'89 KACC 1989. 10. 27~28

A Neural Network Architecture for Dynamic Control of Robot Manipulators

°Yeon-Sik Ryu and Se-Young Oh

Department of Electrical Engineering
Pohang Institute of Science and Technology

Neural network control has many innovative potentials for intelligent adaptive control. Among many, it promises real time adaption, robustness, fault tolerance, and self-learning which can be achieved with little or no system models.

In this paper, a dynamic robot controller has been developed based on a backpropagation neural network. It gradually learns the robot's dynamic properties through repetitive movements being initially trained with a PD controller. Its control performance has been tested on a simulated PUMA 560 demonstrating fast learning and convergence.

1 Introduction

[1] Most of the neural network control (neurocontrol) applications reported to date have been primarily in kinematic control. In general, kinematic control is quite inadequate for obtaining good dynamic performance and high positioning accuracy with manipulators operating at high speeds under varying operating conditions. Dynamic control is effected by the computed torque technique, but it has many difficulties. First, the nonlinear dynamic model is overly complex, making real time implementation difficult. Besides, an accurate dynamic model is impossible to obtain. Adaptive control schemes do not require a priori knowledge of robot dynamics. However, its major drawbacks include excessive computational requirements and high sensitivities to numerical precision and observation noise that tends to increase at an undesirable rate as the number of system state variables increase.

Recently, neurocontrol is drawing considerable attention for the following great potentials:

(Due to Massive Parallelism of Neural Nets)

1) Real-Time Control

2) Fault-Tolerance

3) Robustness Against Noise

(Due to the Learning Capability of Neural Nets)

1) No Need for the Model of the Plant or its Environment

2) Adaptation to Changing Environment

3) Easy Inclusion of Sensor Data Fusion

4) Constant Performance Improvement Through Learning

5) No Need for the Knowledge On Difficult Control Laws

6) Simple Programming

During the past three years, considerable amount of neurocontrol research has been directed to inverse kinematics[1,2,3]. However, little research has been reported on dynamic control except a very few. Miller applied CMAC(*Cerebellar Model Articulation Controller*) to a visual servo problem for a GE P-5 robot[4] and also to a dynamic control simulation of a 2 d.o.f manipulator[5]. It used a huge lookup table even for two degrees of freedom. The training was done in a supervised learning mode.

Kawato et al.[6,7] used a single layer neural network to solve the inverse dynamics problems. A feedback error learning scheme was used to train the single layer of weights that linearly combine many terms in the known dynamic model. As such, it can not be used to a system with little or no knowledge on its model. This is somewhat similar in concept to Pao's functional link network[8].

In this paper, we use BP neural network to control the dynamics of PUMA 560 (in case *3 degree of freedom*). Neural network input is composed of observed states and desired accelerations after 5 sampling time. BP neural network is used to predict the joint torques required to follow a desired angle position after $5dt$, and these torques are used as feedforward terms in parallel with a linear feedback controller. BP neural network learning is implemented by linear feedback controller.

[1]This research was supported by a grant from the Korean Science Foundation and also in part by a grant from the Research Institute of Industrial Science & Technology.

2 Proposed Control Architecture

Figure 1 shows a feedforward neural net controller in parallel with a PD linear controller the sum of which provide the commanded joint torques for a robot manipulator. The Motion Design block generates the ideal trajectory as well as the commanded trajectory as input to the neural net. The commanded trajectory $S_d(t)$ is a linear combination of the actual observed trajectory and the ideal trajectory. The BP network is trained in an unsupervised mode such that the feedback error is directly used to adjust the weights. Kawato calls this feedback error learning.

In control problem, the joint torques are a nonlinear mapping of all the joint positions, velocities, and accelerations.

$$\tau = f(\theta, \dot{\theta}, \ddot{\theta}) \tag{1}$$

Funahashi[9] proved that any continuous mapping can be approximately realized by multilayer neural networks with at least one hidden layer whose output functions are sigmoidal - *nondecreasing, differentiable.*

In our BP network, the input layer and the output PEs act as linear neurons while the hidden layer PEs are sigmoidal.

