A Learning Control Scheme Based on Neural Networks for Repeatable Robot Trajectory Tracking

Jizhong Xiao*, Qing Song**, Danwei Wang**
*Robotics Research Center, School of MPE
**School of Electrical and Electronic Engineering
Nanyang Technological University, Singapore 639798
E-mail: mjzxiao@ntu.edu.sg

Abstract
This paper presents an iterative learning controller using neural network (NN) for the robot trajectory tracking control. Basic control configuration is briefly presented and a new weight-tuning algorithm of NN is proposed with a dead-zone technique. Theoretical proof is given which shows that our modified algorithm guarantees the convergence of NN estimation error in the presence of disturbance. The simulation study demonstrates that the proposed weight-tuning algorithm is robust and less sensitive to noise compared to the standard back-propagation (BP) algorithm in identifying the robot inverse dynamics. Moreover, the simulation results also shows that the proposed NN learning control scheme can greatly reduce tracking errors as the iteration number increases.

1. Introduction
Research on iterative learning control for robot system has received considerable attention over the past decades [1-3]. Learning control methodologies make use of the repetitive nature of tasks to obtain improved performance without the necessity of a parametric model of the system. Hence, this technique is suitable for a large class of industrial robot manipulators, which essentially carry out repetitive tasks and the accurate model of robot system is normally unavailable.

The ability to learn a non-linear mapping and the non-requirement of a priori information about the system model make the NN an attractive alternative to the conventional learning control scheme. Various design methods have been proposed in the application of NN to the robot control field. The differences in those schemes are largely in the role that NN is playing in the control system and the way it is trained. The most popular control scheme is one which utilise the learning ability of NN to identify the system inverse and thus generate control input to the plant [4-7]. As for the training method of NN, back-propagation (BP) algorithm is dominantly used in the control field. However, conventional BP algorithm of NN may become divergent in the presence of noise [4,9]. Some of the theoretical results are examined in [9-13] on the weight updating law and the convergence performance of NN controller.

In this paper, we propose a robust weight-tuning algorithm for multi-layer NN using the dead-zone technique to estimate the inverse dynamics of the robot. Theoretical analysis shows that this algorithm guarantees the convergence of NN estimation error in the presence of disturbance. Thus, when it is employed in the neural learning control scheme, the repeatable structured & unstructured uncertainties of robot systems can be compensated. Simulation results also demonstrate that the neural learning controller can greatly reduce tracking errors after several iterations and the NN weight-tuning algorithm with dead-zone technique is robust to noise.

2. The Structure of Neural Learning Controller
A practical digital control system for a $n$-DOF manipulator can be modelled as follows [14]:

$$\text{Tor}(k) = M(q(k)) \cdot \dot{q}(k) + V(q(k), \dot{q}(k)) + G(q(k)) + F(q(k)) + T_e + d$$

(1)

where $T_e$ account for modelling error and other repeatable structured and unstructured uncertainties. $d$ represents non-repeatable random disturbance. The $n \times 1$ vectors, $q, \dot{q}, \ddot{q}$ are the joint angle, joint angular velocity, and joint angular acceleration, respectively. $M(q)$ is the $n \times n$ inertia matrix; $V(q, \dot{q})$ is the $n \times 1$ vector of coriolis and centripetal torque, $F(\dot{q})$ is the $n \times 1$ vector representing coulomb friction and viscous friction force; $G(q)$ is the $n \times 1$ gravitational torque.

The proposed neural learning controller structure for practical digital control robot is shown in Fig 1. This scheme consists of a feedback PD controller and a feedforward neural controller. In the feedback loop, the fixed gain PD controller makes the overall system stable along a desired trajectory. In the feedforward path, a NN is used to predict the desired actuator torque.
Torque Error

is a constant parameter. can be presented by the NN as algorithm using dead-zone technique to reject noise.

propose a three-layer NN and a modified weight-tuning training method. To ensure the convergence of the NN, we control schemes. However, the difference lies in the NN

The overall control system has a learning-then-control property of the NN is exploited to estimate the robot inverse dynamics off-line based on input-output pairs gathered in the (iteration.

