Kinematics control of redundant manipulators using a CMAC neural network combined with a genetic algorithm

Yangmin Li and Sio Hong Leong

Department of Electromechanical Engineering, Faculty of Science and Technology, University of Macau, Av. Padre Tomás Pereira S.J., Taipa, Macau SAR (P.R.China)

(Received in Final Form: March 6, 2004)

SUMMARY
A method is proposed to solve the inverse kinematics and control problems of robot control systems using a cerebellar model articulation controller neural network combined with a genetic algorithm. Computer simulations and experiments with a 7-DOF redundant modular manipulator have demonstrated the effectiveness of the proposed method.

KEYWORDS: Redundant manipulators; Kinematics; CMAC neural networks

1. INTRODUCTION
The design of a redundant manipulator involves an inverse kinematics problem in determining the joint variables corresponding to any desired end-effector position and orientation. In order to control the manipulator, the joint trajectories must be found which yield the desired end-effector trajectory. Since a redundant manipulator has more degrees of freedom (DOF) than necessary, the solution is not unique. The Jacobian pseudoinverse algorithm has been widely used for solving such inverse kinematics problems, because it is able to satisfy additional constraints through mapping the velocities corresponding to the additional constraints onto the null space of the Jacobian while tracking the desired workspace trajectory. However, the inversion of the Jacobian matrix causes difficulties at or near singularities. Neural networks (NNs) have been widely applied in robotic control in recent years. They are normally composed of many neurons, which make them robust and fault-tolerant. Since NNs are able to solve highly nonlinear problems, they are considered very promising for application to robotic control problems. Among the many kinds of NNs, CMAC (Cerebellar Model Articulation Controller) NNs will be discussed in this paper.

A CMAC NN is an associative NN 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 a CMAC assures local generalization. Although a CMAC is based on a linear representation of neurons in the cerebellum, in operation the representation behaves nonlinear. Their simple learning method and fast convergence make CMAC NNs suitable for real-time control.

Genetic Algorithms (GAs) are another powerful tool for engineering optimization. They are non-deterministic search algorithms based upon the evolution of successive populations of individuals. Under some external pressure, a new population is deduced from the previous one. From generation to generation, the populations, pushed by an external pressure which is the function to be optimized, migrate through the variable space under investigation based on the mechanisms of selection, recombination, reproduction and mutation. This migration stops when either convergence is reached, or after a specified maximum number of generations.

Previous research has now suggested a new method to solve the inverse kinematics problem for a redundant manipulator using a CMAC NN combined with a GA. The CMAC NN is used for learning the system, and the GA is then used to optimize the objective function corresponding to the inverse kinematics problem. A CMAC NN can provide useful data for enhancing the performance of the GA.

2. CONVENTIONAL MANIPULATOR INVERSE KINEMATICS
The position \( x \) of the end-effector of a kinematically redundant manipulator is a function of the joint variable \( \theta \).

\[
   x = f(\theta)
\]  

The differential equation for the kinematics of the system can be expressed as:

\[
   \dot{x} = J(\theta)\dot{\theta}
\]

where \( \dot{x} \) is the \( m \times 1 \) end-effector velocity vector, \( \dot{\theta} \) is an \( n \times 1 \) joint velocity vector, and \( J(\theta) \) is the \( m \times n \) Jacobian matrix. Solving for \( \dot{\theta} \), we have:

\[
   \dot{\theta} = J^+(\theta)\dot{x}
\]

and

\[
   J^+ = J^T(JJ^T)^{-1}
\]

where \( J^+ \) is the pseudoinverse matrix of the Jacobian matrix that locally minimizes the norm of joint velocities. Furthermore, the joint velocity can be expressed as

\[
   \dot{\theta} = J^+(\theta)\dot{x} + (I - J^+J)\dot{q}_a
\]

where \( \dot{q}_a \) is a vector of arbitrary joint velocities projected in the null space of \( J \). The redundancy can be resolved by specifying \( \dot{q}_a \) so as to satisfy an additional constraint.

