ISSN: 0128-7680
Pertanika J. Sci. & Technol. Supplement **12(2)**: 139 - 158 (2004) © Universiti Putra Malaysia Press

A Self-learning Nonlinear Variable Gain Proportional Derivative (PD) Controller in Robot Manipulators

[1]Loo Chu Kiong, [2]Mandava Rajeswari, [1]Wong Eng Kiong & [1]M. V. C. Rao**
[1]Faculty of Engineering and Technology, Multimedia University (Melaka), 75450 Melaka, Malaysia
E-mail: ckloo@mmu.edu.my
[2] School of Industrial Technology, Universiti Sains Malaysia, Penang, Malaysia
E-mail: mandava@cs.usm.my

ABSTRACT

This paper proposes a nonlinear variable gain Proportional-Derivative (PD) controller that exhibits self-constructing and self-learning capabilities. In this method, the conventional linear PD controller is augmented with a nonlinear variable PD gain control signal using a dynamic structural network. The dynamic structural network known as Growing Multi-Experts Network grows in time by placing hidden nodes in regions of the state space visited by the system during operation. This results in a network that is "economic" in terms of network size. The proposed approach enhances the adaptability of conventional PD controller while preserving its' linear structure. Based on the simulation study on variable load and friction compensation, the fast adaptation is shown to be able to compensate the non-linearity and the uncertainty in the robotic system.

Keywords: Nonlinear PD Controller, feedback error learning, dynamic structure neural network, friction compensation, variable load compensation

INTRODUCTION

Since their inception over thirty years ago, industrial robots have been widely used in industry. Despite a large number of sophisticated robot control methods reported in the robotics and control literature, linear PD controller is still one of the commonly used controllers in commercial industrial robots. Robot manipulators have highly nonlinear dynamics. Since the robot dynamics is nonlinear while PD control is a linear-control, applying PD control to trajectory-following problems would require a gain scheduling approach using local models; that is the robot dynamics is linearized about some operating points so that the PD gains can be selected to achieve certain performance specifications. The trajectory, however, varies with time (as opposed to a set-point). Therefore, the gains so selected may not be appropriate due to the linearization. Such a predesign needs significant effort and a priori robot model information, and cannot adapt itself to model error or time variation in an on-line manner. The limitations mentioned above have inspired the idea of model free self-adaptive independent joint controller proposed in this paper. The basic challenge of designing a model free self-adaptive controller is how to provide an on-line learning algorithm that does not require preliminary off-line learning. Instead, the controller is able to automatically improve its performance on the fly even when the robot model is unknown or partially known. This shares the idea of universal model-free controllers such as neural networks. Therefore, a neural network based feedforward controller that exhibits self-constructing and self-learning capabilities is presented in this paper.

[**] Corresponding author

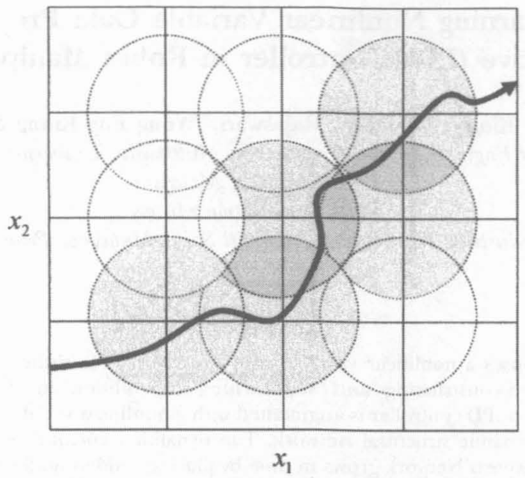


Fig. 1: Two dimension input space divided by grids. Basis functions are located on the mesh points. The bold arrow indicates the input state trajectory. It is noted that basis functions positioned along the trajectory are active, the rest are innate and redundant

Problem

Until the beginning of the nineties, the principal type of neural networks used for system identification and control problems was the multilayer perceptron (MLP) with sigmoidal hidden units in Psaltis *et al.*(1988); Khalid *et al.*(1992). However, slow learning and local minima problem of MLP have led to the development of modular neural networks such as the radial basis function (RBF) neural networks Kadirkamanathan *et al.*(1993). The RBF network has gained increasing attention among researchers owing to its modular structure and fast learning capability. With the use of Gaussian functions, the RBF network forms a local representation. Each basis function responds only to inputs in the neighborhood of a reference vector known as basis function center. The spread of the neighborhood is determined by the variance of basis function. In using RBF networks as neural network based controller, one typically initializes the center of the basis functions on regular points of a square mesh covering a relevant region of space where the input state is known to be contained, and denoted by a compact set $\chi_n \subset \Re^n$. This approach is widely used in robotic control application Ge (1996); Ge *et al.* (2001); Ge *et al.* (2002a); Ge *et al.* (2002b). This situation is also encountered in system identification problems, where the state components represent delayed versions of the plant output response. Therefore, state variables move along specific trajectories within the state space, χ_n, depending on the excitation conditions of the model. This implies that placing basis functions on all mesh points within χ_n results in a lot of redundancy. *Fig. 1* illustrates this problem.

Nevertheless, even assuming that the basis functions are placed along the trajectory, they may be positioned close to each other owing to network adaptation. It is illustrated in *Fig. 2*. *Fig. 3* indicates that the basis function A appears to be redundant since basis function A may produce a response amplitude similar to basis function B for an input signal *x*. Under this situation, either basis function A or B alone is able to perform the approximation sufficiently well for the covered region.

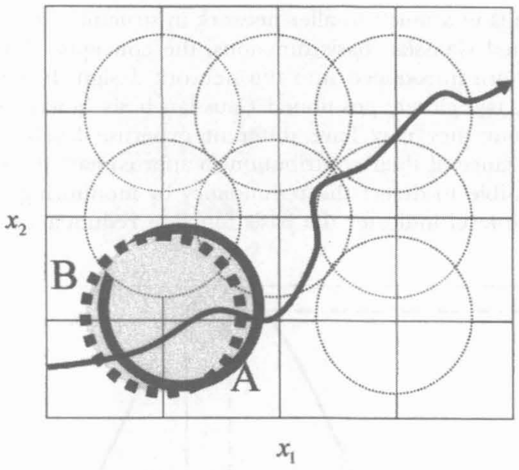


Fig. 2: Two basis functions are placed close to each other. One of them appears to be redundant

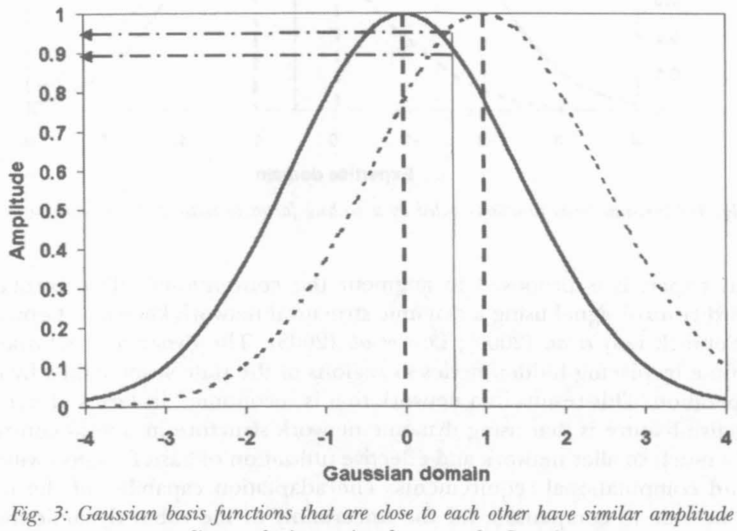


Fig. 3: Gaussian basis functions that are close to each other have similar amplitude corresponding to the input signal near to the Gaussian center

Overall, the applicability of modular neural network with static structure and using basis function is limited by the following issues:

- For a system identification or control problem, the initialization of Gaussian basis function based on mesh grid method may result in redundancy.
- Gaussian basis functions that are located close to each other may result in redundancy since they may generate similar activation values corresponding to an input signal close to their center.