Assuming \( q_d(k) \) is the desired robot motion trajectory, then the control scheme can be described as:

\[
Tor'(k) = U_{\beta}'(k) + U_j'(k) = M(q_j(k)) \cdot \dot{q}_j(k) + V(q_j(k), \dot{q}_j(k)) + G(q_j(k)) + F(q_j(k)) + T_e + K_e(q_j(k) - q(k)) + K_v(\dot{q}_s(k) - \dot{q}(k))
\]

The superscript \( j \) of each variable indicates its value at the \( j \)th iteration. \( U_{\beta}'(k) \) is the feedforward compensation term. The aim of the NN feedforward controller is to find a network which can accurately learn the robot inverse dynamics and thus provide approximation of the non-linear function \( U_j'(k) \). To achieve this, an off-line training process is needed. Thus, the iterative neural learning controller is then operated in two processes: training period and control period.

During the training period, universal approximation property of the NN is exploited to estimate the robot inverse dynamics off-line based on input-output pairs gathered in the \( j \)-th operation cycle. During the control period, the weights of the well-trained NN controller are fixed and provide feedforward control signal to compensate for the repeatable uncertainty and non-linear effects. After each operation of the robot, the NN will be retrained with the operation data obtained from previous iteration.

3. Feedforward NN Learning Controller Design

The overall control system has a learning-then-control feature, which may look similar to the other learning control schemes. However, the difference lies in the NN training method. To ensure the convergence of the NN, we propose a three-layer NN and a modified weight-tuning algorithm using dead-zone technique to reject noise.

![Fig. 1 Scheme of neural learning controller (jth iteration, control period)](image1)

Assuming \( q_d(k) \) is the desired robot motion trajectory, then the control scheme can be described as:

\[
Tor'(k) = U_{\beta}'(k) + U_j'(k) = M(q_j(k)) \cdot \dot{q}_j(k) + V(q_j(k), \dot{q}_j(k)) + G(q_j(k)) + F(q_j(k)) + T_e + K_e(q_j(k) - q(k)) + K_v(\dot{q}_s(k) - \dot{q}(k))
\]

The superscript \( j \) of each variable indicates its value at the \( j \)th iteration. \( U_{\beta}'(k) \) is the feedforward compensation term. The aim of the NN feedforward controller is to find a network which can accurately learn the robot inverse dynamics and thus provide approximation of the non-linear function \( U_j'(k) \). To achieve this, an off-line training process is needed. Thus, the iterative neural learning controller is then operated in two processes: training period and control period.

![Fig. 2 Training scheme of the neural network](image2)

During the training period, universal approximation property of the NN is exploited to estimate the robot inverse dynamics off-line based on input-output pairs gathered in the \( j \)-th operation cycle. During the control period, the weights of the well-trained NN controller are fixed and provide feedforward control signal to compensate for the repeatable uncertainty and non-linear effects. After each operation of the robot, the NN will be retrained with the operation data obtained from previous iteration.

The Structure of the NN

![Fig. 3 The structure of the multi-layer neural network](image3)

Fig. 3 shows the structure of the multi-layer NN. The input to the NN is the actual operation vector \( x(i) = [q^T, \dot{q}^T, \ddot{q}^T] \in \mathbb{R}^n \). Each input state passes through a non-linear threshold activation function \( h(x) \) and the first layer of the NN is \( \Pi(i) = [h(x_1(i)) \ldots h(x_{3n+1}(i))]^T \in \mathbb{R}^{3n+1} \), with the selected activation function as

\[
h(x) = \frac{1}{1 + e^{-\lambda x}}
\]

where \( \lambda > 0 \) is a constant parameter.

The output of the NN can be computed as:

\[
Tor_c(\Pi(i)) = v_0(i) + w(i)^T H(\Pi(i), \Pi(i))
\]

where \( v_0(i) \in \mathbb{R}^n \) and \( w(i) \in \mathbb{R}^{n \times n_h} \) are the adjustable biases and weights of the output layer of the NN; \( n, \ n_h \) is the number of neurons in the output layer and hidden layer respectively. \( w(i) = [w_{ij}(i)] \in \mathbb{R}^{n_h \times (3n+1)} \) is the adjustable weight matrix, with \( w_{ij}(i) \) linked the \( j \)th neuron of the input layer to the \( i \)th neuron of the hidden layer. \( H(w(i), \Pi(i)) \) is the output of the hidden layer, which is a nonlinear vector:

\[
H(w(i), \Pi(i)) = [h(\sum_{j=1}^{n} w_{ij} \Pi_j) \ldots h(\sum_{j=1}^{n} w_{ij} \Pi_j)]^T
\]

Note that including “1” as the first term in the vector \( \Pi(i) \) allows one to incorporate the bias as the first column of \( \Pi(i) \), so that \( w(i) \) contains both the weights and biases of the first-to-second layer connection. We use the notation \( || \cdot || \) for both the Euclidean norm of vector and the Frobenius norm of matrix throughout this paper.

