Self-Adaptive Neural Architectures for Control Applications

Sheng-De Wang and Hackerd M. S. Yeh

Department of Electrical Engineering
National Taiwan University, Taipei 10764, Taiwan

Abstract — This paper is aimed to explore the potential use of the modeling capacity of neural networks for control applications. A neuromorphic controller, called Self-Adaptive Neural Controller (SANC), is thus designed by utilizing the neural modeling capacity. The results of this approach reveal at least two expected benefits: "learning from example" and dynamical adaptation. With the learning-from-example ability, SANC is essentially application independent, even if the plant considered is too complex or too uncertain to be modeled by precise mathematical expressions. With the dynamical-adaptation feature, SANC is shown to be robust, adaptive and capable of learning, even if the environment varies too large to be controlled by use of traditional controllers.

1. Introduction

The theories and techniques of automatic control have played a vital role in modern society. Advances in the theory and practice of automatic control provide the better control quality and the opportunity to meet the needs of more applications. On the other hand, as social and technological systems are becoming increasingly complex and highly coupled, the demand of more powerful control theory is rapidly growing. Although adaptive control techniques are developed for systems to perform over larger ranges of uncertainties, such as Model Reference Adaptive Control (MRAC) and Self Tuning Regulator (STR) [1], there are still some fundamental limitations in the current adaptive techniques. For example, the computational complexity grows geometrically with the number of unknown parameters.

The trend of control theory, therefore, is intended to develop more general approaches, which can solve as many applications as possible, even if the plants are too complex to be modeled with concise mathematical expressions or the environmental conditions vary too large to be controlled by use of traditional controllers. The control problem of this kind is in no ways a trivial one. Fortunately, the emergence of neural networks reveals a new possibility. This paper is then aimed to explore the potential use of neural networks for control.

To apply neural computing to control systems, we can consider neural networks as special cases of either intelligent systems or adaptive systems. Like traditional artificial intelligence, the objective of neural computing is to create a computing model that mimics the functionality of the brain in a very simplified manner. Hence, neural computing can be considered as another candidate of intelligent control systems; Bavarian [2] demonstrated several simple examples. Especially, in situations where input is noisy or incomplete, a neural network can still produce reasonable results. This makes neural computing attractive for intelligent control.

On the other hand, the way neural networks store information is both distributed and associative. In addition, some networks, e.g. feedforward neural networks, perform some form of interpolation. Therefore, we may consider neural networks as identification tools to model unknown or uncertain systems. Moreover, neural networks can adapt, on-line, their internal knowledge representation (i.e. the strength of interconnections) to keep track of the variations of systems. Thus, we may imagine neural networks as adaptive systems, and term this capability as neural modeling, which has played a major role of this paper.

Guez et. al. [4] demonstrates the power of neural modeling by using an trainable neuromorphic controller (TAC) to learn (i.e. to model) an available controller (e.g. a human expert). The so-called TAC may work better than his teacher in some properties, such as robustness. However, Guez does not utilize the on-line adaptation capacity of neural networks: TAC can not adapt itself to compensate variations of the environment. The other major drawback of TAC is that a teacher is needed, and the teacher must be skilled. Otherwise, some heuristic techniques must be introduced to filter data. Similar but early work was done in 1964 by Widrow et. al. [5] [6] using the Adaline (ADAptive LInear NEuron) and Madaline (Multiple Adalines). Like Guez's TAC, a skilled human teacher, who makes "Bang-bang" control decisions, is used to train the neural networks.

Neural models used above are basically multilayer networks with backpropagation (error). However, there are other possibilities. Guez et. al. [3] proposed a neural estimator by using Hopfield network as an associative and distributed storage. Barto et. al. [7] developed two neuron-like elements, so-called Associative Search Element (ASE) and Adaptive Critic Element (ACE), to perform unsupervised learning, and then to solve the cart-pole problem although it was eventually unstable.

2. Neural Modeling

2.1 Feedforward Neural Networks

Since 1950s, many neural network models have been developed [8] [9]. Among these we are, from the viewpoint of identification, interested in so-called feedforward neural networks [10]. The major reasons are 1) feedforward neural networks are popular, simple, and well known; and 2) their modeling capacity are excellent and the mathematical

analysis has been well established [11] [12] [13] [14] [15]. Although they suffer from some drawbacks, they are good enough for our applications.