In this paper, it is proposed to adaptively place the basis functions in the strategic regions in order to cope with the problem of a large number of redundant basis functions. This results in a much smaller network in structure. In order to differentiate two closely positioned Gaussian basis functions, the concepts of "expertise level" and "expertise domain" are introduced into the network design. It is illustrated in *Fig. 4.* Based on this idea, two closely positioned Gaussian basis functions may share similar expertise domain but they may have different expertise levels. The expertise level indicates the significance of their contribution to approximate the region they recover. It is, therefore, possible to detect the redundancy by monitoring their expertise level value. Low expertise level indicates the basis function redundancy.

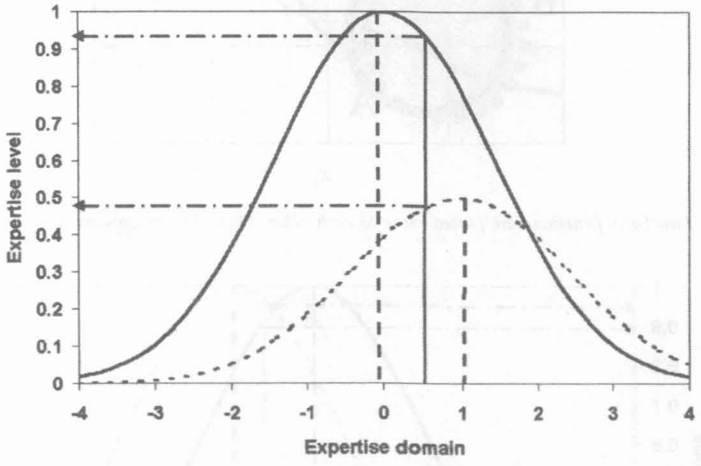


Fig. 4: Gaussian basis functions gated by a scaling factor to indicate their expertise level

In this paper, it is proposed to augment the conventional PD controller with a feedforward control signal using a dynamic structural network known as Growing Multi-Experts Network Loo *et al.* (2000); Loo *et al.* (2003). The dynamic structural network grows in time by placing hidden nodes in regions of the state space visited by the robot during operation. This results in a network that is "economic" in terms of network size. The attractive feature is that using dynamic network structure in neural control design results in a much smaller network and effective utilization of basis function with reduced storage and computational requirements. The adaptation capability of the network is shown to be able to compensate for the uncertainty of the robot dynamic model.

Growing Multi-Experts Network (GMN)

Control and modeling problem involves approximation of an unknown nonlinear function. In nonlinear systems the function complexity varies throughout the input space. Instead of trying to approximate the whole function globally, the input space is partitioned into subspaces that are easier to handle. This may be achieved by effectively using 'divide and conquer' strategy where the problem space is decomposed into overlapping regions and local experts approximate the function in every region. In this context, the problem space decomposition requires a gating unit that may clearly define the zone of influence of local experts.

The local experts are associated with a weighting factor that indicates the importance of each local expert. Finally, the network overall output is computed by combining the outputs from all the local experts.

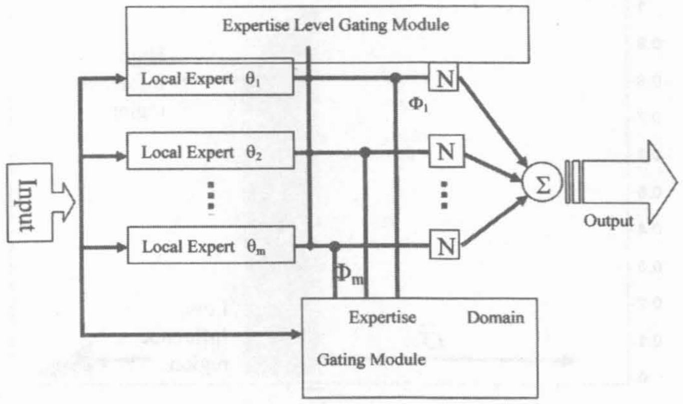


Fig. 5: Growing Multi-Experts Network architecture. N is the normalization symbol

Overall, GMN consists of three main building modules namely, expertise domain, expertise level, and local experts. Local experts may have a nonlinear function, linear function or even a mixture of both. In this paper, a polynomial function, mainly a linear model, is considered. Expertise domain is used to define the influence regions where the local experts perform local approximation. The competency of local experts is indicated by expertise level. High value of expertise level means the local expert is making significant contribution to the overall network output. *Fig. 5* shows the Growing Multi-Experts Network architecture. The design of these three modules and the learning algorithm of GMN are described in detail in the following sections.

Expertise Domain

Gaussian basis functions have localized receptive fields. With this property, the Gaussian basis function is obviously a suitable choice for an expertise domain gating function to indicate the validity of input data with respect to the local expert model. The distance metric $d(\bar{x};\bar{c}_k,\bar{\sigma}_k)$ for Gaussian basis function k as defined in equation (1) can scale the spread of the Gaussian basis function relative to its center $\bar{\sigma}_k$ and the Gaussian basis function takes the distance metric as input and $\bar{\sigma}_k$ as diagonal matrix.

$$d(\bar{x};\bar{c}_k,\bar{\sigma}_k) = \left|\bar{x}-\bar{c}_k\right|^T \bar{\sigma}_k^{-1} \left|\bar{x}-\bar{c}_k\right| \qquad (1)$$

$$\mu_k\left(d(\bar{x};\bar{c}_k,\bar{\sigma}_k)\right) = \exp\left(-d(\bar{x};\bar{c}_k,\bar{\sigma}_k)\right) \qquad (2)$$

Expertise Level

When two expertise domains (Gaussian basis functions) are located close to each other, there will be a high degree of overlapping. As a result, one of the local experts associated with the expertise domains may appear to be redundant. In such a case, a single local expert is sufficient to approximate the desired function within the expertise domain. In order to avoid such a redundancy, expertise level gating function is introduced to denote the importance of the local expert model so that the unimportant local expert may be deleted. Each expertise domain is multiplied by a weighting factor α_i known as