Assuming that the ideal weights of the NN are \( v^*, \ v_0^* \) and \( w^* \), then \( Tor \) can be presented by the NN as

\[
Tor = Tor_c(\Pi(i)) = v_0^* + w^T H(w^*, \Pi(i)) + \varepsilon(i)
\]

where \( \varepsilon(i) \in \mathbb{R}^n \) is a reconstruction and measurement error vector with the boundary \( ||\varepsilon(i)|| \leq \varepsilon_n \).
Thus, NN estimate error \( e(i) \in R^n \) is given by
\[
e(i) = T_{or}(x(i)) - T_{or}(x(i)) = v_i^T(H(w_i,i),\tau(i)) + v_i^T(H(i) + e(i))
\]
where \( H(i) \in R^{n \times 1} \) is defined as
\[
H(i) = H(w_i,i,\tau(i)) - H(w_i,i,\tau(i)) = \left[ \frac{1}{H(w_i,i,\tau(i))} \right] \left[ \frac{1}{H(w_i,i,\tau(i))} \right]
\]
and \( \tilde{v}(i) \in R^{m \times n} \) is the weight error matrix of the output layer: \( \tilde{v}(i) = [\tilde{v}_i] = \left[ \begin{array}{c} \tilde{v}_1^T \\ \tilde{v}_2^T \\ \vdots \\ \tilde{v}_n^T \end{array} \right] \).

3.2 Robust Training Algorithm for the Output Layer

To update the weight \( \tilde{v}(i) \), the robust weight-tuning algorithm can be written in the matrix form
\[
\tilde{v}(i+1) = \tilde{v}(i) - \alpha(i) \frac{\partial e(i)}{\partial \tilde{v}(i)} = \tilde{v}(i) + \alpha(i)H(w_i,i,\tau(i))e(i)
\]
where \( E(i) = \frac{1}{2}e^T(i)e(i) \) is the cost function and \( \alpha(i) \) is a variable learning rate based on the dead zone scheme defined as
\[
\alpha(k) = \begin{cases} 1 & \text{if } \| e(i) \| > \Delta_x, \\ 0 & \text{if } \| e(i) \| \leq \Delta_x, \end{cases}
\]
and \( \Delta_x = \frac{2(v_v\tilde{H}_m + e_v)}{\sqrt{3 - 2H_m^2}} \) with \( 0 < H_m^2 < \frac{3}{2} \), with \( \| \tilde{v} \| \leq v_m \) and \( \| \tilde{H} \| \leq R_m \).

**Theorem 1:** The robust training algorithm in equation (8) using the dead zone scheme has the following properties
\[
\| \tilde{v}(i+1) \| \leq \| \tilde{v}(i) \| \quad \text{for} \quad i = 0,1,\ldots
\]
\[
\text{Proof:} \quad \text{From equation (7), we have}
\]
\[
\| e(i) \| = e^T(i)e(i) = e^T(i)H(w_i,i,\tau(i)) + e^T(i)\Delta(i)
\]
\[
\leq \text{tr} (e^T(i)e^T(i)H(w_i,i,\tau(i))) + \| e^T(i)\|\| \Delta(i) \|
\]
with \( \Delta(i) = \tilde{v}^T(i)\tilde{H}(i) + e(i) \).

