S_{1234} + l_5S_{12345} \\ l_1C_1 + l_2C_{12} + l_3C_{123} + l_4C_{1234} + l_5C_{12345} \end{bmatrix} \qquad (13)$$

where: l_1, l_2, l_3, l_4, l_5 denotes length of each link respectively and q_1, q_2, q_3, q_4, q_5 represents each joint angle respectively. $S_{ijklm}=sin(q_i+q_j+q_k+q_l+q_m)$, $C_{ijklm}=cos(q_i+q_j+q_k+q_l+q_m)$, $\{i\ j\ k\ l\ m\}=\{1\ 2\ 3\ 4\ 5\}$.

The relationship between the end-effector velocity and the joint angular velocity is:

$$\dot{x} = J(q)\dot{q} \qquad (14)$$

where:

$$J(q) = \begin{bmatrix} \frac{\partial X}{\partial q_1} & \frac{\partial X}{\partial q_2} & \frac{\partial X}{\partial q_3} & \frac{\partial X}{\partial q_4} & \frac{\partial X}{\partial q_5} \\ \frac{\partial Z}{\partial q_1} & \frac{\partial Z}{\partial q_2} & \frac{\partial Z}{\partial q_3} & \frac{\partial Z}{\partial q_4} & \frac{\partial Z}{\partial q_5} \end{bmatrix}$$

in which:

$$\frac{\partial X}{\partial q_1} = l_1C_1+l_2C_{12}+l_3C_{123}+l_4C_{1234}+l_5C_{12345}$$
$$\frac{\partial X}{\partial q_2} = l_2C_{12}+l_3C_{123}+l_4C_{1234}+l_5C_{12345}$$
$$\frac{\partial X}{\partial q_3} = l_3C_{123}+l_4C_{1234}+l_5C_{12345}$$
$$\frac{\partial X}{\partial q_4} = l_4C_{1234}+l_5C_{12345}$$
$$\frac{\partial X}{\partial q_5} = l_5C_{12345}$$
$$\frac{\partial Z}{\partial q_1} = -(l_1S_1+l_2S_{12}+l_3S_{123}+l_4S_{1234}+l_5S_{12345})$$
$$\frac{\partial Z}{\partial q_2} = -(l_2S_{12}+l_3S_{123}+l_4S_{1234}+l_5S_{12345})$$
$$\frac{\partial Z}{\partial q_3} = -(l_3S_{123}+l_4S_{1234}+l_5S_{12345})$$
$$\frac{\partial Z}{\partial q_4} = -(l_4S_{1234}+l_5S_{12345})$$
$$\frac{\partial Z}{\partial q_5} = -l_5S_{12345}$$

The desired end-effector trajectory in our simulation will be a circle with radius of $0.25\ m$ and centered at $(0.5, 0.6)$. The desired angular velocity (w) is $0.5p\ rad/s$. The trajectory equations are as follows:

$$X_d(t) = 0.5 - 0.25cos(wt) \qquad (15)$$
$$Z_d(t) = 0.25 + 0.25sin(wt) \qquad (16)$$

Thus the cycle time is $4\ s$. The initial configuration of the manipulator is shown in Figure 3 with $q_1(0)=1.730385$, $q_2(0)=-1.219399$, $q_3(0)=0.0$, $q_4(0)=-2.29649$ and $q_5(0)=0.0$.

In the following simulation, the norm error is defined as:

$$E_{norm}(t) = \sqrt{[X_d(t)-X(t)]^2 + [Z_d(t)-Z(t)]^2} \qquad (17)$$

and the average error is defined as:

$$E_{av} = \frac{\sum_{n=1}^{2p/(wDt)} E_{norm}(nDt)}{2p/(wDt)} \qquad (18)$$

4.2 CMAC neural network structure

The structure of the CMAC neural network is shown in Figure 5 where the input vector dimension is 7 and the output vector dimension is 5.

In our simulation, the number of vectors in CMAC memory is 10000 and the generalization parameter is 64.

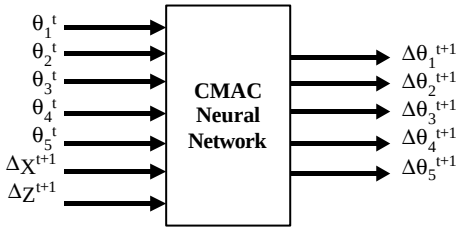


Fig. 5 Input / Output of the CMAC neural network

4.3 Simulation results

First of all, we would like to see how the optimization processes perform without the use of CMAC neural network. Table 2 shows the results for different optimization iteration numbers from 1 to 100. We can see that up to 50 iterations, the norm error has become stable with maximum norm error of about *0.18 mm*. The table shows also the average error against number of iterations.