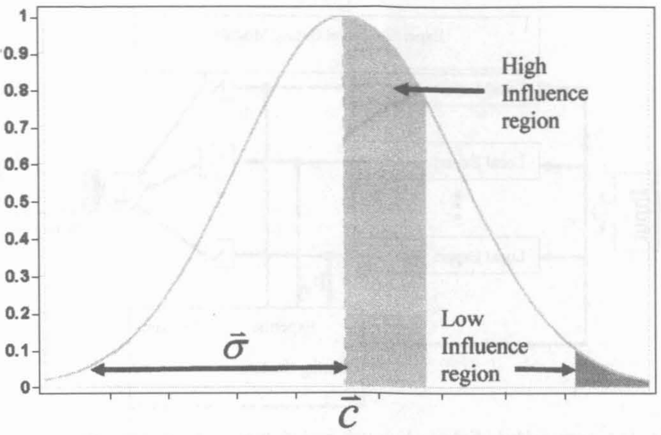


Fig. 6: Gaussian basis function used as expertise domain gating module

expertise level function. It is noted that α_i must also be a non-decreasing function of an adjustable parameter β_i. Smoothness of expertise function is also required because it is necessary for the gradient-based learning method. In addition, function α_i must be nonlinear because otherwise, it may be absorbed by local model function. To satisfy these requirements, the weighting factor α_i is defined as a logistic sigmoid function so that α_i is bounded $(0 < \alpha_i < 1)$ for any real value of β_i where β_i is an adjustable parameter

$$\alpha_i = \frac{1}{1 + e^{-\beta_i}} \tag{3}$$

The expertise level function is then multiplied by expertise domain function. Since the value of α_i is bounded $(0 < \alpha_i < 1)$, the peak value of expertise domain (Gaussian basis function) $\mu_k(d(\vec{x};\vec{c}_k,\vec{\sigma}_k))$ is gated by a weighting factor α_i. Therefore, the peak value of expertise domains varies according to the changes of β_i in accordance with the importance of their contribution to the overall network output.

Fig. 7 shows the effect of varying control parameter β on the peak value of expertise domain. In GMN, the initial value of β is always set to zero and it is subject to error-based learning. The slope of expertise level function $\alpha(\beta)$ is the greatest in the mid-range in the vicinity of $\beta = 0.5$. Near both extremes, the slope diminishes asymptotically towards zero as depicted in *Fig. 8*. Therefore, the expertise level value is sensitive to the onset of learning, which exhibits fast learning capability at the initial stage. This is closely similar to the learning curve of a human being.

Partition of Unity

For modeling tasks the expertise domain (Gaussian basis functions) should form a partition of unity for the input space, i.e. at any point in the input space, the sum of all expertise domains should be unity. This is an essential requirement for the network to be able to globally approximate any desired function as complex as the local expert model. The partition of unity also ensures that any variation in output over the input space is due only to the local expert model. Therefore, the gated expertise domains are normalized to achieve the partition of unity.

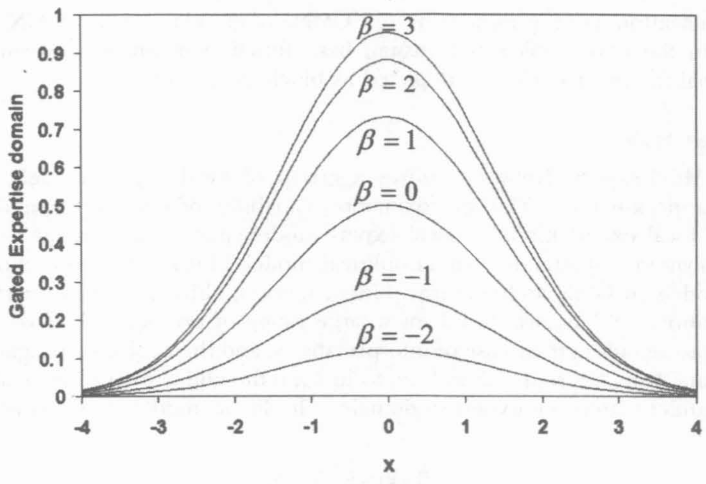


Fig. 7: β is the control parameter of expertise level function, $\alpha(\beta)$. The modifying value of β has the effect of suppressing the expertise domain

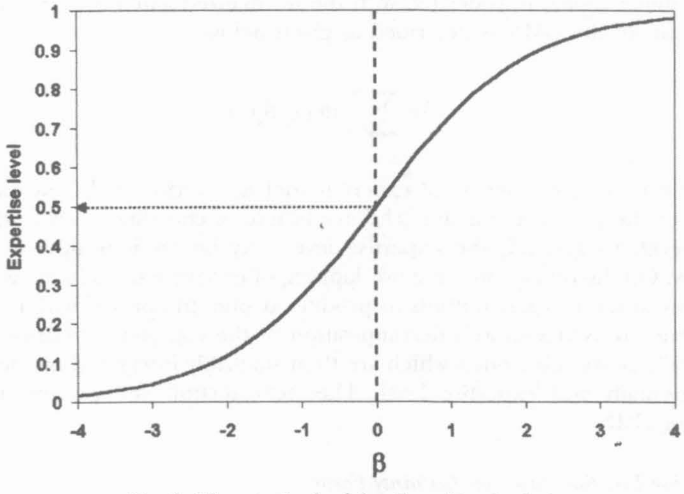


Fig. 8: The expertise level function of local experts

$$\hat{\Phi}_k(\vec{x}) = \frac{\alpha_k \cdot \mu_k\left(d\left(\vec{x};\vec{c}_k,\vec{\sigma}_k\right)\right)}{\sum_{k=1}^{m} \alpha_k \cdot \mu_k\left(d\left(\vec{x};\vec{c}_k,\vec{\sigma}_k\right)\right)} \tag{4}$$

where $\mu_k(d(\vec{x};\vec{c}_k,\vec{\sigma}_k))$ is the general unnormalised expertise domain, therefore the m normalized expertise domains $\hat{\Phi}_i(\vec{x})$ sum to unity.

$$\sum_{k=1}^{m} \hat{\Phi}_k(\vec{x}) = 1 \tag{5}$$

Normalization is very important for GMN, often making the GMN model less sensitive to the poor choice of Gaussian basis function parameters Werntges (1993). The normalization procedure is depicted as block 'N' in *Fig. 5.*

Local Expert Model

Growing Multi-experts Network utilizes a group of local expert models to perform function approximation. The approximation capability of GMN is dependent on the choice of local expert models. Local expert models may be a constant model, linear model, polynomial model or even a nonlinear model. Linear function is chosen as local expert models in GMN architecture. Linear models, although very restricted in their representational ability, are useful for a large range of problems. This is due to their simple representation, their ease of interpretability, and their robustness against noisy or missing data. It makes sense, therefore, to include the ability to at least be able to form a linear model within the expertise domain. The linear model is expressed as follows:

$$\bar{\theta}_k(\bar{x}) = \bar{x}^T \cdot \bar{b}_k + b_{0,k} \tag{6}$$

where $\bar{b}_k$ and $b_{0,k}$ are adjustable parameters of the local expert and $\bar{x}$ is the input vector. Each local linear model is associated with the normalized gated expertise domain. The overall output of the GMN is described as given below:

$$\hat{y} = \sum_{k=1}^{m} \hat{\bar{\Phi}}_k(\bar{x}) \cdot \bar{\theta}_k(\bar{x}) \tag{7}$$

The contribution of each local expert model is determined by the validity of the input data to the expertise domain. The magnitude of contribution is further gated by expertise level. In general, the expertise level may be an indicator of local expert redundancy. On the other hand, the overlapping of expertise domain allows the smooth combination of local expert outputs to produce a smooth approximation. The trained network structure is viewed as a decomposition of the complex, nonlinear system into a set of locally active sub-models which are then smoothly integrated by their associated expertise domain and expertise level. The next section will present the learning algorithm of GMN.