2.1 Input to the Neural Network

The input to the neural network is the commanded joint trajectory which is a concatenation of the observed joint position, velocity, and the commanded joint acceleration which, if held constant for the next five sample times, would intercept the ideal trajectory after five sample times. This is essentially what Miller used in [5]. Kawato uses the ideal trajectory for the joint torques, however.

2.2 Output from the Neural Network

The net input to a Processing Element (PE) j is

$$net_j = \sum_{i=1}^{n} w_{ji} o_i \tag{2}$$

where w_{ji} is the weight from PE i to PE j.
The output of a PE j in a hidden layer is

$$o_j = f(net_j) \tag{3}$$

where f is a sigmoidal activation function:

$$o_j = \frac{1}{1 + e^{\frac{-(net_j + b_j)}{b_0}}} \tag{4}$$

The output layer PEs are simple linear neurons as follows:

$$o_j = slope * net_j + bias \tag{5}$$

2.3 BP Training

In training the network, we use the usual backpropagation learning rule (*the generalized delta rule*). However, for on-line training, the network was trained in an unsupervised mode where the PD controller output is used as the error for training. That is, the output layer weights are adjusted through

$$t_k - o_k = k_p * (\theta_{ik}(t) - \theta_{ok}(t)) + k_v * (\dot{\theta}_{ik}(t) - \dot{\theta}_{ok}(t)) \tag{6}$$

where t_k, o_k represent the target and actual values, respectively, and,
θ_{ik} represents the ideal joint angle k and θ_{ok}, the observed joint angle k.
This output error is backpropagated to the inner layers as follows:

$$\delta_j = f_j'(net_j) \sum_k \delta_k w_{kj}. \tag{7}$$

A momentum term was also used in the rule:

$$\Delta w_{ji}(n+1) = \eta \delta_j o_i + \alpha \Delta w_{ji}(n). \tag{8}$$

where the subscript n indexes the presentation number, η is a learning rate constant and α is a momentum constant which determines the effect of past weight changes on the current direction of movement in the weight space. In our simulation, both η and α were 0.1 [10].

The PD controller is used not only to initially train the network but also to correct for minor errors even after the network has been fully trained.
As learning proceeds, the PD error torque constantly decreases thereby reducing the motor power. Eventually, the full joint torques are supplied by the neural network.

3 Simulation Results

In computer simulation, the control architecture as described in the previous section produced the dynamic joint torques for a PUMA 560 robot. The system has been implemented in C on an IBM PC/AT. Notice that even a PC has a good potential for real-time control using neurocontrol methodology.

The dynamics for the PUMA 560 robot is given as:

$$\tau = M(\theta)\ddot{\theta} + V(\theta, \dot{\theta}) + G(\theta) \tag{9}$$

where $M(\theta)$ is the $n \times n$ mass matrix of the manipulator, $V(\theta, \dot{\theta})$ is an $n \times 1$ vector of centrifugal and Coriolis terms, and $G(\theta)$ is an $n \times 1$ vector of gravity terms.

Its dynamics parameters were obtained from Armstrong et al.[11]. The forward dynamics equations for simulation came from Walker and Orin[12].

The fully conected network consists of the input, two hidden, and output layers. The two hidden layers have 35 and 8 PEs respectively. The number of input layer PEs is 14 due to the inclusion of some nonlinear dynamic terms. The number of memory cells to store the connection weights is 2481.