A neural network consists of many processing elements, analogous to the biological neuron, interconnected together with a lot of connection weights, analogous to the strength of synapses. A processing element has many input paths, and generates a value on the output path according to the values on these input paths. The generated value, then, transfers to input paths of other elements via connection weights. Elements may be connected in arbitrary way (even recursively). However, they are usually organized into a sequence of layers with full or sparse connections between successive layers. This is the so-called multilayered neural networks.

2.2 Mathematical Analysis of Multilayer Networks

For a given network, the modeling capacity means the existence of a set of connection weights such that the response of the network can approximate the desired response within the tolerant error. Fig. 2.1 illustrates a typical modeling problem, where E is an error function depending on the output of the system, $\mathbf{f}(\mathbf{x})$, and the output of the network, $\mathbf{F}(\mathbf{x};\mathbf{W})$. Assume that $\mathbf{x}$, $\mathbf{f}(\mathbf{x})$ and $\mathbf{F}(\mathbf{x};\mathbf{W})$ are all column vectors and $\mathbf{W}$ is a vector of parameters consisting of all connection weights. Also, suppose that two functions $\mathbf{f}(\mathbf{x})$ and $\mathbf{F}(\mathbf{x};\mathbf{W})$ of the point $\mathbf{x}$ are defined within a certain point set Ω in a space of any number of dimensions. Usually, we define E, a scalar function, as

$$E(\mathbf{F}(\mathbf{x};\mathbf{W}), \mathbf{f}(\mathbf{x})) = \begin{cases} \int_{\mathbf{x}\in\Omega} \mathbf{e}(\mathbf{x})^T \mathbf{I}\, \mathbf{e}(\mathbf{x})\, d\mathbf{x} & \textit{the continuous case} \\ \sum_{\mathbf{x}\in\Omega} \mathbf{e}(\mathbf{x})^T \mathbf{I}\, \mathbf{e}(\mathbf{x}) & \textit{the discrete case} \end{cases} \tag{2.1}$$

where $\mathbf{e}(\mathbf{x}) = \mathbf{f}(\mathbf{x})-\mathbf{F}(\mathbf{x};\mathbf{W})$ and $\mathbf{I}$ is an identity matrix

.Moore and Poggio [11] has shown that networks with two layers of hidden units can uniformly(to within any $\varepsilon>0$) any real continuous function mapping R^M to R. It is easy to extend the results of Moore and Poggio [11] to the case of f is a function mappingR^M to R^N. Networks with two layers of hidden units are sufficient such that $\mathbf{F}(\mathbf{x};\mathbf{W})$ can uniformly approximate (to within any $\varepsilon > 0$) $\mathbf{f}(\mathbf{x})$, which is assumed to be continuous, over a point set Ω. In these cases, the activation functions of networks are assumed to be the sigmoid function.

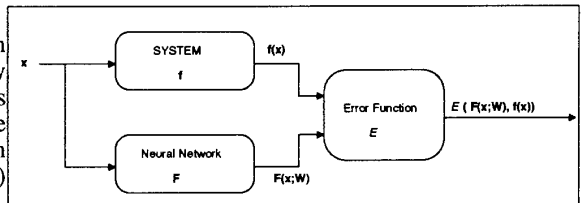

Fig. 2-1 Modeling a system by using a neural network.

2.3 Learning Algorithms

The learning algorithm used in this paper is the backpropagation algorithm [10]. The backpropagation algorithm is a popular, simple, but slow learning algorithm. It is, in fact, nothing but the gradient method plus the chain rule. The way to modify weights can be described by (2.2).

$$\Delta w_{s,j,i}^{(l+1)} = -\alpha \left. \frac{\partial E(\mathbf{F}(\mathbf{x};\mathbf{W}), \mathbf{f}(\mathbf{x}))}{\partial w_{s,j,i}} \right|_{w_{s,j,i}^{(l)}} + \beta \Delta w_{s,j,i}^{(l)} \tag{2.2}$$

where $w_{s,j,i}^{(l)}$ is weights joining i^{th} neuron in layer (s-1) to j^{th} neuron in layer s after training l times. By using the chain rule, it is easy to derive the backpropagation algorithm given in the literature. The second term in (2.2), so-called momentum, acts as a low-pass filter on the delta weights, and a lower learning coefficient but faster learning is allowed. In addition, according to experimental experience, it is better to decrease α and β as learning in progress.