Local Expertise Insertion based on Certainty Factor

Certainty factor was introduced by Buchanan and Shortlife (1985) for a medical diagnostic system. Donald K. Wedding II and Krzysztof J. Cios conducted a comparative study of this method and the other technique for measuring RBF reliability Wedding *et al.* (1995). Certainty factor can be generated by GMN networks by using the $\mu(d(\bar{x};\bar{c},\bar{\sigma}))$ values from the expertise domain. Referring to Equation (2), it is obvious that this function returns values between 0.0 and 1.0 in the same range as allowed by certainty factors.

When the output $\mu(d(\bar{x};\bar{c},\bar{\sigma}))$ of an expertise domain is high (near 1.0), then this indicates that a point lies near the center of a cluster and therefore the network is confident about the prediction. A value of 1.0 exactly indicates that the data falls directly upon the center of an expertise domain. A value that is very small (near 0.0) will lie far outside the expertise domain, so the network is most likely to perform extrapolation. In this scheme, N values of $\mu(d(\vec{x};\vec{c},\vec{\sigma}))$ are generated, one for each of the local experts in the hidden layer of the GMN network. The following equation is used to calculate the certainty factor using the recursive formula Wedding (1995).

$$CF_i = CF_{i-1} + (1 - CF_{i-1})\max_i\left(\mu\left(d\left(\vec{x};\vec{c},\vec{\sigma}\right)\right)\right) \tag{8}$$

In this equation, the value of *i* refers to the number of local experts that will be used to determine the certainty factor of the GMN network. In general, the value of *i* should be set to N (the number of local experts in the hidden layer), but it can also be set to a lower value if computational time is critical to the application. The term CF_{i-1} refers to the certainty factor of the network using only $i - 1$ nodes to calculate the confidence level. The value of CF_0 (the certainty when local experts in the hidden layer are used) is defined to be 0.0. Lastly, the term $\max_i(\mu(d(\vec{x};\vec{c},\vec{\sigma})))$ is defined as the *ith* largest value of $\mu(d(\vec{x};\vec{c},\vec{\sigma}))$. So, $\max_1(\mu(d(\vec{x};\vec{c},\vec{\sigma})))$ would mean the largest value of $\mu(d(\vec{x};\vec{c},\vec{\sigma}))$ and $\max_2(\mu(d(\vec{x};\vec{c},\vec{\sigma})))$ would be the second largest value of $\mu(d(\vec{x};\vec{c},\vec{\sigma}))$.

On-line GMN Identification Algorithm

For online identification application using RBF network, a recursive algorithm is required for the learning process of GMN network. For an effective adaptive identification scheme, the learning process requires allocation of new local experts depending on the inputs as well as network parameter adaptation. The algorithm is explained as follows:

The network begins with two local experts that are initialized randomly. As observations are received some of the input data will be taken as the new local experts. The following two criteria decide whether an input x_n should be added to the hidden layer of the network:

$$\begin{aligned} &CF_i(n) \le \varepsilon_{cf} \\ &e_{rmsn} = \sqrt{\frac{\sum_{i=n-(M-1)}^{n}\left[y(i) - \hat{y}(i)\right]^2}{M}} \ge e_{oldrmsn} \end{aligned} \tag{9}$$

If the criteria above are fulfilled, then allocate a new local expert with

$$\begin{aligned} c_{new} &= x_n \\ \sigma_{new} &= \frac{\sum_{i=1}^{i=Nb}\left\|\vec{c}_i - \vec{c}_{nr}\right\|}{Nb} \\ a_{new} &= 0 \\ \psi_{new} &= 0 \end{aligned} \tag{10}$$

$\vec{c}_{nr}$ is the center of a local expert whose distance from the incoming input feature x_n is the nearest among those of all the other local experts. The spread of new expertise domain is calculated as the average distance of *Nb* neighborhood local experts from $\vec{c}_{nr}$.

If the criteria above are not fulfilled, then

$$\vec{c}_i^{\,n} = \vec{c}_i^{\,n-1} + \eta_c \cdot \left[\left(-\frac{\partial E^{n-1}}{\partial \vec{c}_i^{\,n-1}}\right) + \eta_{cM} \cdot \left(\vec{c}_i^{\,n-1} - \vec{c}_i^{\,n-2}\right)\right] \tag{11}$$

$$\vec{\sigma}_i^{\,n} = \vec{\sigma}_i^{\,n-1} + \eta_\sigma \cdot \left[\left(-\frac{\partial E^{n-1}}{\partial \vec{\sigma}_i^{\,n-1}}\right) + \eta_{\sigma M} \cdot \left(\vec{\sigma}_i^{\,n-1} - \vec{\sigma}_i^{\,n-2}\right)\right] \tag{12}$$

$$\alpha_i^{n} = \alpha_i^{n-1} + \eta_\alpha \cdot \left[\left(-\frac{\partial E^{n-1}}{\partial \alpha_i^{n-1}}\right) + \eta_{\alpha M} \cdot \left(\alpha_i^{n-1} - \alpha_i^{n-2}\right)\right] \tag{13}$$

$$\vec{P}_i^{\,n} = \frac{1}{\lambda} \cdot \left[\vec{P}_i^{\,n-1} - \frac{\vec{P}_i^{\,n-1} \cdot \vec{x}^{\,n} \cdot \vec{x}^{\,n^T} \cdot \vec{P}_i^{\,n-1^T}}{\dfrac{\lambda}{\Phi_i^n} + \vec{x}^{\,n^T} \vec{P}_i^{\,n-1} \vec{x}^{\,n}}\right] \tag{14}$$

$$\vec{\psi}_i^{\,n} = \vec{\psi}_i^{\,n-1} + \Phi_i^n \cdot P_i^n \cdot \vec{x}^{\,n} \cdot E_i^{n-1} \tag{15}$$

The expertise domain center $\vec{c}_i$, the expertise domain spread $\vec{\sigma}_i$ and the expertise level $\overline{\alpha}_i$ are learned using momentum-based gradient descent method where η_c, η_σ, η_α are learning rates and η_{cM}, $\eta_{\sigma M}$, $\eta_{\alpha M}$ are momentum rates. The local expertise linear models are learned by Forgetting Weighted Recursive Least Square learning algorithm Ljung (1983). The initial parameter vector $\vec{\varphi}_i(0)$ is set as zero. The initial covariance matrix $P(0)$ is assigned as one. To cope with changing environment, in general, λ is set a value within $0.9 < \lambda < 1$. For δ_{delete} consecutive observations, if the normalized expertise level of local expertise model is below the average expertise value, then prune the respective local expert.

$$\frac{\gamma_k}{\sum_{k=1}^{m} \gamma_k} < \frac{1}{m} \tag{16}$$

m is the total number of local experts.

Robot Arm Dynamics

A robot manipulator is typically modeled as a serial chain of n rigid bodies. In general, one end of the chain is fixed to some reference surface while the other end of the chain is free, thus forming an open kinematic chain of moving rigid bodies. The vector equation of motion of such a device is written in the form:

$$D(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = T_0 \tag{17}$$

This vector equation represents the dynamics of an interconnected chain of ideal rigid bodies, where is the vector of joint actuator torques and is the vector of generalized joint positions. It is a matrix, usually referred to as the manipulator mass matrix containing the kinetic energy functions of the manipulator. It represents torques arising from centrifugal and Coriolis forces caused by gravitational effects when the manipulator is moving in its work space. In general, equation (17) is very complicated for all but the simplest manipulator configuration. The matrices in equation (17) have distinctive properties Lewis *et al.* (1993) that are the direct results of the application of the Euler-Lagrange method of formulating the dynamic equations of the manipulator system. Properties such as symmetry, boundedness and positive definiteness enable the exploitation of the manipulator model to design a model based control law for tracking control application properties Lewis *et al.* (1993). The resulting control law generates actuator torques that cause the manipulator to follow a desired trajectory in its workspace while maintaining small tracking errors. Ideal performance is achieved only when complete knowledge of manipulator model is available. However, in reality, model uncertainty is inevitable even when the best effort has been put into the analysis and derivation of the dynamic model.