The above pseudoinverse formulation involves the inverse of \( J \) and is computationally demanding. As such, it is not suitable for real-time control.
3. CMAC NEURAL NETWORKS

The structure of a CMAC NN is shown in Fig. 1. In the input space, S are the sensors in the real world, which can be any possible input vectors. The S are mapped into C points in the conceptual memory A. “Close” inputs have overlaps in A but “far” inputs do not.

Since the input space is very large, A is mapped onto a smaller physical memory A’. As a result, any input to a CMAC will occupy C physical memory locations. The output Y is the summation of the content (weight) of the C locations in A’.

The associative mapping within the CMAC NN assures that nearby points generalize, while distant points do not generalize. Also, since the relationship between A’ and Y is linear, but from S to A’ is nonlinear, the nonlinear nature of a CMAC NN performs a fixed nonlinear mapping from the input vector to a many-dimensional output vector.

The training of a CMAC NN is based on data Y(s) and Yd, where Yd is the desired output corresponding to the input S, using the least mean square (LMS) training rule. The weights are updated with the following equation:

\[ w(t + 1) = w(t) + \eta (Y_d - Y(s))/C \]  

where \( \eta \) is the learning step length.

4. GENETIC ALGORITHMS

The main difficulty in optimization lies in identifying the feasible region, particularly the subspace of the feasible region where the constraints are satisfied. The feasible region can be any shape, and can be neither convex nor concave. The most widely used method to treat constraints is to incorporate them in the objective function.

Genetic algorithms (GAs) are adaptive methods which can be used to solve optimization problems by applying the evolutionary processes of biological organisms. Over many generations, natural populations evolve according to the principles of natural selection and “survival of the fittest”. The power of GAs comes from the fact that the technique is robust, and can deal successfully with a wide range of problem areas, including those which are difficult for other methods to solve. GAs are not guaranteed to find the global optimum solution to a problem, but they are generally good at finding “acceptably good” solutions to problems “acceptably quickly”.

GAs embody a direct analog of natural behavior. They work with a population of “individuals”, each representing a possible solution to a given problem. Each individual is assigned a “fitness score” according to how good a solution to the problem it is. The fitness function is usually a weighted sum of the original objective function minus some penalty for every constraint violation. The highly fit individuals are given opportunities to “reproduce”, by “cross breeding” with other individuals in the population. This produces new individuals as “offspring”, which share some features taken from each “parent”. The least fit members of the population are less likely to get selected for reproduction and so “die out”. Fig. 2 shows a general GA process.
CMAC NNs are capable of learning and accumulating knowledge about a system and providing an approximate solution. However, training data must first be provided from which the GA can “learn”. It is usually preferred that the training be done online without affecting the system’s performance. On the other hand, since a manipulator usually has joint limits (maximum and minimum joint position, joint velocities and joint accelerations etc.) the nonlinear constrained inverse kinematics problem must be solved to provide data for training the CMAC NN. As a result, for online training and good solutions even during the training of a CMAC NN, an optimization tool is required which can handle constraints. In order to combine well with CMAC NNs, the optimization tool chosen must be iterative. The number of iterations can then be reduced or the accuracy can be improved for a fixed number of iterations as the CMAC NN is trained. Among the different optimization methods, the GA is one of the most powerful tools for solving nonlinear constrained problems and promises to satisfy all the requirements.

CMAC NNs simplify complicated analytical models, decreasing the amount of calculation, which makes them faster than other approaches for real-time control. They are robust and fault tolerant. On the other hand, a GA has a simple structure. They are also accurate given sufficient time for generation, but it is time consuming to allow them to evolve for high accuracy. GAs are able to find multiple local minima, and sometimes the global minimum.

As a result, CMAC NNs combined with a GA can be used directly without previous offline training, because the GA is responsible for the providing acceptable solutions in the beginning stages. As time passes, CMAC NNs can accumulate experience and the performance of the system can be enhanced due to the increasing importance of the CMAC NN’s output.