Combining equations (8) and (12), we have
\[
\| \tilde{v}(i+1) \| - \| \tilde{v}(i) \| \
\leq -2\alpha(i)\text{tr} (H(w_i,i,\tau(i)))e^T(i)e(i) + (\alpha(i))^2\| H(w_i,i,\tau(i))e^T(i) \|
\]
\[
= -2\alpha(i)\text{tr} (e^T(i)e^T(i)H(w_i,i,\tau(i))) + (\alpha(i))^2\| H(w_i,i,\tau(i))e^T(i) \|
\]
\[
\leq -2\alpha(i)(\| e(i) \| - \| e(i) \| - \| e(i) \|) + (\alpha(i))^2\| H(w_i,i,\tau(i))e^T(i) \|
\]
\[
= 2\alpha(i)(\| e(i) \|^2 + \| e(i) \| - \| e(i) \| + (\alpha(i))^2\| H(w_i,i,\tau(i))e^T(i) \|
\]
\[
\leq 2\alpha(i)(\| e(i) \|^2 + \| e(i) \| - \| e(i) \| + (\alpha(i))^2\| H(w_i,i,\tau(i))e^T(i) \|
\]
\[
\leq 2\alpha(i)(\| e(i) \|^2 + \| e(i) \| - \| e(i) \| + (\alpha(i))^2\| H(w_i,i,\tau(i))e^T(i) \|
\]
\[
= \alpha(i)(\| e(i) \|^2 - 4(v_v\tilde{H}_m + e_v)^2)
\]
\[
\leq \alpha(i)(\| e(i) \|^2 - 4(v_v\tilde{H}_m + e_v)^2)
\]
\[
\leq -\frac{\alpha(i)}{2}[3 - 2H_m^2]\| e(i) \|^2 - 4(v_v\tilde{H}_m + e_v)^2]
\]
\[
\text{Note that the second equality above is from the property of } t(AB) = tr(AB), \text{ if } A \text{ and } B \text{ are of the same dimension, the first inequality is from the definition that } \alpha(i) \text{ is either one or zero as defined in equation (9), the second inequality is due to the inequality } 2ab \leq a^2 + b^2 / 2 [15].
\]

According to the definition of \( \alpha(i) \) in equation (9), we have
\[
\| e(i) \|^2 \leq \frac{2(\| v_v\tilde{H}_m + e_v \|)}{\sqrt{3 - 2H_m^2}} = \Delta_x.
\]
(15)

This completes the proof.

3.3 Robust Training Algorithm for the Hidden Layer

To update \( w(i) \), the robust weight-tuning algorithm can be written in the matrix form
\[
w(i+1) = w(i) - \beta(i) \frac{\partial e(i)}{\partial w(i)} = w(i) + \beta(i)H(i)e(i)v(i)\tilde{v}(i)
\]
(16)

with \( H(i) = \text{diag}(h_1(i),...,h_n(i)) \), where
\[
h(i) = \text{diag}(h(w_i,i,\tau(i)),...,h(w_n,i,\tau(i))) \in R^{n \times n},
\]
and \( \beta(i) \) is a variable learning rate based on the dead zone scheme defined as
\[
\beta(k) = \begin{cases} 1 & \text{if } \| e(i) \| > \Delta_x, \\ 0 & \text{if } \| e(i) \| \leq \Delta_x, \end{cases}
\]
and the range of the dead zone is given by
\[
\Delta_x = \frac{2\sqrt{h_{\min}e_{\min}}}{\sqrt{3h_{\min} - 2h_n\lambda_n^2\tau_{\max}}} \quad \text{with} \quad 0 < \frac{h_{\min}\lambda_n^2\tau_{\max}}{h_{\min}} < \frac{3}{2}
\]
(18)

where \( h_{\min} = \min(h_1(i),...,h_n(i)) > 0 \) is the minimum value of the derivative of the threshold function for a bounded signal \( \| \tilde{v}(i) \| \leq \tau_{\max}, \| \tilde{v}(i) \| \leq \tilde{e}_m, \) with \( \tilde{e}(i) = v_v^T(i)\tilde{H}(w_v,i,\tau(i)) + e(i) \), and \( \| e(i) \| \leq v_{\max} \).

**Theorem 2:** The robust training algorithm in equation (16) using the dead zone scheme has the following properties
\[
\| \tilde{v}(i+1) \| \leq \| \tilde{v}(i) \| \quad \text{for} \quad i = 0,1,\ldots
\]
\[
\text{Proof:} \quad \text{Note that the activation function } h(.) \text{ in equation (3) is non-decreasing function, so there exists a unique positive number } \mu(.) \text{ with } \lambda \geq \mu(.) \geq 0, \text{ because the minimum}
\]

value of \( h'(i) \) is \( \lambda \), such that the system estimation error \( e(i) \) can be presented as the following:

\[
e(i) = Tor(x(i)) - Tor_{nn}(x(i)) = v'(i)(H(x(i))) - H(v(i), x(i)) + \tilde{e}(i) = v'(i)(h(w_{n1}^T, x(i)) - h(w_{n2}^T, x(i)) - h(w_{n3}^T, x(i))) + \tilde{e}(i)
\]

where \( \tilde{e}(i) = w^* - w(i) \in \mathbb{R}^n \) is the weight error matrix, and \( \mu(i) \in \mathbb{R}^{n \times n} \) is a unique positive matrix.