Nonlinear Variable Gain Proportional-Derivative (PD) Controllers

In general, the linear discrete-time PD controller can be written as

$$T_{pd}(t) = K_p\left(\theta_d(t) - \theta(t)\right) + K_d\left(\dot{\theta}_d(t) - \dot{\theta}(t)\right) \tag{18}$$

Now, we rewrite equation (7) to represent the general GMN controller in the following manner:

$$u(t) = \frac{\sum_{i=1}^{m} \alpha_i \Phi_i(\vec{x})}{\sum_{i=1}^{m} \alpha_i \Phi_i(\vec{x})} \cdot \left(b_{0i} + b_{1i}(x)x_1(t) + b_{2i}(x)x2_1(t)\right) \tag{19}$$

Without loss of generality, we assume a_{0j} = 0 for all j in GMN controller, x_1(t) = θ_d(t)−$\theta(t)$ and $x_2(t) = \dot{\theta}_d(t) - \dot{\theta}(t)$ and thus (18) can be written as

$$u(t) = \frac{\sum_{i=1}^{m} \alpha_i \Phi_i(\vec{x})}{\sum_{i=1}^{m} \alpha_i \Phi_i(\vec{x})} \cdot \left(b_{i1}(x)\,(\theta_d(t) - \theta(t)) + b_{2i}(x)\,(\dot{\theta}_d - \dot{\theta}(t))\right) \tag{20}$$

From equation (20), one sees that the GMN controller is actually a nonlinear controller with variable gain changing with state of the input variables $(\theta_d(t) - \theta(t))$ and $(\dot{\theta}_d(t) - \dot{\theta}(t))$. Thus, the output from GMN controller becomes a nonlinear variable gain control signal written as

$$T_{npd}(t) = \frac{\sum_{i=1}^{m} \alpha_i \Phi_i(\vec{x})}{\sum_{i=1}^{m} \alpha_i \Phi_i(\vec{x})} \cdot \left(b_{i1}(x)\,(\theta_d(t) - \theta(t)) + b_{2i}(x)\,(\dot{\theta}_d - \dot{\theta}(t))\right) \tag{21}$$

The hybrid linear PD controller and GMN becomes a nonlinear adaptive PD controller and can be written as

$$\begin{aligned} T_c(t) &= T_{pd}(t) + T_{npd}(t) \\ &= \left(K_p + \frac{\sum_{i=1}^{m} \alpha_i \Phi_i(\vec{x})}{\sum_{i=1}^{m} \alpha_i \Phi_i(\vec{x})} \cdot b_{1i}(x) \right) \left(\theta_d(t) - \theta(t)\right) + \\ &\quad \left(K_d + \frac{\sum_{i=1}^{m} \alpha_i \Phi_i(\vec{x})}{\sum_{i=1}^{m} \alpha_i \Phi_i(\vec{x})} \cdot b_{2i}(x) \right) \left(\dot{\theta}_d(t) - \dot{\theta}(t)\right) \end{aligned} \tag{22}$$

Feedback-Error Learning

Kawato and Mitsuo (1993) offer a complete solution to the inverse dynamics problems that use connectionist networks. Their work is based on the study of physiological information. They called this learning method feedback-error learning. In this paper, we adopt this learning scheme using feedback-error learning for a dynamic structural network applied to an adaptive nonlinear PD controller. In this learning scheme, the conventional PD controller serves the two purposes as the linear controller and as a stable trainer that provides error signal for the Growing Multi-Experts Network. It is shown in *Fig. 9.*

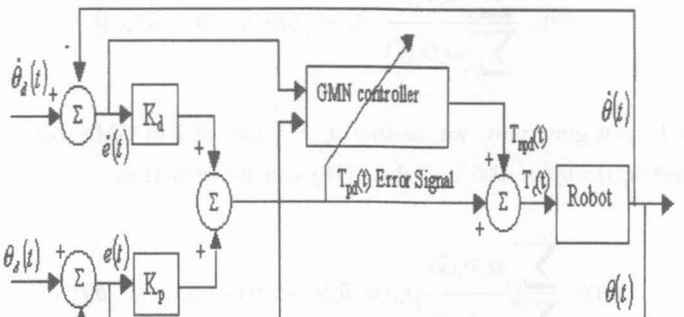


Fig. 9: Block diagram of nonlinear adaptive PD controller

Simulation Studies – Variable Load Compensation

The simulations are based on a situation that is often encountered when a 2-link robotic arm is used to transfer work pieces in a production line Song *et al.* (1994) as shown in *Fig. 10.* Suppose that the end-effector starts moving from position P_0 at $t_0 = 0$. It then picks up a container that contains $n_p = 4$ parts at time $t_p^+ = 2.0s$, moves to a workstation P_1 where $n_{d_1} = 2$ parts are removed from the container at time $t_{d_1}^+ = 4.0s$ and finally moves to work station P_2 where the remaining parts $n_{d_2} = 2$ are removed from the container at

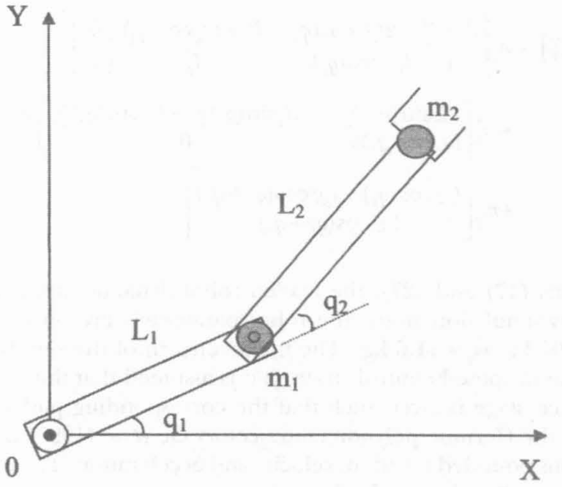


Fig. 10: The two-link robotic arm

time $t_{d_2}^+ = 6.0s$. The load transitions can suddenly change in robot dynamics and are unpredictable. So the control strategy developed in this paper is an attractive scheme in this application because it does not require exact values for the mass of the parts in the container, the pick-up time, or the drop-off times.