Now, we apply the CMAC neural network to do the simulation. The optimization iterations and learning rates are shown in Table 3. We found that after 1300 trainings, the maximum norm error is about *0.3 mm* (*0.02%* of workspace dimension). The result is comparable with that of applying the optimization process alone. Notice that there is only one optimization iteration, thus CMAC neural network really can reduce the optimization time. In order to observe whether the system is stable, we have tested above 5000 trainings. From Table 3, we can see that after 6000 trainings, the norm error is similar to the result after 1000 trainings. Also, the average error is mostly below *0.1 mm*. Therefore we can say that the system is stable.

Next, the robustness of the system will be tested. The link length parameters of the manipulator are changed and the CMAC neural network performances are observed. The simulation results after 1000 trainings as mentioned above are used to test the system. We change the dimension of the manipulator to: l_1=*0.5 m*; l_2=*0.1 m*; l_3=*0.4 m*; l_4=*0.35 m*; l_5=*0.15 m*. The result is shown in Table 4, from which we can see that the average error caused by the dimension parameter changing has decreased to a stable after about 100 trainings. Also the maximum norm error is about *0.25 mm* after 700 trainings, which is close to the result of *0.3 mm* norm error above. Then we can say that CMAC neural network is quite robust. Because of the limitation on pages, we omit many output figures from the simulation results.