Figure 2 plots the evolution of the RMS (*Root Mean Square*) error during the on-line learning attempts. An attempt is a robot's movement along a complete trajectory for each learning trial and this trial is repeated for the neural network to learn and improve its dynamics. *Table 1* shows the RMS errors for the succession of attempts. The attempt labeled no learn is essentially using the network as has been trained up to the previous attempt and no on-line weight modification takes place. This indicates that after sufficient learning, the network with fixed weights can be used instead of the constant weight adjustment during on-line learning. The parameters used in the simulation are listed in *Table 2.*

Figure 3 plots both the actual and the ideal joint angle trajectories with random neural outputs. In this case, neural network weights take the pseudorandom values between -0.5 and 0.5 and this weight is fixed throughout the trajectory with no learning taking place. The plot essentially shows the PD controller's performance. *Figure 4* exhibits the joint angle trajectories during the first learning attempt and also after 31 attempts. It shows that even after one learning attempt, the neurocontrol learned the arm's dynamics very well. In *Figure 3* and *Figure 4*, the dashed line represents desired trajectory while the solid line represents the actual trajectory. To show the error reduction process, *Figure 5* plots the PD controller outputs. As learning goes on, the role of PD controller is quickly diminished.

3.1 Comparison with a CMAC method

For comparision with the CMAC method [13], a two d.o.f arm has also been simulated and the results are shown in *Figure 6.* It can be seen that the BP method converges faster. Besides, the BP method uses 1294 cells while the CMAC uses 11552 cells. By increasing the number of hidden PEs, even better performance was achieved.

3.2 Loading Effects

In order to demonstrate the neurocontroller's robustness under varying loads, 1 kg load has been added to the end-effector and the same learning process as before was effected. *Figure 7* shows the RMS error reduction process with on-going learning attempts. It quickly adapts to a different load.

4 Conclusion

The use of a multilayer BP networks for the dynamic control of manipulators has been investigated. A PD controller is used for initial learning as well as for fine tuning after learning. Absolutely no model for the robot nor its environment has been assumed for the controller. It demonstrates that the neural network can gradually learn the robot's dynamics and thereby constantly improve its control performance through the repetitive on-line learning process. It also adapts to varying loads and can potentially adapt to the robot's own characteristic changes.

Within a reasonable range of robot speeds, relatively small number of hidden PEs are sufficient to achieve a goal performance. Neurocontrol performance can vary with several parameters such as learning rates, momentum, and the number of hidden PEs. So far, these are chosen rather empirically. It is prospected that the applications of neurocontrol to robotics and automation will grow in the very near future.

References

[1] G. Josin, D. Charney and D. White, "*Robot Control Usiny Neural Networks* ", IEEE Conference on Neural Networks, 1988.

[2] Richard K. Elsley, "*A learning Architecture for control Based on backpropagation Neural Networks* ", International Neural Network Socity 1988.

[3] Dejan J. Sobajic, Jun-Jie Lu and Yoh-Han Pao, "*Intelligent Control of the Intelledex 605T Robot Manipulator* ", IEEE Conference on Neural Networks, 1988.

[4] W. Thomas Miller III, "*Sensor-Based Control of Robotic Manipulators Using a General Learning Algorithm* ", IEEE Journal of Robotics and Automation, Vol. RA-3, No.2, April 1987.

[5] W. Thomas Miller III, Filson H. Glanz, and L. Goldon Kraft III, "*Application of a General Learning Algorithm to the Control of Robotic Manipulators* ", The International Journal of Robotics Research, Vol. 6, No. 2, Summer 1987.

[6] Mitsuo Kawato, Yoji Uno, Michiake Isobe, and Ryoji Suzuki, "*Hierarchical Neural Network Model for Voluntary Movement with Application to Robotics* ", IEEE Control Systems Magazine, April, 1988.

[7] Hiroyuki Miyamoto, Mitsuo Kawato, Tohru Setoyama, and Ryoji Suzuki, "*Feedback Error Learning Neural Network for Trajectory Control of a Robotic Manipulator* ", Neural Networks, Vol. 1, 1988.

[8] Yoh-Han Pao, "*Adaptive Pattern Recognition and Neural Networks* ", Addison Wesley, 1988.