6. SIMULATIONS
The following simulations were developed using Borland C++ Builder software in a Microsoft Windows 98 environment. Real number representation was used for the GA, which is a modification of the traditional binary coded GA. Both planar and spatial manipulators were investigated.

6.1. Simulation of a 5-DOF planar manipulator
A 5-link (5-DOF) planar manipulator connected with revolute joints \((m = 2, n = 5, \text{redundancy} = n - m = 3)\) was simulated, as shown in Fig. 3. From Fig. 4, the Denavit-Hartenberg notation for the manipulator is derived in Table I.

The lengths of the links, joint positions and velocity limits, as well as the initial positions, are shown in Table II.

The relation between the position of the end-effector and the joint positions is:

\[
\begin{bmatrix}
X \\
Z
\end{bmatrix} = \begin{bmatrix}
l_1S_1 + l_2S_{12} + l_3S_{123} + l_4S_{1234} + l_5S_{12345} \\
l_1C_1 + l_2C_{12} + l_3C_{123} + l_4C_{1234} + l_5C_{12345}
\end{bmatrix} (7)
\]

where \(l_1, l_2, l_3, l_4, l_5\) are the lengths of the links; \(\theta_1, \theta_2, \theta_3, \theta_4, \theta_5\).

Table I. Denavit-Hartenberg Parameters of a 5-DOF Manipulator.

<table>
<thead>
<tr>
<th>Link</th>
<th>(a_i)</th>
<th>(a_i)</th>
<th>(d_i)</th>
<th>(\theta_i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>(l_1)</td>
<td>0</td>
<td>0</td>
<td>(\theta_1 - 90^\circ)</td>
</tr>
<tr>
<td>2</td>
<td>(l_2)</td>
<td>0</td>
<td>0</td>
<td>(\theta_2)</td>
</tr>
<tr>
<td>3</td>
<td>(l_3)</td>
<td>0</td>
<td>0</td>
<td>(\theta_3)</td>
</tr>
<tr>
<td>4</td>
<td>(l_4)</td>
<td>0</td>
<td>0</td>
<td>(\theta_4)</td>
</tr>
<tr>
<td>5</td>
<td>(l_5)</td>
<td>0</td>
<td>0</td>
<td>(\theta_5)</td>
</tr>
</tbody>
</table>

Table II. Dimensions, joint and velocity limits and initial configuration of the 5-DOF Manipulator.

<table>
<thead>
<tr>
<th>No.</th>
<th>Link Length (m)</th>
<th>Joint Position (deg.)</th>
<th>Joint Velocity (deg./s)</th>
<th>Initial Joint Position (deg.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5</td>
<td>150 -90</td>
<td>57.3 -57.3</td>
<td>99.1</td>
</tr>
<tr>
<td>2</td>
<td>0.25</td>
<td>90 -90</td>
<td>57.3 -57.3</td>
<td>-69.9</td>
</tr>
<tr>
<td>3</td>
<td>0.25</td>
<td>90 -90</td>
<td>57.3 -57.3</td>
<td>0.0</td>
</tr>
<tr>
<td>4</td>
<td>0.275</td>
<td>90 -150</td>
<td>57.3 -57.3</td>
<td>-131.8</td>
</tr>
<tr>
<td>5</td>
<td>0.225</td>
<td>90 -90</td>
<td>57.3 -57.3</td>
<td>0.0</td>
</tr>
</tbody>
</table>
\[ \theta_5 \text{ are the joint positions; } S_1 = \sin(\theta_1); \ S_2 = \sin(\theta_1 + \theta_2); \]
\[ S_{123} = \sin(\theta_1 + \theta_2 + \theta_3); \ S_{1234} = \sin(\theta_1 + \theta_2 + \theta_3 + \theta_4); \]
\[ S_{12345} = \sin(\theta_1 + \theta_2 + \theta_3 + \theta_4 + \theta_5); \]

\[ C_1 = \cos(\theta_1); \ C_{12} = \cos(\theta_1 + \theta_2); \]
\[ C_{123} = \cos(\theta_1 + \theta_2 + \theta_3); \]
\[ C_{1234} = \cos(\theta_1 + \theta_2 + \theta_3 + \theta_4); \]
\[ C_{12345} = \cos(\theta_1 + \theta_2 + \theta_3 + \theta_4 + \theta_5). \]