In summary, we can comment neural modeling by the following merits: 1) neural modeling is based on "learning-from-example" without any demand of priori knowledge about the system considered; 2) the modeling capacity is excellent even if activation functions are fixed; and 3) it is less sensitive to noise (so training with noisy data is possible).

3. The Self-Adaptive Neural Controller

3.1 The SANC Architecture

A SANC consists of two neural networks, as depicted in Fig. 3.1. One, called NNp, is used to model the plant, and to keep track of the variations of environmental conditions, etc. Once NNp has modeled the plant, a gradient method is used to generate control signals by assuming that NNp is the same as the plant. The other network, called NNc, is then used to memorize the generated control signals.

The operation of SANC can be divided into three different phases: Initial Training (IT), Controlling and Adapting

(CA), and MEmorizing (ME).

The Initial-Training Phase

The first step to use SANC is the IT phase in which NNp is trained to overall approximate the dynamics of the plant. This phase is usually off-line by training with the previously observed data or with the data generated according to the mathematical model of the plant. However, it is possible to train NNp on-line as shown in Fig. 3.2. In Fig. 3.2 NNp stands by and follows the response of the plant, which is controlled by another controller (e.g. a skilled human) or a random generator. Notice that we must feed the current state of the plant to NNp as part of input since the plant is usually dynamic and multi-layer networks are nothing but a mapping of the form: $R^M \rightarrow R^N$ as described in the previous section. In some situations, where not all states are measurable, modification is needed by adding previous measured outputs together with the current one as part of the input of NNp. Both on-line and off-line learning are straightforward and easy to implement. In both cases, the plant can be imagined as a training-pattern generator.

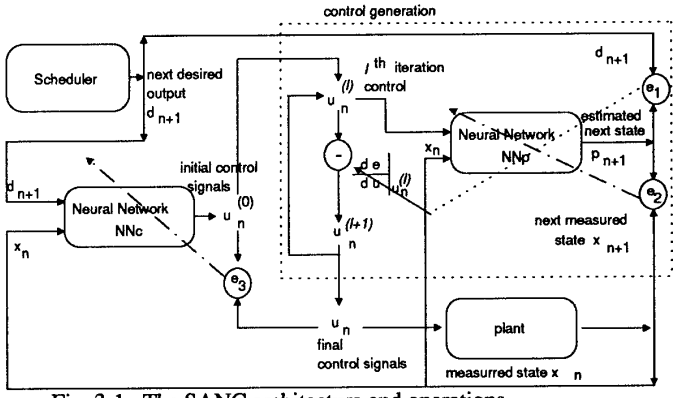

Fig. 3-1. The SANC architecture and operations.

Algorithm 3.1

Input: state $\mathbf{x}_n$ and desired output $\mathbf{d}_{n+1}$ at time instant n and n+1.

Output: the control signal $\mathbf{u}_n$

Parameters: $\mathbf{u}_{max}$, $\mathbf{u}_{min}$, MAIN (Maximal Allowed Iteration Number) and MATE (Maximal Allowed Tolerant Error)