Table 2 Performance of optimization process without CMAC

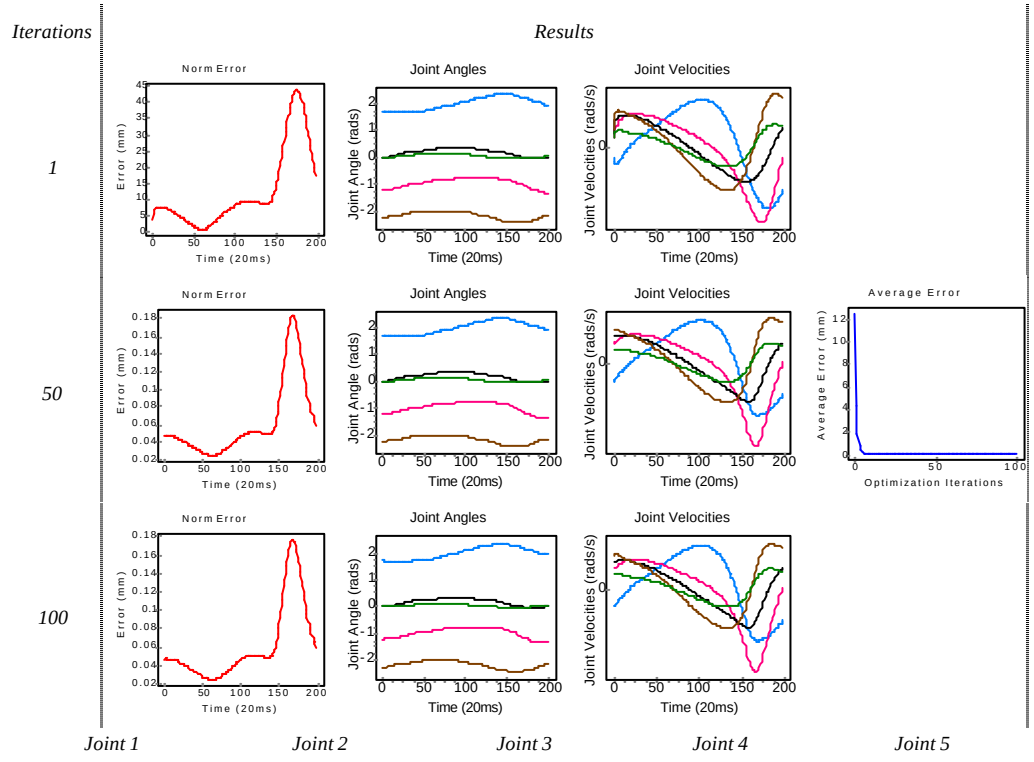


Table 3 CMAC performance during training

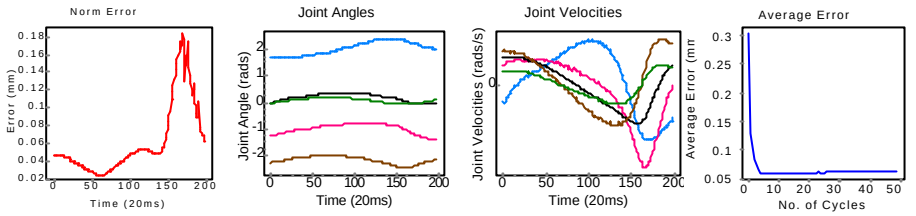
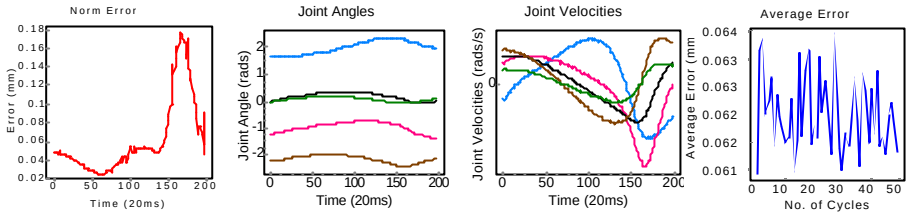
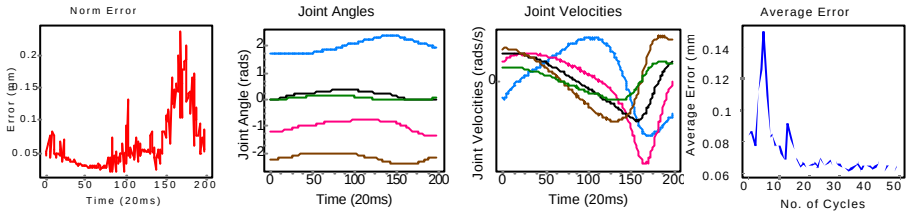
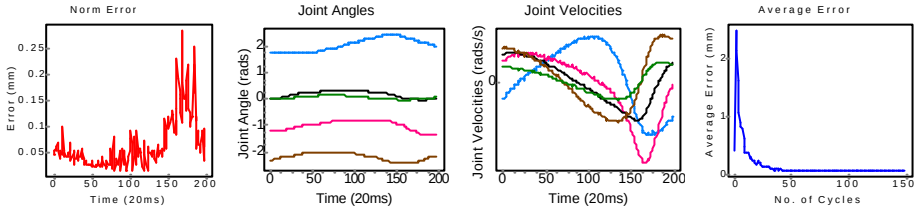
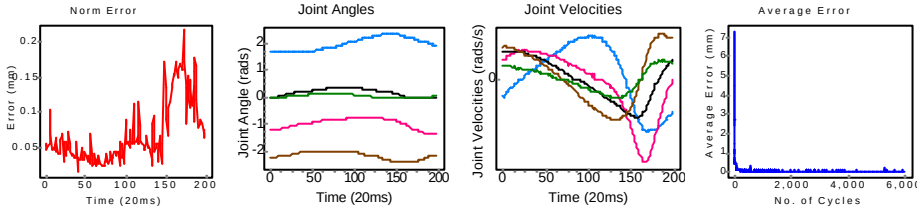
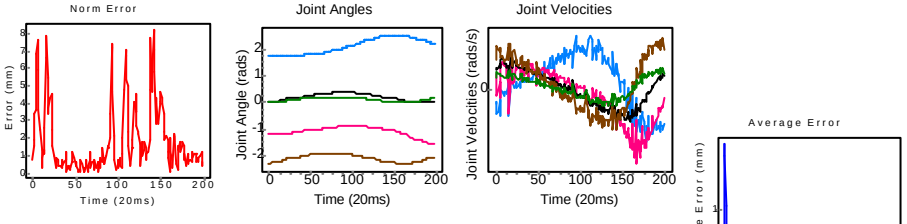
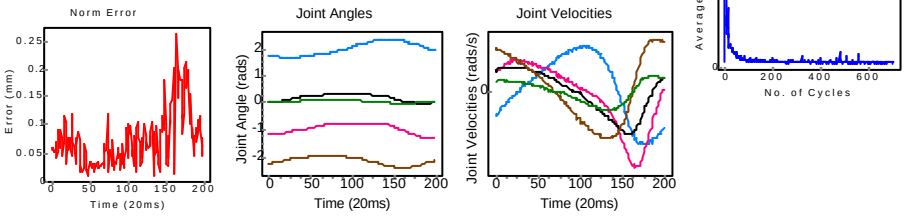
Iterations /h_{CMAC} / Cycles	*Results*
12/1/50	
6/1/50	
3/1/50	
2/1/150	
1/1/350 *1/0.5/200* *1/0.25/5450*	
	Joint 1 *Joint 2* *Joint 3* *Joint 4* *Joint 5*

Table 4 Robustness performance after link lengths changes

Training	*Results*
1st	
700th	
	Joint 1 *Joint 2* *Joint 3* *Joint 4* *Joint 5*

5. CONCLUSION

In this paper the CMAC neural network has been applied in solving the inverse kinematics problem of redundant manipulators. Simulations for a five links manipulator have been performed to evaluate the CMAC performance as well as its robustness characteristics. We can see that CMAC is very good for real time control application due to its fast learning and simple computation properties. Further improvements will be expected in future work, which include increasing the fault-tolerant ability, performing cyclic motions, adding some limitations such as joint angle limits and joint velocity limits, and even obstacle avoidance ability.