[9] Ken-Ichi Funahashi, "*On the Approximate Realiza-*

tion of Continuous Mappings by Neural Networks ", Neural Networks, Vol. 2, , 1989.

[10] David E. Rumelhart, James L. McClelland , and the PDP Research Group, "*Parallel Distributed Processing* ", MIT Press,Vol. 1, 1986.

[11] Brian Armstrong, Oussama Khatib, and Joel Burdick, "*The Explicit Dynamics Model and Inertial Parameters of the PUMA 560 Arm* ", IEEE Conference on Robotics and Automation, 1986.

[12] M. W. Walker and D. E. Orin, "*Efficient Dynamic Computer Simulation of Robotic Mechanisms* ", Journal of Dynamic Systems, Measurement, and Control, Vol. 104, pp.205-211, September, 1982.

[13] Kwanghee Nam and Tae-Young Kuc, "*An Application of the CMAC to Robot Control* ", The Korean Automatic Control Conference, Vol. 2, 1988.

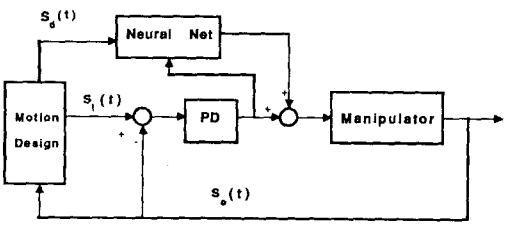


Figure 1. A block diagram of the learning control architecture based on multilayer neural network.

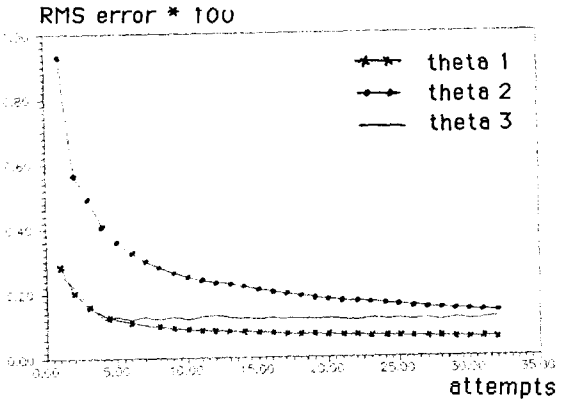


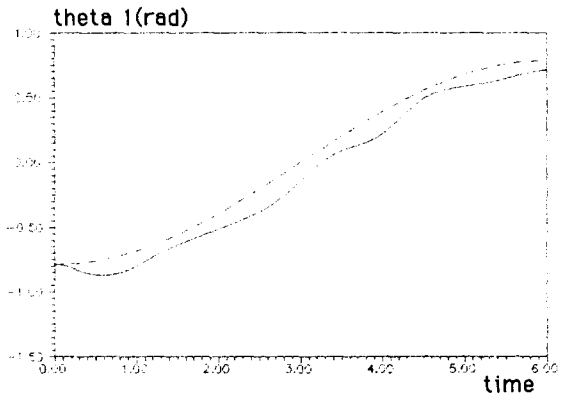
Figure 2. Learning curve for the proposed neural net control architecture.

attempt	RMS (theta1)	RMS (theta2)	RMS (theta3)
1	0.002831	0.009326	0.002632
11	0.000854	0.002389	0.001269
no learn	0.000844	0.002337	0.001321
13	0.000824	0.002285	0.001235
21	0.000678	0.001744	0.001171
no learn	0.000674	0.001720	0.001137
23	0.000666	0.001698	0.001192
31	0.000590	0.001410	0.001159
no learn	0.000588	0.001396	0.001213

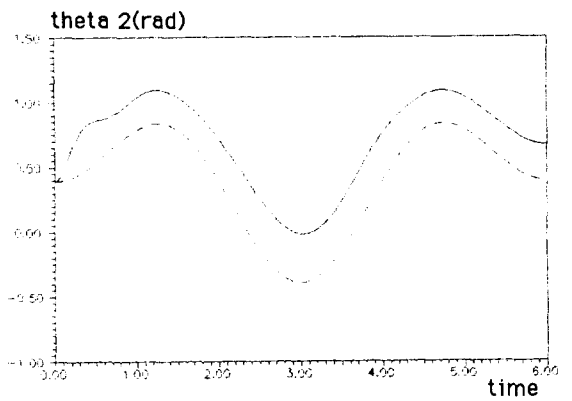
Table 1. *RMS* errors for successive learning attempts.