The relation between the velocity of the end effector and the joint velocity is:

\[ \dot{x} = J(\theta) \dot{\theta} \]

where

\[
J(\theta) = \begin{bmatrix}
\frac{\partial X}{\partial \theta_1} & \frac{\partial X}{\partial \theta_2} & \frac{\partial X}{\partial \theta_3} & \frac{\partial X}{\partial \theta_4} & \frac{\partial X}{\partial \theta_5} \\
\frac{\partial Z}{\partial \theta_1} & \frac{\partial Z}{\partial \theta_2} & \frac{\partial Z}{\partial \theta_3} & \frac{\partial Z}{\partial \theta_4} & \frac{\partial Z}{\partial \theta_5}
\end{bmatrix}
\]

and

\[
\begin{align*}
\frac{\partial X}{\partial \theta_1} &= l_1 C_1 + l_2 C_{12} + l_3 C_{123} + l_4 C_{1234} + l_5 C_{12345} \\
\frac{\partial X}{\partial \theta_2} &= l_2 C_{12} + l_3 C_{123} + l_4 C_{1234} + l_5 C_{12345} \\
\frac{\partial X}{\partial \theta_3} &= l_3 C_{123} + l_4 C_{1234} + l_5 C_{12345} \\
\frac{\partial X}{\partial \theta_4} &= l_4 C_{1234} + l_5 C_{12345} \\
\frac{\partial X}{\partial \theta_5} &= l_5 C_{12345} \\
\frac{\partial Z}{\partial \theta_1} &= -l_1 S_1 + l_2 S_{12} + l_3 S_{123} + l_4 S_{1234} + l_5 S_{12345} \\
\frac{\partial Z}{\partial \theta_2} &= -l_2 S_{12} + l_3 S_{123} + l_4 S_{1234} + l_5 S_{12345} \\
\frac{\partial Z}{\partial \theta_3} &= -l_3 S_{123} + l_4 S_{1234} + l_5 S_{12345} \\
\frac{\partial Z}{\partial \theta_4} &= -l_4 S_{1234} + l_5 S_{12345} \\
\frac{\partial Z}{\partial \theta_5} &= -l_5 S_{12345}
\end{align*}
\]

The desired end-effector trajectory in this simulation is a circle with radius of 0.25 m and centered at (0.5 m, 0.6 m). Angular velocity \( \omega \) is 0.5 \( \pi \) rad/s, and thus the cycle time is 4 seconds. The trajectory equations are as follows:

\[
\begin{align*}
X_d(t) &= 0.5 - 0.25 \cos(\omega t) \\
Z_d(t) &= 0.6 + 0.25 \sin(\omega t)
\end{align*}
\]

6.2. Simulation of a 5-DOF planar manipulator with a CMAC NN

In order to avoid the effort involved in training the CMAC NN before use, the network was trained during operation using the control diagram shown in Fig. 5. This is online training.

In the diagram above, the “Gradient” is an optimizer using the descent gradient method, a good tool for local minimization of unconstrained problems. Usually it requires some iterations before reaching the minimum. In this simulation, the optimizer iterated only once in order to reduce the time required. The gradient method was applied to increase the accuracy of the system output at the beginning in order to compensate for the untrained CMAC NN.

The structure of the CMAC NN is shown in Fig. 6, with a CMAC NN before use, the network was trained during operation using the control diagram shown in Fig. 5. This is online training.