1. Obtain $\mathbf{u}_n^{(0)}$ by applying $\mathbf{x}_n$ and $\mathbf{d}_{n+1}$ as the input of NNc, and then extracting its output.
2. Let $l \leftarrow 0$
3. Obtain $\mathbf{p}_{n+1}^{(l)}$ by applying $\mathbf{u}_n^{(l)}$ and $\mathbf{x}_n$ as the input of NNp, and then extracting its output.
4. Obtain J_{n+1} by $J_{n+1}=(e_1)_{n+1}{}^T Q(e_1)_{n+1}+u_n{}^T R u_n$, where $(e_1)_{n+1}=d_{n+1}-p_{n+1}$, p_{n+1} is the output of the NNp at time instant n+1, and Q, R are weighting matrices.
5. Use a backpropagation algorithm and see the control signal u_n to NNp as an additional connection weight of NNp to obtain $\varepsilon = \partial J_{n+1}/\partial u_n \big|_{u_n^{(l)}}$.
6. Let $\mathbf{u}_n^{(l+1)} \leftarrow \mathbf{u}_n^{(l)} + \eta\,\varepsilon$.
7. $l \leftarrow l+1$.
8. If $\mathbf{u}_n^{(l)} \geq \mathbf{u}_{max}$, then $\mathbf{u}_n = \mathbf{u}_{max}$ and exit.
9. If $\mathbf{u}_n^{(l)} \leq \mathbf{u}_{min}$, then $\mathbf{u}_n = \mathbf{u}_{min}$ and exit.
10. If $l >$ MAIN or $\varepsilon <$ MATE, then let $\mathbf{u}_n = \mathbf{u}_n^{(l)}$ and exit.
11. Goto *Step 3*.

The Controlling-and-Adapting Phase

Once NNp is trained, we can use it with NNc to generate control signals according to a gradient and iterative algorithm. It first generates an initial control signal by making use of NNc. Then, the generated signal is fed into NNp to produce an output vector, which is considered as the next state of the plant estimated by NNp. By comparing the output of NNp with the desired response, an error signal is then formed and backpropagated through NNp. Thus, a new control signal is generated by subtracting the backpropagated error from current control signal. Obviously, this is a kind of gradient method. Algorithm 3.1 sums up the control generation algorithm. This control signal is then fed into NNp again, and the previous process repeats until time-out or until the error is tolerable. In order to adapt NNp to keep track of the variations, and in order to train NNc to memorize the generated control signals, on-line learning for NNp and NNc driven by the error e2 and e3, respectively, is also activated while controlling, as depicted in Fig. 3.1. This step is the control operation of SANC, and we call it the Controlling and Adapting (CA) phase.

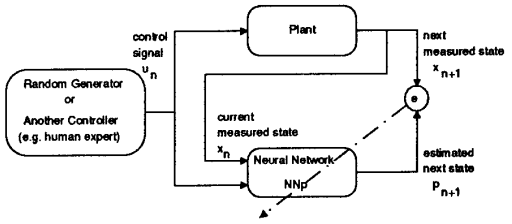

Fig. 3-2. Function blocks of on-line training

The Memorizing Phase

Since the backpropagation algorithm suffers from the temporally unstable shortcoming, it is necessary to train a neural network iteratively. It is easy and efficient to iteratively improve the neural network by using off-line learning although on-line learning is theoretically possible. The third step is then to train NNc off-line as illustrated in Fig. 3.3. To do this, a table recording the previous control trace must be established in CA phase. An entry, say n^{th} entry, of the

table should contain three items about n^{th} sample: the system state (or measured output) $\mathbf{x}_n$, the desired next state $\mathbf{d}_{n+1}$, and the corresponding control signal $\mathbf{u}_n$, generated in CA phase. Then, we may train NNc by selecting randomly an entry of the table, say n^{th}, putting $\mathbf{x}_n$ and $\mathbf{d}_{n+1}$ into the input layer of NN_c, and modifying connection weights by using $\mathbf{u}_n$ as the desired output. This step is then denoted as the MEmorizing (ME) phase.

In summary, IT and ME phases can be considered as overall learning, which models the plant and the algorithm on the whole space. In the other end of spectrum, the CA phase can be considered as a specialized learning scheme, which approximates them only along the control trajectory.

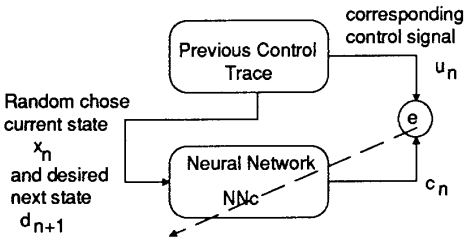

Fig. 3-3 Function blocks of off-line training NNc.