The payload variation model can be represented as

$$m_p^* = \left(\sum_{i=1}^{n_p} m_p(i) + m_c\right) U_p\left(t - t_p^+\right) - \sum_{i=1}^{n_{d_1}} m_p(i) U_{p_1}\left(t - t_{d_1}^+\right) - \sum_{i=1}^{n_{d_1}} m_p(i) U_{d_2}\left(t - t_{d_2}^+\right) \tag{23}$$

where m_p, m_c mare weight of the parts and container. They are assigned as 1.2 kg and 0.5 kg respectively. In the simulation, a two-link robot is used with the dynamics model from equation (23). The two-link robot is illustrated in *Fig. 10.*

$$D_0(q) = \begin{bmatrix} (m_1 + m_2)l_1^2 + m_2 l_2^2 + 2m_2 l_1 l_2 \cos(q_2) & m_2 l_2^2 + m_2 l_1 l_2 \cos(q_2) \\ m_2 l_2^2 + m_2 l_1 l_2 \cos(q_2) & m_2 l_2^2 \end{bmatrix} \tag{24}$$

$$C_0(q,\dot{q}) = \begin{bmatrix} (m_1 + m_2) l_1 l_2 \sin(q_2)\dot{q}_2 & -m_2 l_1 l_2 \sin(q_2)\dot{q}_1 - m_2 l_1 l_2 \sin(q_2)\dot{q}_2 \\ m_2 l_1 l_2 \sin(q_2)\dot{q}_1 & 0 \end{bmatrix} \tag{25}$$

$$G_0(q) = \begin{bmatrix} (m_1 + m_2) l_1 g \cos(q_1) + m_2 l_2 g \cos(q_1 + q_2) \\ m_2 l_2 g \cos(q_1 + q_2) \end{bmatrix} \tag{26}$$

where m_i and l_i represent the mass and length of link $i(i = 1,2)$, respectively; $g = 9.81\text{m/s}^2$ is the gravitational constant; and q_1 and q_2 are the angles for the joints. Note that in deriving these equations, it is assumed that the mass of each link is a point mass, concentrated at the distal end of the link. Let the parts and the container be treated as a point-mass payload, which has the total mass m_p^*.

$$\begin{aligned}T_L\left(m_p,q,\dot{q},\ddot{q}\right) &= m_p^*\begin{bmatrix} l_1^2+l_2^2+2l_1l_2\cos(q_2) & l_2^2+l_1l_2\cos(q_2) \\ l_2^2+l_1l_2\cos(q_2) & l_2^2\end{bmatrix}\begin{bmatrix}\ddot{q}_1\\ \ddot{q}_2\end{bmatrix} \\ &+m_p^*\begin{bmatrix} l_1l_2\sin(q_2)\dot{q}_2 & -l_1l_2\sin(q_2)\dot{q}_1-l_1l_2\sin(q_2)\dot{q}_2 \\ l_1l_2\cos(q_2)\dot{q}_1 & 0\end{bmatrix}\begin{bmatrix}\dot{q}_1\\ \dot{q}_2\end{bmatrix} \\ &+m_p^*\begin{bmatrix} l_2g\cos(q_1)+l_2g\cos(q_1+q_2) \\ l_2g\cos(q_1+q_2)\end{bmatrix}\end{aligned} \tag{27}$$

From equations (17) and (27), the loaded robot dynamic model can be expressed as T_0 + T_L. In this simulation study, the robot parameters are set as l_1 = 0.432 m; l_2 = 0.432 m; m_1 = 15.91 kg; m_2 = 11.6 kg;. The main concern of the simulation is to test the effectiveness of the proposed control strategy; it is assumed that the desired path for the end-effector at each stage is given such that the corresponding paths for the joints are determined to be the Hermite polynomial trajectory Ge *et al.* (1998) of the third degree in t with continuous bounded position, velocity and acceleration. The general expression for this kind of desired trajectory is given as:

$$q_d\left(t,t_d\right)=q_0+\left(-2.0\frac{t^3}{t_d^3}+3.0\frac{t^2}{t_d^2}\right)\left(q_f-q_0\right) \tag{28}$$

where q_0 and q_f are the arm initial and final positions, respectively, and t_d represents the time at which the desired arm trajectory reaches the desired final position. In this simulation example, the following values were chosen for the desired trajectories

$$t_d=2.0s,\quad q_d(0)=\left[0.0\ 0.0\right]^T rad,\quad q_d\left(t_d\right)=\left[1.0\ 2.0\right]^T rad$$

The gains for the PD-controller were chosen

$$K_p=\left[900\ 400\right],\quad K_d=\left[30\ 20\right]$$

On-line GMN parameter settings are depicted in Table 1.

TABLE 1
Parameter setting of on-line GMN for nonliear PD controller

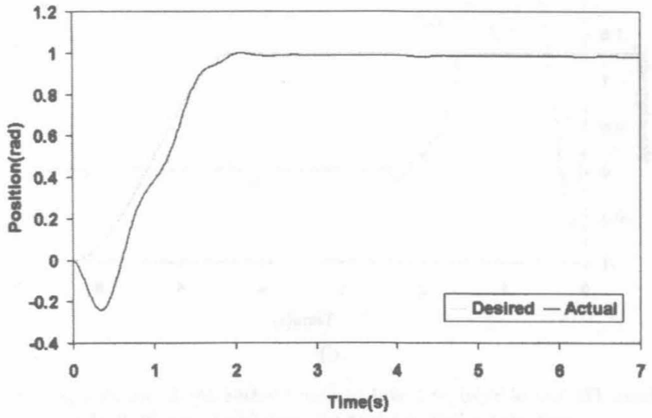
	1st Joint	2nd Joint
Certainty Factor, δ_{CF}	0.6	0.6
Pruning threshold, δ_{prune}	0.01	0.01
Adaptation parameter, η_c	0.01	0.01
Adaptation parameter, η_σ	0.01	0.01
Adaptation parameter, η_β	0.98	0.98
Momentum term, η_M	0.1	0.1
Forgetting factor, λ	0.998	0.998
Number of adaptation steps before pruning, δ_{delete}	100	100
Edge removal threshold, $age_{\max}$	500	500

Case 1: Non-adaptive Control

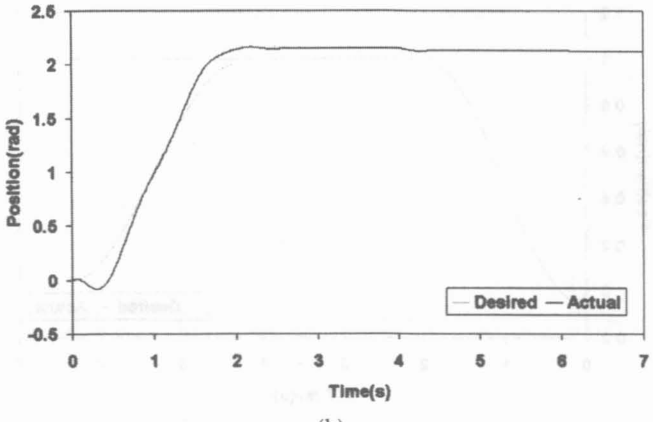
When the GMN controller is not activated, the resulting control action is effectively a simple PD control. It is observed from these results that the non-adaptive PD control has a significant joint angle and joint velocity tracking error due to the dynamic effects of load transition. *Figs. 11a* and *11b* show the desired trajectories and trajectories obtained from non-adaptive PD controller. Joint angle 2 has substantial tracking error as shown in *Fig. 11b*. *Figs. 11c* and *11d* illustrate the joint velocity tracking profile for first joint and second joint trajectory respectively.