The objective function being optimized by the descent gradient method is:

\[
U = \frac{\alpha}{2} e^T e + \frac{\beta}{2} \dot{\theta}^T \dot{\theta}
\]

where \( e \) is the position error vector, \( \dot{\theta} \) is joint velocity vector, \( \alpha \) and \( \beta \) are the weight constants.

![Fig. 5. Block diagram of a CMAC NN using the gradient method.](image)

Fig. 5. Block diagram of a CMAC NN using the gradient method.

![Fig. 6. Input and output of the CMAC neural network.](image)

Fig. 6. Input and output of the CMAC neural network.
Fig. 7. Objective function value for the first 100 trainings.

The second term on the right is for minimizing the joint velocity, thus smoothing the motion as well as minimizing energy consumption. Then

$$\frac{\partial U}{\partial \dot{\theta}} = -\alpha e^T J \Delta t + \beta \dot{\theta}^T$$  \hspace{1cm} (11)

where $J$ is the Jacobian matrix and thus,

$$\Delta \theta_{k+1} = \Delta \theta_{k} - \eta \frac{\partial U}{\partial \Delta \theta}$$  \hspace{1cm} (12)

where $k$ is the number of iterations and $\eta$ is the length of a learning step.

In this simulation, the number of vectors in the CMAC memory was 10,000. The generalization parameter $C$ was 64, and the training step length is 1. The sampling time $\Delta t$ was 0.02s, with the constants $k = 1$, $\alpha = 1$, $\beta = 1$, $\eta = 0.003$.

In the simulation, the average value of the objective function was used as the performance index.

$$Obj = \frac{1}{(t/\Delta t)} \sum_{j=1}^{(t/\Delta t)} \left[ \sqrt{\sum (X_d - X)^2} + \sum_{i=1}^{n} \theta_i^2 \right]$$  \hspace{1cm} (13)

where $n$ is the number of joints. After training 100 iterations, from Fig. 7 it is clear that the average value of the objective function decreased dramatically and then stabilized at 0.7. It is obvious that the CMAC NN was “learning” during the operation, and helped to reduce the objective function value gradually. It can generate a better initial value for the gradient optimization process.

The length of the links was then changed so that $l_1 = 0.5$ m, $l_2 = 0.3$ m, $l_3 = 0.2$ m, $l_4 = 0.3$ m, $l_5 = 0.2$ m. Also, joints 2 and 4 were locked, as shown in Fig. 8.

During another 100 training runs, Fig. 9 shows that the objective function again decreased dramatically and stabilized. This demonstrates that the CMAC NN delivered a fast response and was tolerant to the changes.

6.3. Simulation of a CMAC NN combined with a GA

The GA is a well known tool for finding optimal solutions to constrained nonlinear problems. Here, a GA was applied to the inverse kinematics problem of the redundant manipulator shown in Fig. 10.

The GA used was a steady state GA with overlapping populations with an adjustable amount of overlap. The algorithm creates a population of individuals by “cloning” its initial “genome” or population of solutions. In each generation, the algorithm creates a temporary population of individuals, adds these to the previous population, then removes the worst individuals in order to return the population to its original size. Here, 0.5 was used as the replacement percentage. The number of generation used was 20. The population size was 10. Uniform mutations were applied, with probability of 0.25, and the crossover method was arithmetic crossover with a probability of 0.9. Roulette wheel selection and uniform initialization were applied. The objective function for minimization was the same as equation (10), with $\alpha = \beta = 2$.

Using the GA as the optimizer facilitated handling physical constraints such as the joint position limits and joint velocity limits. In order to satisfy the constraints, the search space of the GA was changed from time to time, depending on the current joint positions.