4. SANC for Robust Adaptive Control

To demonstrate the power of SANC experimentally, in this section we will apply SANC to the tracking problem of robot manipulators whose dynamics is nonlinear and imprecise. SANC is then shown to be robust, adaptive, and capable of learning. The "learn-from-example" property of neural modeling is also illustrated by training NNp not only based on a mathematical model but also by use of noisy data.

4.1 Robot Manipulators

The typical problem of robot manipulators is to determine the nature of the joint torques so that the end-effector follows the desired trajectory as fast and accurate as possible. The need for adaptive control is due to both parametric and structural uncertainties. Parametric uncertainties arise from imprecise knowledge of the manipulator mass properties, unknown loads, uncertainty in the load position, and so on. Structural uncertainties are due to the presence of high-frequency unmodeled dynamics, resonant modes, etc.

In this chapter, we are interested in a single degree of freedom manipulator. The dynamic equations are [1]

$$(m\, l_c^2 + I)\, q + \dot{m}\, l_c\, g \cos q = u \tag{4.1}$$

where

q:joint displacement, m: mass of link,
I:centroidal inertia of link, l: length of link,
l_c:centroidal length of link, u: joint torque, and
g :gravity.

In addition, we define the state variables of the system as the angular position and angular velocity of the link respectively, i.e.,

$$\mathbf{x} = < q, \dot{q} >^T. \tag{4.2}$$

and assume that they are available from suitable sensors. In the derivation of (4.1), effects due to friction and input-saturation were neglected. For numerical data, link 2 of Unimation PUMA 560 robot manipulator is used, and its parameters are $l = 0.432$ and $m = 15.91$ kg. We also assume that $l_c = l/2$.

A lot of methods have been developed to solve the problem of robot manipulator control. Some of them are adaptive systems [17], and some of them are expert controllers [16]. In following sections we would like to employ SANC by different implementation strategies, and to study its robustness, capability of adaptation, etc.

4.2 Training SANC with the Mathematical Model

In this section, we discuss how to train NNp during the IT phase by using the data generated by the mathematical model described by (4.1). First, we randomly generate the current state variables and corresponding control signals. NNp uses them as the input vector to produce its own output vector, and then calculates the difference between this output vector and the next state obtained by use of (4.1). Finally, the weights of NNp are modified to reduce the difference by using the backpropagation algorithm. In other words, NNp is trained to model (4.1).

The learing rate and other parameters used in this experiment are listed as follows.

$\alpha_n = 0.5 \times 0.9^{0.00008 \times n}$ $\beta_n = 0.5 \times 0.93^{0.00008 \times n}$

$\eta = 50{,}000$ $u_{max} = 500$ (Kg rad/s^2)

$u_{min} = -500$ (Kg rad/s^2) Sampling Rate = 20 Hz

$\mathbf{Q} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$ $\mathbf{R} = 0.$

In addition, the desired trajectory is expressed as

$$q_d(t) = \frac{\pi}{6}[\,1 + \sin(\pi\, t)]. \tag{4.3}$$

4.3 Training SANC with Noisy Data

In some situations the plant under consideration may be too complex or too uncertain to be modeled by concise mathematical models. As a result, there are only measured data, which are generated by the actual operation of the plant, available. Needless to say, measured data are usually distorted by noise.

For the purpose of simulation, we use the same method as that described in the previous section to generate training patterns. However, we distorts these patterns with noise, which is assumed to be the uniform distribution with mean equal to 0. Other parameters, such as the learning coefficient, the desired trajectory, etc., are the same as those listed in the previous section.

4.4 Simulation Results

In both noisy and noiseless cases, NNp has been trained 100,000 times in IT phase. Then, we employ SANC to control the robot manipulator to keep track of the desired trajectory as described in (4.3).

Fig. 4.1, where noiseless training patterns are used, shows the effects of the Maximal Allowed Iteration Number (MAIN) in the first CA phase. For the purpose of comparison, we define the performance measure as (4.4).

$$J = \sum_{1 \le j \le 40} (q(t_j) - q_d(t_j))^2 \tag{4.4}$$

where $t_j = \frac{j}{20}$.