Case 2: Adaptive Control

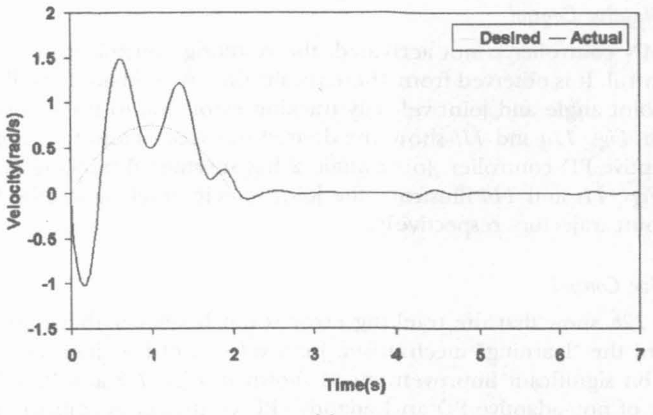
Figs. 12a and *12b*, show that the tracking error is much smaller than the non-adaptive case because of the "learning" mechanism. Joint velocity of first joint and second joint link also exhibit significant improvement as shown in *Figs. 12c* and *12d*. The position tracking error of non-adaptive PD and adaptive PD controller is compared in *Figs. 13a* and *13b*. It is clear that addition of GMN control signal makes a significant improvement in the tracking performance whereas *Figs. 14a* and *14b* indicate substantial reduction in velocity tracking error.



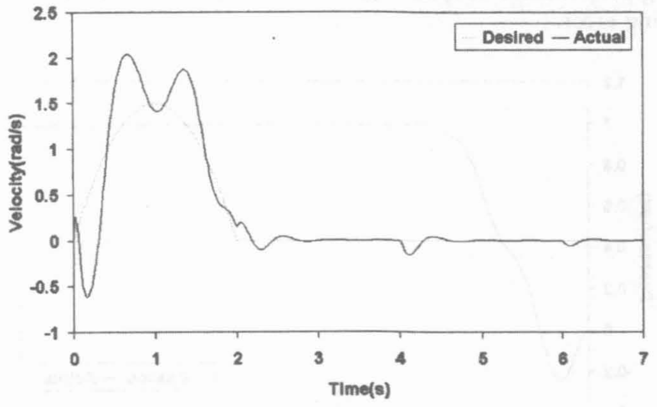
(a)



(b)

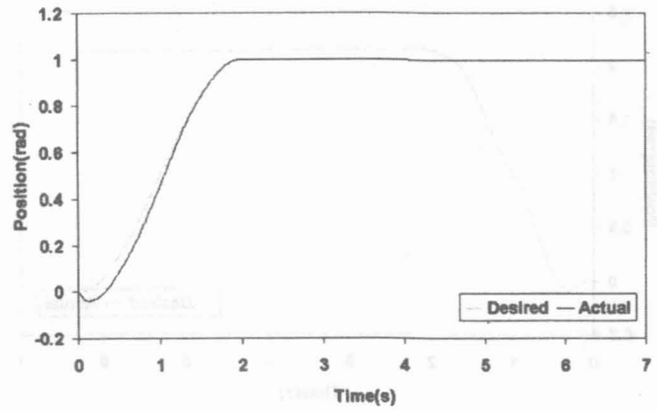


(c)

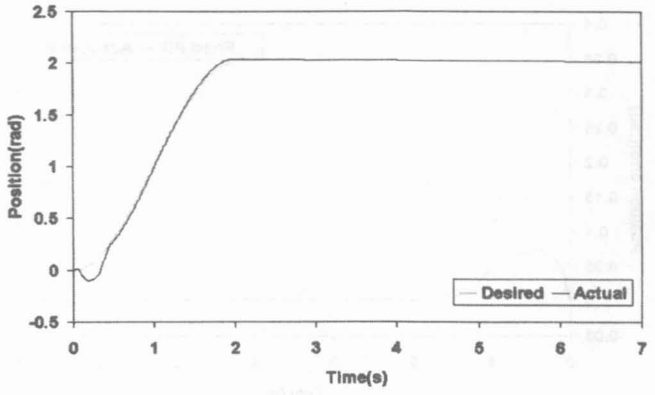


(d)

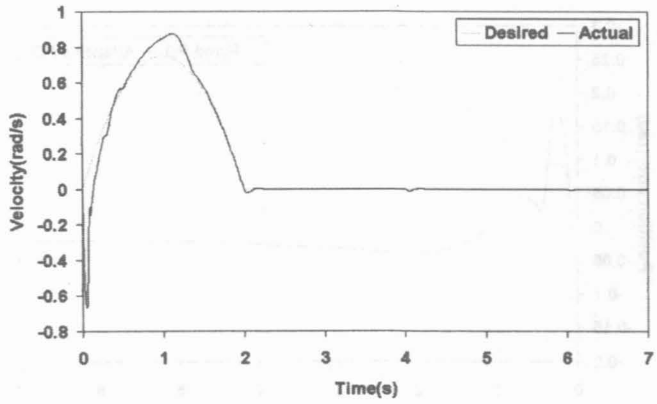
Fig. 11: Fixed PD controller (a) first joint position tracking (b) second joint position tracking (c) first joint velocity tracking (d) second joint velocity tracking



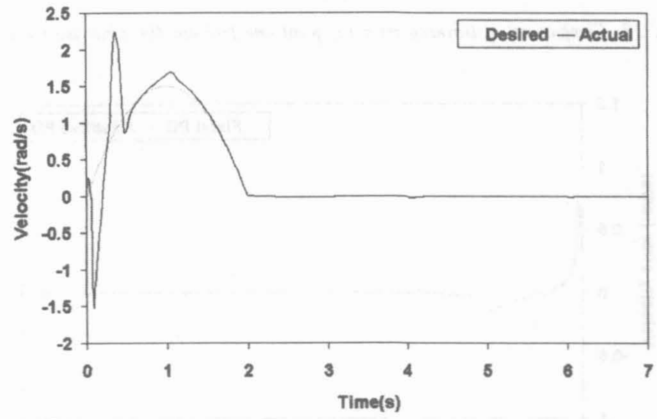
(a)



(b)

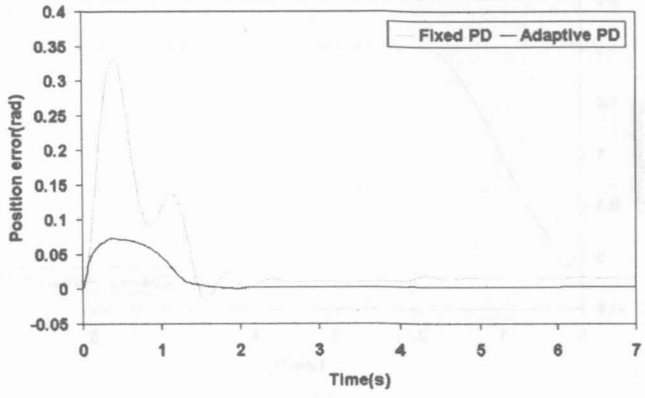


(c)

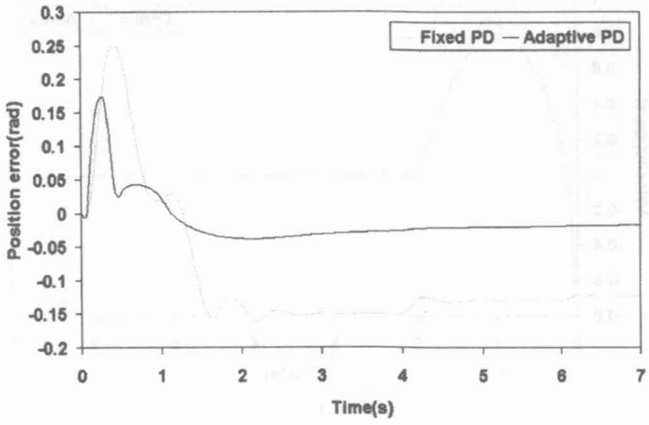


(d)