Since

$$\theta_l \leq \theta_{(t+1)} \leq \theta_u$$  \hspace{1cm} (14)

$$\dot{\theta}_l \leq \dot{\theta}_{(t+1)} \leq \dot{\theta}_u$$  \hspace{1cm} (15)
Then
\[
\dot{\theta}(t+1) = \frac{\dot{\theta}_l(t+1) - \dot{\theta}_l(t)}{\Delta t}
\]  
(16)

and
\[
\max \left\{ \dot{\theta}_l, \frac{\dot{\theta}_l - \dot{\theta}_u}{\Delta t} \right\} \leq \dot{\theta}(t+1) \leq \min \left\{ \dot{\theta}_u, \frac{\dot{\theta}_u - \dot{\theta}_l}{\Delta t} \right\}
\]
(17)

where \(\dot{\theta}_l(t)\) is the joint position at time \(t\); \(\dot{\theta}_l\) and \(\dot{\theta}_u\) are the lower and upper limits of the joint position; \(\dot{\theta}_l\) and \(\dot{\theta}_u\) are the lower and upper limits of the joint velocity; and \(\dot{\theta}(t+1)\) is the joint velocity at time \(t+1\).

6.4. Simulation of an integrated modular CMAC NN combined with a GA and a descent gradient optimizer

When combining the CMAC NN and GA into a new control system, it is important to keep the gradient optimizer to ensure that the system can at least find a solution near the local minimum. Beyond that, there is always hope of finding a better solution among the other local minima, and maybe the global minimum. In online training, it is important to keep the error minimal and the system stable, even at the beginning.

In the simulations, the initialization of the GA was no longer uniform. The output of the descent gradient optimizer was used as the initial population for the GA. The aim was to provide the best possible initial population for the GA in order to get a better result with the same number of generations and same population size.

The average objective function values during the first 100 iterations are shown in Fig. 11. Clearly, the value stabilized after a few training runs.

In the simulations, the initialization of the GA was no longer uniform. The output of the descent gradient optimizer was used as the initial population for the GA. The aim was to provide the best possible initial population for the GA in order to get a better result with the same number of generations and same population size.

The average objective function values during the first 100 iterations are shown in Fig. 11. Clearly, the value stabilized after a few training runs.

Fig. 10. Block diagram of a CMAC NN combined with a GA and a descent gradient optimizer.

Fig. 11. Performance of a CMAC NN with a GA and descent gradient optimizer over 100 iterations.

Fig. 12. Control block diagram of an integrated modular CMAC NN combined with a GA and a descent gradient optimizer.

A proposed modular CMAC NNs enhanced with a GA and a descent gradient optimizer is shown in Fig. 12. The CMAC NN with a memory size of 10,000 has been divided into 5 smaller CMAC NNs, each with memory size of 2000. The modular CMAC NNs are proposed in order to provide several solutions for the later optimization process. Since more starting points for optimization can lead to a better solution, using a modular CMAC NN will provide an advantage at the expense of more calculation effort.

The optimizer which has been added is used for improving the performance of the system further, and at the same time
providing different arrangements of the order in which the optimization proceeds. It can also provide different data for training the CMAC NNs. As a result, the output from the different CMAC NNs will be different, and the probability of finding better solutions will be increased.

The results of the first 100 operations are shown in Fig. 13. The average objective function value is lower than that of the previous systems.

Changes were again made to the length of the links of the manipulator as shown in Fig. 8, and joints 2 and 4 were again locked. After another 100 iterations, the results are shown in Fig. 14.

Finally, the long-term performance of the proposed system was simulated. Since the system will be trained during its working life in response to changes, it is important to confirm that the system is stable if it is being trained continuously. The system was trained over the course of 10,000 iterations, and the results are shown in Fig. 15.

The average objective function value was very stable throughout the long-term operation. So such a system can be trained over its lifetime preparing to respond to joint failure or other such changes.

6.5. Simulation of spatial manipulators

So far, the simulations discussed have been 2-dimensional. The simulation will now be extended to a spatial manipulator to see whether the system performance resembles to that of the planar manipulator.