Parameter	value	Parameter	value
learning rate	0.1	position gain	100.0
momentum	0.1	velocity gain	10.0
slope	0.02	bias	0.5
slope (with payload) : 0.0167			

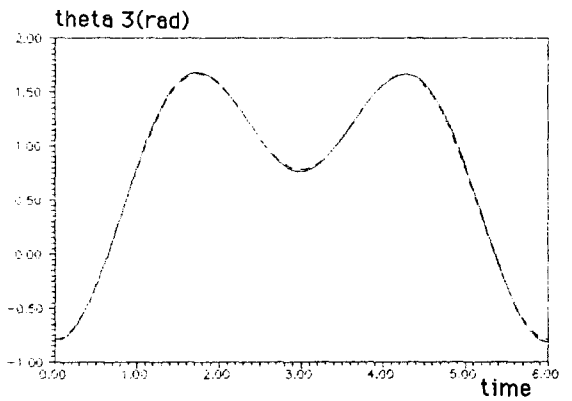
Table 2. Parameters used in simulation.



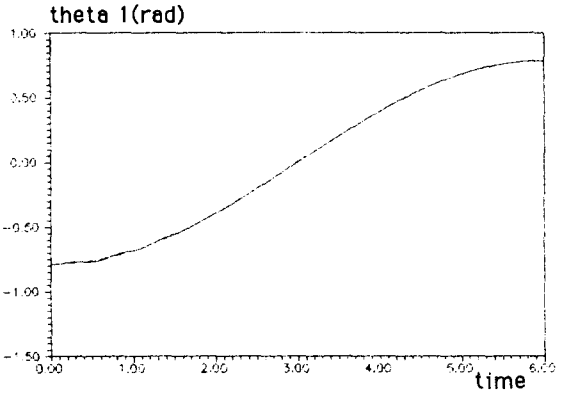
(a)

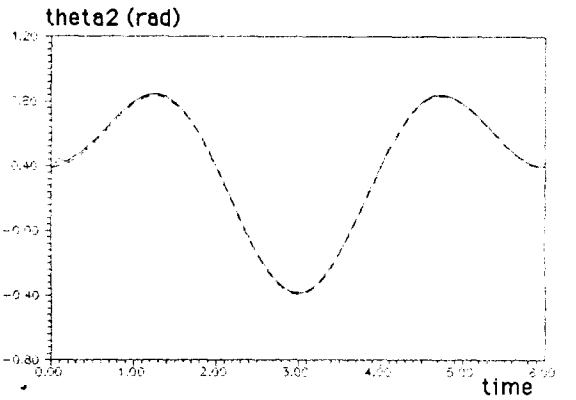


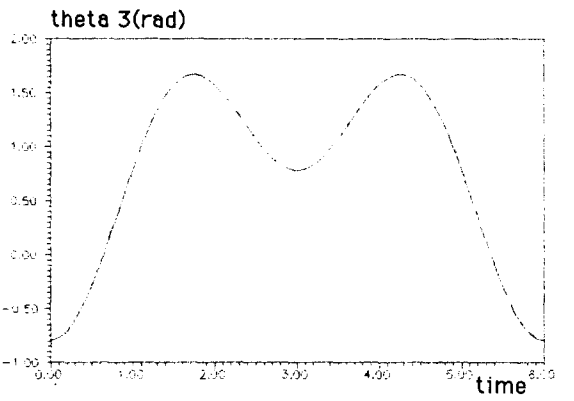
(b)