Fig. 12: Adaptive PD controller (a) first joint position tracking (b) second joint position tracking (c) first joint velocity tracking (d) second joint velocity tracking

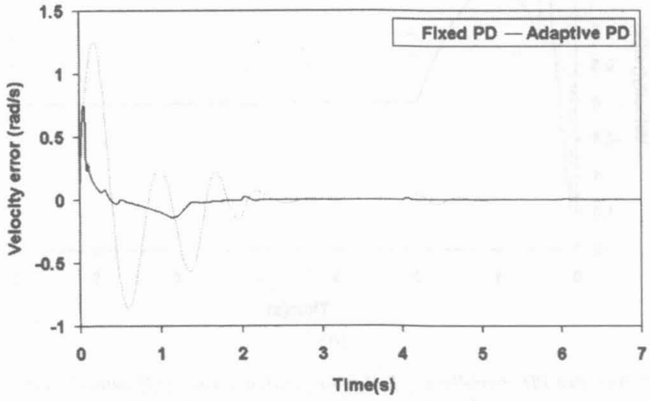


(a)

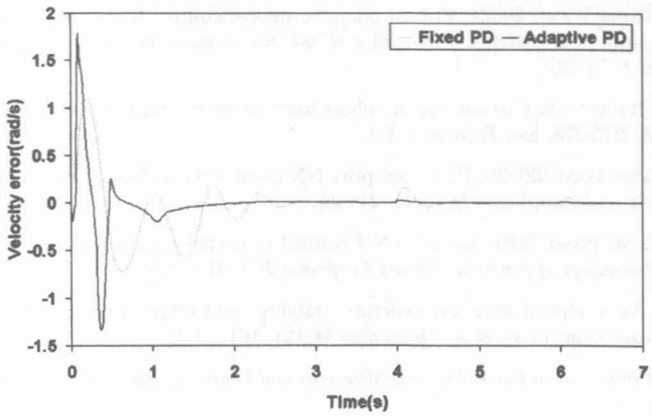


(b)

Fig. 13: Comparison of tracking error (a) joint one position (b) joint two position



(a)



(b)

Fig. 14: Comparison of tracking error (a) joint one velocity (b) joint two velocity

DISCUSSION

The proposed adaptive PD controller is compared with conventional PD controller. The comparison of the two schemes is based on computer simulation. Comparison of the position tracking performance for the two control schemes shows a substantial improvement in steady state error. The augmented control signals from GMN networks have successfully reduced the steady state error as shown in *Figs. 13a* and *13b*. The maximum tracking errors for joint one and two using GMN-based control scheme were found to be 0.10 and 0.05. In the case of conventional PD control scheme, the maximum tracking errors were 0.24 and 0.16 respectively. The reduction of tracking error is due to the efficacy of adaptive PD controller in compensating for the unstructured uncertainty caused by variable load in two-link robot.

CONCLUSION

We have presented a novel self-learning nonlinear PD controller using a class of dynamic structure network known as Growing Multi-Experts Network. This method results in a network that is "economic" in terms of network size and learns the required control behavior on the fly during the robot operation. The effectiveness of the nonlinear adaptive PD controller scheme was put to test with variable load and friction compensation in robot manipulators. The performance results demonstrate the learning capabilities and better tracking performance of nonlinear adaptive PD controller in comparison to conventional linear PD controller.

REFERENCES

BUCHANAN, B. G. and E. H. SHORTLIFE. 1985. *Rule-based Expert Systems.* Addison-Wesley, Reading, MA.

ERLIC, M. and W. S. LU. 1995. A reduced-order adaptive velocity observer for manipulator control. *IEEE Transactions on Robotics Automation* 11:293-303.

GE, S. S., T.H. LEE and C. J. HARRIS. 1998. *Adaptive Neural Network Control of Robotic Manipulators.* World Scientific Publishing Co. Pte. Ltd.

GE, S. S. and CONG WANG. 2002a. Robust adaptive neural control for a class of perturbed strict feedback nonlinear systems. *Proceedings of the 4th World Congress on Intelligent Control and Automation* **1**: 77-82.

GE, S. S. 1996. Robust adaptive control of robots based static neural networks. In *Proc. IFAC World Congress,* **A:** 273-278. San Francisco, CA.

GE, S. S. and CONG WANG. 2002b. Direct adaptive NN control for a class of nonlinear systems. *IEEE Transactions on Neural Networks* **13(1)**: 214-221.

GE, S. S. and CONG WANG. 2001. Adaptive NN control of partially known nonlinear strict-feedback systems. *Proceedings of American Control Conference* **2:** 1241 –1246.

HAO YIN. 1998. An analytical study on structure, stability and design of General Nonlinear Takagi-Sugeno Fuzzy Control systems. *Automatica* **34(12):** 1617-1623.

JOHN, J. CRAIG. 1989. *Introduction to Robotics, Mechanics and Control.* p. 205. Addison-Wesley Publishing Company.

KADIRKAMANATHAN, V. 1993. A function estimation approach to sequential learning with neural networks. *Neural Computation* **5:** 954-975.

KADIRKAMANATHAN, V. 1994. A statistical inference based growth criterion for the RBF network. In *Proc. IEEE Workshop. Neural Networks for Signal Processing* **IV**: 12-21.

KHALID, M. and S. OMATU. 1992. A neural network controller for a temperature control system. *IEEE Control System* **12(3):** 58-64.

LEWIS, F. L., K. LIU and A. YESILDIREK. 1995. Neural net robot controller with guaranteed tracking performance. *IEEE Trans. On Neural Network* **6(3)**: 703-715.

LJUNG, L. and T. SODERSTROM. 1983. *Theory and Practice of Recursive Identification.* Cambridge: MIT Press.

LOO, C. K. and M. RAJESWARI. 2000. Growing multi-experts network. *TENCON 2000,* Sept. Kuala Lumpur, Malaysia.

LOO, C. K., M. RAJESWARI and M. V. C. RAO. 2003. Extrapolation detection and novelty-based node insertion for sequential growing multi-experts network. *Applied Soft Computing* **3(2)**: 159-175.

MIYAMOTO, H., M. KAWATO, T. SETOYAMA and R. SUZUKI. 1993. Feedback-error learning for a closed-loop system using feedback-error-learning. *Neural Networks* **6:** 933-946.

OZAKI, T., T. SUZUKI, T. FURUHASHI, S. OKUMA and Y. UCHIKAWA. 1991. Trajectory control of robotic manipulators using neural networks. *IEEE Transactions on Industrial Electronics* **38(3)**: 185-202.

PSALTIS, D., A. SIDERIS and A. A. YAMAMURA. 1988. A multilayered neural network controller. *IEEE Control Systems Magazine* **8:** 17-21.

SONG, Y. D. and J. N. ANDERSON. 1994. The dynamic effects and compensation of load transitions in robotic systems. *Journal of Robotic Systems* **11(6):** 541-556.

WEDDING, D. K. and K. J. CIOS. 1995. Time series forecasting by combining RBF neural networks and the Box-Jenkins model. *Neurocomputing* **9:** 149-168.

WERNTGES, H. W. 1993. Partitions of unity improve neural function approximation. In *Proc. IEEE Int. Conf. Neural Networks,* **2:** 914-918. San Francisco, CA.