A 7-DOF spatial manipulator was exploited for the simulations and experiments. The manipulator and its initial configuration are shown in Fig. 16. The length of the

![Fig. 16. A 7-DOF spatial manipulator.](image)
Table III. Link length, maximum and minimum joint positions, velocities and accelerations and initial joint positions of the 7-DOF spatial manipulator.

<table>
<thead>
<tr>
<th>No.</th>
<th>Link Length (m)</th>
<th>Joint Position (deg.)</th>
<th>Joint Velocity (deg./s)</th>
<th>Joint Acceleration (deg./s²)</th>
<th>Initial Joint Position (deg.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.155</td>
<td>[-150, 150]</td>
<td>[-14, 14]</td>
<td>[-590, 590]</td>
<td>7.0</td>
</tr>
<tr>
<td>2</td>
<td>0.181</td>
<td>[-90, 90]</td>
<td>[-14, 14]</td>
<td>[-590, 590]</td>
<td>23.5</td>
</tr>
<tr>
<td>3</td>
<td>0.205</td>
<td>[-150, 150]</td>
<td>[-14, 14]</td>
<td>[-590, 590]</td>
<td>6.5</td>
</tr>
<tr>
<td>4</td>
<td>0.138</td>
<td>[-150, 150]</td>
<td>[-14, 14]</td>
<td>[-590, 590]</td>
<td>7.3</td>
</tr>
<tr>
<td>5</td>
<td>0.134</td>
<td>[-90, 90]</td>
<td>[-14, 14]</td>
<td>[-590, 590]</td>
<td>27.0</td>
</tr>
<tr>
<td>6</td>
<td>0.100</td>
<td>[-90, 90]</td>
<td>[-24, 24]</td>
<td>[-980, 980]</td>
<td>49.3</td>
</tr>
<tr>
<td>7</td>
<td>0.162</td>
<td>[-150, 150]</td>
<td>[-35, 35]</td>
<td>[-1400, 1400]</td>
<td>2.8</td>
</tr>
</tbody>
</table>

links, maximum and minimum joint positions, velocities and accelerations are listed in Table III. From Fig. 17, we can derive the Denavit-Hartenberg notation for the manipulator shown in Table IV.

In the simulation, the manipulator was required to follow the trajectory of a straight line in space at a linear velocity $V$ of 0.02 m/s and with a cycle time of 20 s, as expressed in the following equations:

$$
\begin{align*}
X_d(t) &= 0.1 + 0.02t \\
Y_d(t) &= 0.1 + 0.02t \\
Z_d(t) &= 1.17 - 0.02t 
\end{align*}
$$

The sampling rate was 200 ms. The parameters of the CMAC NN and the GA were identical with those used in the planar simulation. The control block diagram is shown in Fig. 12. The system was tested for 100 iterations tracking the spatial line, and the results are shown in Fig. 18. Clearly, the objective function decreased with more iterations and became stable. The system’s long-term operational performance is shown in Fig. 19 for 10,000 iterations. Again, the objective function was stable throughout the long-term operation.

The length of links 4, 5 and 7 were then changed to be 0.172 m, 0.162 m and 0.1 m respectively. Moreover, joints 3, 5 and 7 were locked. The results during the first 100 operations after the changes were made are presented graphically in Fig. 18.
Fig. 19. Long term operational performance of the simulated 7-DOF spatial manipulator.

Fig. 20. Performance for the simulated 7-DOF spatial manipulator during the first 100 iterations after changes were made.

in Fig. 20. The control system clearly was tolerant and responded quickly to the changes. The simulation results for the spatial manipulator were thus similar to those with the planar manipulator.

7. EXPERIMENT
The proposed control system was then tested with the real 7-DOF manipulator shown in Fig. 21. Its structure, joint limitations and initial configuration are identical to those of the spatial manipulator in the previous simulation. The target trajectory was again the same line in space as in the simulation. The sampling rate in the experiment was 250 ms.