The performance measures based on (4.4) are listed in Table 4.1, which reveals several interesting properties. First, SANC operates much better at the second run than the first run. SANC is then said to be capable of learning. However, it does not seem to further improve the performance at the third run since the performance of SANC is bounded to the convergence speed and the modeling accuracy of the backpropagation algorithm. Second, SANC is shown to be able to be trained with noisy data although the performance is degraded by noise. Moreover, SANC is shown to be quite robust with respect to parameter changes and noise disturbances in Figs. 4.2 to 4.4. Each figure corresponds to the cases of the mass of the link being varied and the measured output being distorted by noises with some degree of Signal-to-Noise-Ratio(SNR).

5. Conclusions

In this paper we have studied the use of the modeling capacity of neural networks for control applications. A neuromorphic controller, SANC, has been developed by utilizing the modeling capacity of networks. The modeling capacity is further demonstrated by employing networks as memory components of SANC. These components are NNp and NNc. SANC is shown not only to exhibit the expected benefits of neural modeling, but also to be more robust, adaptive and capable of learning than conventional control.

The expected benefits of invoking neural modeling are 1) the modeling capacity, 2) the property of "learning from example", 3) the capability of dynamic adaptation, and 4) parallel, distributed, fault-tolerant and associated memory. In sum, we conclude that SANC can be a basis for robust and adaptive control although the gradient method used in SANC is simple and poor.

References

[1] Narendra, K.S., and Annaswamy, A.M., *Stable Adaptive Systems*. New Jersey: Prentice-Hall, Inc., 1989.

[2] Bavarian, B., "Introduction to Neural Networks for Intelligent Control," *IEEE Control Systems Magazine*, pp. 3-7, Apr. 1988.

[3] Guez, A., Eilbert, J.L., and Kam, M., "Neural Network Architecture for Control," *IEEE Control Systems Magazine*, pp. 22-25, Apr. 1988.

[4] Guez, A., and Selinsky, J., "A Trainable Neuromorphic Controller," *Journal of Robotic Systems* 5(4), pp. 363-388, 1988.

[5] Widrow, B., "The Original Adaptive Broom Balance," *IEEE Conference on Circuits and Systems*, pp. 351-357, Philadelphia, PA, 1987.

[6] Tolat, V.V., and Widrow, B., "An Adaptive "Broom Balancer" with Visual Inputs," *IEEE International Conference on Neural Networks*, pp. II 641-647, San Diego, 1988.

[7] Barto, A.G., Sutton, R.S., and Anderson, C.W., "Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problem," *IEEE Transaction on Systems, Man, and Cybernetics*, vol. SMC-13, No. 5, pp. 834-846, Sep./Oct. 1983.

[8] Hecht-Nielsen, R., "Neurocomputing: picking the Human Brain," *IEEE Spectrum* 25(3), pp. 36-41, Mar. 1988.

[9] Lippman, R.P., "An Introduction to Computing with Neural Nets," *IEEE ASSP Magazine*, pp.4-22, Apr. 1987.

[10] Rulmelhart, D.E., and McClelland, J.L., *Parallel Distributed Processing*. Cambridge: MIT Press, 1987.

[11] Moore, B, and Poggio, T., "Representation Properties of Multilayer Feedforward Networks," *INNS First Annual Meeting*, Boston, Sep. 1988.
[12] Werbos, P., *Beyond Regression: New Tools for Prediction and Analysis in the behavioral Sciences*. Ph.D. thesis, Hardvard U. Committee on Applied Mathematics, Nov. 1974.
[13] Werbos, P., "Building and Understanding Adaptive Systems: A Statistical/Numerical Approach to Factory Automation and Brain Research," *IEEE Transaction on Systems, Man and Cybernetics*, vol. SMC-17, No. 1, pp. 7-20, Jan.-Feb. 1987.
[14] Werbos, P., "Backpropagation: Past and Future," *IEEE International Conference on Neural Networks*, pp. I 343-353, San Diego, 1988.
[15] Parker, D.B., "Optimal Algorithms for Adaptive Networks: Second Order Back Propagation, Second Order Direct Propagation, and Second Order Hebbian Learning," *IEEE First International Conference on Neural Networks*, pp. II 593-600, San Diego, 1987.
[16] Geng, Z., and Jamshidi, M., "Expert Self-Learing Controller for Robot Manipulator," *Proceedings of The 27th Conference on Decision and Control*, Austin, Texas,pp. 1090-1095, Dec. 1988.
[17] Tomizuka, M., Horowitz, R., Answar. G., and Jia, Y. L., "Implementation of Adaptive Techniques for Motion Control of Robotic Manipulators," *ASME Journal of Dynamic System, Measurement, and Control*, vol. 110, pp. 62-69, Mar. 1988.