(c)

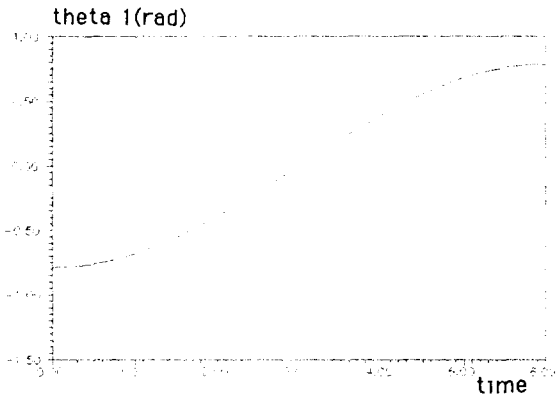


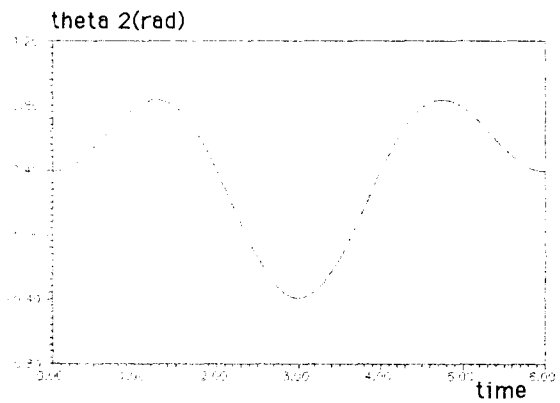


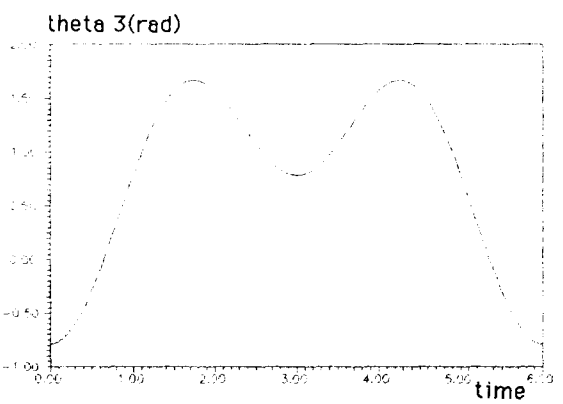


(a)

Figure 9. Plot of the actual and ideal trajectories with random neural outputs for a) joint 1 b) joint 2 c) joint 3.

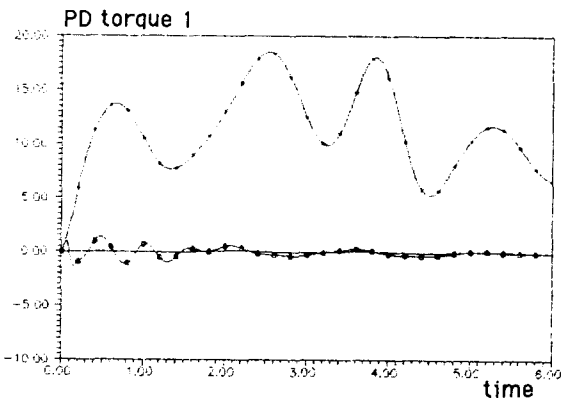




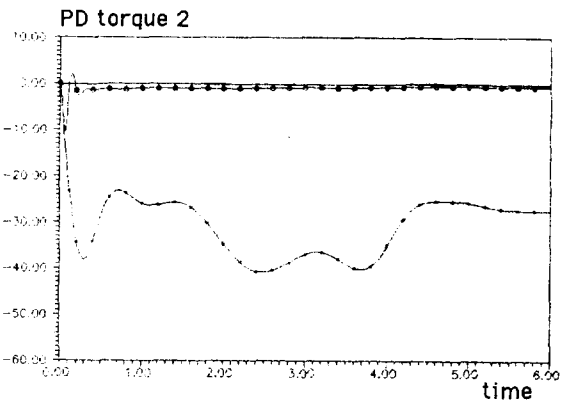


(b)