The manipulator was controlled through the same program used in the simulation. An interface card maintained communication between the computer and the manipulator. The library used for controlling the manipulator was provided by its manufacturer. The control system was changed as shown in Fig. 22 so that the calculated desired joint velocities were input to the manipulator and the outputs were the resulting joint positions provided by its sensors.

Since there are also acceleration limits in the control of a real manipulator, the search space of the GA must be changed from time to time in order to satisfy the constraints on joint positions and velocities:

\[
\dot{\theta}_l \leq \dot{\theta}_{(t+1)} \leq \dot{\theta}_u \\
\ddot{\theta}_u \leq \ddot{\theta}_{(t+1)} \leq \ddot{\theta}_u \leq \frac{2(\theta_{(t+1)} - \theta_l - \dot{\theta}_l \Delta t)}{\Delta t} + \dot{\theta}_l \leq \frac{2(\theta_u - \theta_l - \dot{\theta}_l \Delta t)}{\Delta t}
\]

\[
\dot{\theta}_{(t+1)} = \dot{\theta}_i + \frac{2(\theta_{(t+1)} - \theta_i - \dot{\theta}_i \Delta t)}{\Delta t} \leq \dot{\theta}_{(t+1)}
\]

Fig. 21. The 7-DOF spatial manipulator.
where $\theta_l$ and $\theta_u$ are the joint position lower and upper limits respectively; $\dot{\theta}_l$ and $\dot{\theta}_u$ are the joint velocity lower and upper limits, respectively; $\ddot{\theta}_{\text{max}}$ and $\ddot{\theta}_{\text{min}}$ are the joint maximum and minimum accelerations; $\theta(t)$ and $\theta(t+1)$ are the joint positions at times (t) and (t + 1); $\dot{\theta}(t+1)$ and $\ddot{\theta}(t+1)$ is the joint velocity and acceleration at time (t + 1).

The results after 100 training runs with the spatial line trajectory are shown in Fig. 23. The average objective function value was quite stable, but there were some fluctuations. Although the objective function has no obvious decreasing tendency as more training accumulates, the general performance was good. The result is quite different from that of the simulation, because in the simulation many factors were idealized, but in the experiment these factors could not be predicted or treated accurately. For example, the real manipulator remained in motion during the control system’s calculation period. This calculation time had been idealized as zero in the simulation. The experimental results show the need for some estimation of calculation time in the system. Moreover, the actual joint motion always accelerated at the beginning of each motion and decelerated at the end. This too had not been taken into account during the simulation. However, the system still performed well in solving the inverse kinematics problems, since the objective value was low and quite stable.

An experiment was then performed to test the system’s fault tolerance. Again, joints 3, 5 and 7 were locked, and another 100 training runs were conducted to see how the system would respond to the changes. The result is shown in Fig. 24. The resulting average objective function resembled that generated in the previous experiment. The value was quite stable with some fluctuation. But the fluctuations seemed to stabilize as more training accumulated. Again, the result was quite different from that of the simulation. This is because different environmental factors led to different training of the system and so as the different outputs to deal with the outside factors. In this case, the manipulator seems to have had difficulty following the trajectory. However, the system still reacted stably to the changes.

8. CONCLUSIONS
Traditionally, analytical methods are applied in solving the inverse kinematics problems of redundant manipulators. In this study, another method has been proposed using a CMAC NN with a GA. The simple structure, high speed and simple learning rule of the CMAC NN, plus the power of the GA method in solving constrained nonlinear optimizations, make the proposed method applicable to these inverse kinematics problems and so to the real time control
of manipulators. Simulations of both a planar redundant manipulator and a spatial redundant manipulator have been performed. Experiments have been conducted using a 7-DOF modular redundant manipulator. The results show that the control method is effective in on-line control. Further study is planned on the application of CMAC NNs and GAs in the field of dynamic control of redundant manipulators.

Acknowledgements
This work was supported in part by grant RG009(2)/02-03S/LYM/FST from the Research Committee of the University of Macau.

References