J	Noiseless	SNR=10*dB*	SNR=3*dB*	J	Noiseless	SNR=10*dB*	SNR=3*dB*
MAIN = 12				MAIN = 30			
first run	0.398203	1.26005	O.O.C	first run	0.0111047	0.0123734	0.0519127
second run	0.3271	0.609952	O.O.C	second run	0.00843271	0.00981863	0.0351476
third run	0.327802	0.598343	O.O.C	third run	0.0084329	0.00981977	0.0366522
MAIN = 16				MAIN = 50			
first run	0.0259774	0.0375232	0.693929	first run	0.0108041	0.011969	0.0482515
second run	0.0204605	0.0280958	0.253084	second run	0.00818406	0.00951761	0.0340749
third run	0.0204636	0.028102	0.243459	third run	0.00818395	0.00951701	0.0356036
MAIN = 18				MAIN = 100			
first run	0.0162816	0.0208473	0.234811	first run	0.0107622	0.0119096	0.0480224
second run	0.0125597	0.0158155	0.0983174	second run	0.00813574	0.00946726	0.0338979
third run	0.0125601	0.0158171	0.0981951	third run	0.00813543	0.00946818	0.0354225

Table 4.1 Performance measures about learning capability and noise sensitivity. The initials O. O. C. denotes "out of control".

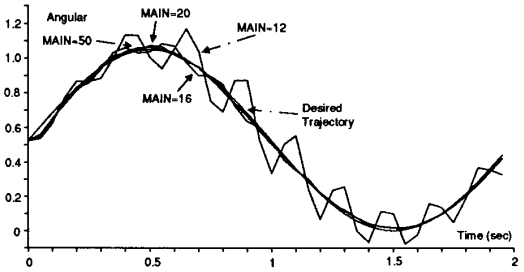

Fig. 4-1 Effects of MAIN; without noise.

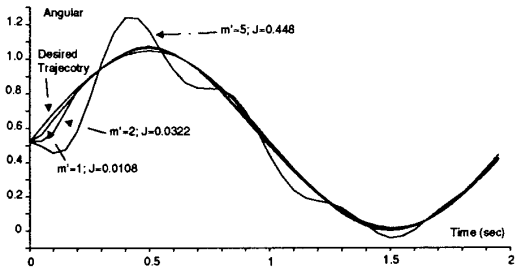

Fig. 4-2 Sensitivity to mass of the link; without noise.

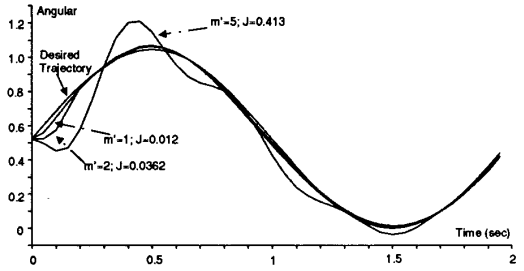

Fig. 4-3 Sensitivity to mass of the link; SNR= 3dB.

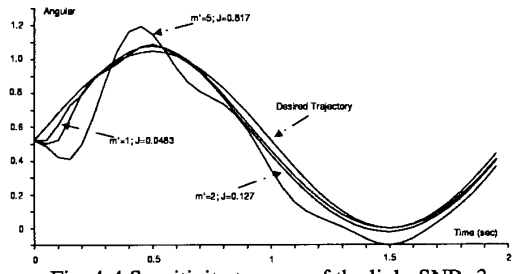

Fig. 4-4 Sensitivity to mass of the link; SNR=3 dB.