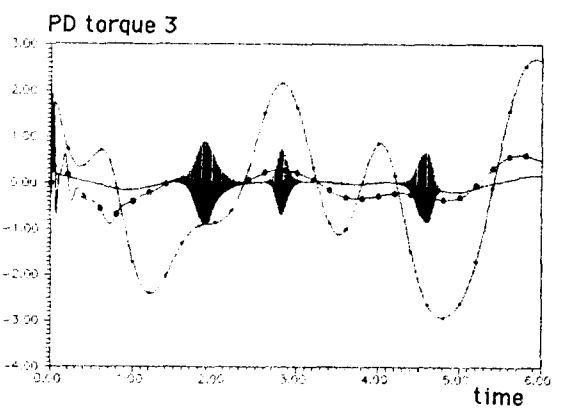
Figure 4. Plot of the joint angle trajectories for a) during the first learning attempt b) during the 31 attempts.



(a)

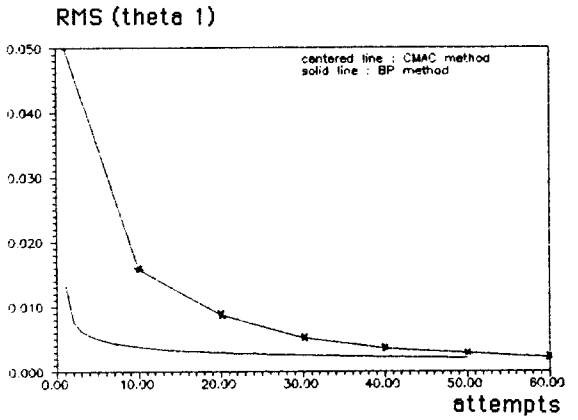


(b)

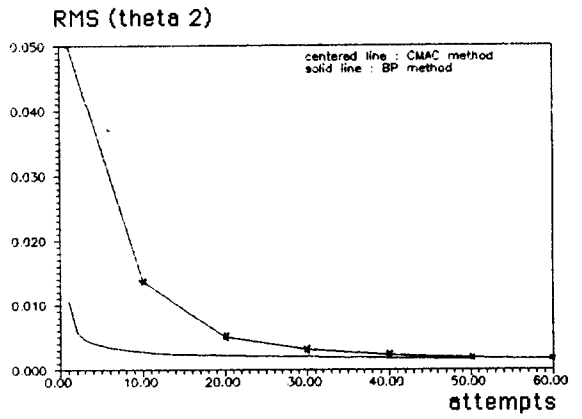


(c)

Figure 5. Plot of the PD controller outputs for a) joint 1 b) joint 2 c) joint 3.
(—+— : attempt with random neural outputs
—●— : 1 attempt
—— : 31 attempt)



(a)



(b)

Figure 6. Comparison of the learning performance betwen the BP method and the CMAC mathod for a) joint 1 b) joint 2.

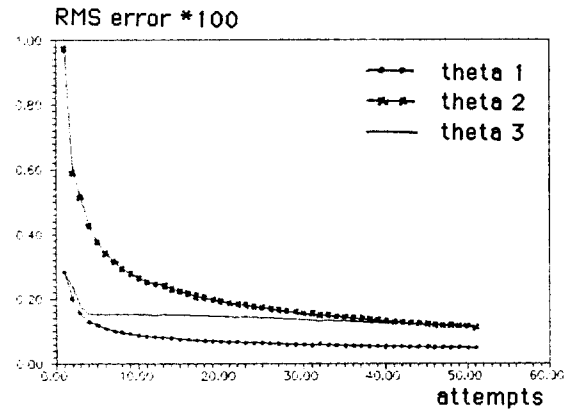


Figure 7. Learning curve with 1 Kg paylaod.