6. ACKNOWLEDGMENTS

This work is supported in part by grant RG008/00-01W/LYM/FST and RG025/00-01S/ LYM/ FST from Research Committee of University of Macau.

7. REFERENCES

[1] L. Sciavicco and B. Siciliano, *Modeling and Control of Robot Manipulators*, The McGraw-Hill Companies, Inc., 1996.

[2] L.H. Tsoukalas and R.E. Uhrig, *Fuzzy and Neural Approaches in Engineering*, Wiley Interscience, 1997.

[3] J.A. Freeman and D.M. Skapura, *Neural Networks, Algorithms, Applications, and Programming Techniques*, Addison Wesley, 1991.

[4] N.Y. Zhang and P.F. Yan, *Neural Network and Fuzzy Control*, Tsing Hua University Press, China, 1998.

[5] W.T. Miller III, F.H. Glanz and L.G. Kraft III, CMAC: An associative neural network alternative to backpropagation, *Proc. of the IEEE*, Vol. 78, No. 10, pp. 1561-1567, 1990.

[6] C.J.B. Macnab and G.M.T. D'Eleuterio, Stable, on-line learning using CMACs for neuroadaptive tracking control of flexible-joint manipulators, Proc. of the IEEE Int. Conf. on Robotics & Automation, pp. 511-517, Leuven, Belgium, 1998.

[7] E. Oyama and S. Tachi, Modular neural net system for inverse kinematics learning, Proc. of the IEEE Int. Conf. on Robotics & Automation, pp. 3239-3246, San Francisco, USA, 2000.

[8] J.F. Gardner, A. Brandt and G. Luecke, Applications of neural networks for trajectory control of robots, IEEE 5th Int. Conf. on Advanced Robotics, Vol. 1, pp. 487-492, 1991.

[9] A.P.L. Ferreira and P.M. Engel, Positioning a robot arm: An adaptive neural approach, Proc. of Int. Workshop on Neural Networks for Identification, Control, Robotics, and Signal/ Image Processing, pp. 440-448,1996.

[10] J.A. Walter, PSOM Network: Learning with few examples, Proc. of the IEEE Int. Conf. on Robotics & Automation, pp. 2054-2059, Leuven, Belgium, 1998.

[11] M. Woo, J. Neider and T. Davis, *OpenGL Programming Guide*, 2nd Edition, Addison Wesley Developers Press, 1997.

[12] R. Kempf and C. Frazier, *OpenGL Reference Manual: the official reference document to OpenGL*, version 1.1, Addison Wesley Developers Press, 1997.

[13] W.Th. Miller and F.H. Glanz, *UNH_CMAC, Version 2.1*, The University of New Hampshire Implementation of the Cerebellar Model Arithmetic Computer - CMAC

METODA CMAC NEURALNE MREŽE I PRIMJENA NA KINEMATIÈKU KONTROLU REDUNDANTNOG MANIPULATORA

SAŽETAK

Veæ mnogo godina ispituju se inverzni kinematièki problemi redundantnih manipulatora. Primjena Jacobian Pseudoinverse Algorithm predstavlja konvencionalnu metodu za rješavanje ovog problema, a koja je djelotvorna i sposobna riješiti redundanciju redundantnih manipulatora. Međutim, napor utrošen na izraèun èini ga neodgovarajuæim za kontrolu realnog vremena. Neuralne mreže mnogo se koriste u posljednje vrijeme za kontrolu robota jer su brze, toleriraju greške i sposobne su za uèenje. U ovom radu predstavljamo primjenu CMAC (Cerebellar Model Articulation Controller) neuralne mreže za rješavanje inverznih kinematièkih problema u realnom vremenu. Izvršene su simulacije za manipulator s pet veza s ciljem ocjene rada CMAC neuralne mreže. Raèunalnom simulacijom ustanovili smo da je CMAC metoda posebno pogodna za kontrolu realnog vremena robota i za rješavanje problema aproksimacije nelinearne funkcije.

***Kljuène rijeèi**: inverzni kinematièki problem, metoda neuralnih mreža, redundantni manipulator, kontrola realnog vremena.*