\[
\mu(i) = \text{diag} \{ \mu_1(i), ..., \mu_n(i) \}
\]

Then from equation (21) we have

\[
e(i) = e'(i) + v'(i)(H'(i) \mu(i) \tilde{w}(i) + e'(i) \tilde{e}(i) \leq \frac{\mu(i)}{h'(i)}e'(i) + \frac{h'(i)}{\mu(i)}e'(i)v(i) + \frac{h'(i)}{\mu(i)}e'(i)\tilde{w}(i)
\]

Similarly, we have

\[
\limsup_{i \to \infty} ||e(i)|| \leq \frac{2\sqrt{\lambda h_{\text{max}} \hat{e}_m}}{\sqrt{h_{\text{max}} - 2n \lambda v_{\text{max}}^2}} = \Delta_e
\]

This completes the proof.

**Remarks:**

The theorems show that the NN estimation error will converge to a ball of radius \( \Delta = \min(\Delta_1, \Delta_2) \) and the NN weight errors are also bounded. Moreover, according to equation (18), we can choose a smaller gain parameter \( \lambda \) to have a smaller estimate error \( e(i) \). This is also necessary to make the term inside square root of the denominator of equation (18) be positive. A smaller \( \lambda \), of course, will lead to a slow neural network learning procedure. A design tradeoff is needed between learning speed and accuracy.

Note that since the teaching signal of NN are recorded from the actual operation, such input-output pairs definitely represent the actual inverse dynamics of robot which include repeatable structured and unstructured uncertainties. The success of the iterative neural learning control scheme depends largely on whether the NN model can converge the actual system inverse dynamic model. This condition can be guaranteed by the well-known universal approximation property of NN [8]. Thus repeatable uncertainties, which can be learned during the training process, can be compensated. However, from a practical point of view, except for the repeatable disturbance, there exist random uncertainties to robot system that are unpredictable and, thus, cannot be learned during training.

The standard BP algorithm is sensitive to such kind of noise and the learning process is degraded. We see from the proof procedure that if the learning rate is a constant, as in the standard BP algorithm, we can not draw conclusion that \( \|v'(i)\| \) and \( \|\tilde{w}(i)\| \) are non-increasing sequence as equation (13) and equation (24) show. So that nothing may be said about the convergence property of the standard BP algorithm. On the contrary, the tuning algorithm with dead-zone scheme proposed here assures the convergence of the neural network. This is the main advantage of our proposed algorithm over the standard BP algorithm.

The general idea of a dead-zone is to stop updating the weights when the NN is insufficient to distinguish between the signal and the noise. The term “robust” comes from the fact that dead-zone scheme have noise rejection ability which is commonly known in adaptive control community. Our proposed robust algorithm with dead-zone has noise rejection ability thus it guarantees the convergence of NN estimation error in the present of disturbance. The simulation results will confirm this.
4 Simulation Studies

In this section, we apply the iterative NN learning control scheme to a realistic two-link direct-drive manipulator system [17].

The dynamic model of the manipulator is as follows:

\[ \text{Tor} = M(q)\ddot{q} + V(q, \dot{q}) + F(\dot{q}) \],

where

\[ M(q) = \begin{bmatrix} p_1 + 2p_2 \cos(q_2) & p_2 + p_1 \cos(q_2) \\ p_2 + p_1 \cos(q_2) & p_2 \end{bmatrix} \]

is configuration dependent inertia matrix.

\[ V(q, \dot{q}) = -\dot{q}_1(2\dot{q}_1 + \dot{q}_2)p_1 \sin(q_2) \]

is a vector representing centrifugal and coriolis effects.

\[ F(\dot{q}) = \begin{bmatrix} k_{f1} \text{sgn}(\dot{q}_1) \\ k_{f2} \text{sgn}(\dot{q}_2) \end{bmatrix} \]

is a vector representing coulomb friction. \( p_1, p_2, p_3 \) are constants related to link mass, link length, inertia and other physical parameters of the robot. \( k_{f1}, k_{f2} \) are coulomb friction coefficient.

The desired trajectory of robot is selected as

\[ \begin{bmatrix} q_{d1} \\ q_{d2} \end{bmatrix} = \begin{bmatrix} \frac{\pi}{2}t - \frac{1}{2}\sin\pi \\ \frac{\pi}{2}\pi - \frac{1}{2}\cos\pi \end{bmatrix} \]

The defaulted values of PD gains for the actual direct-drive manipulator system are \( K_p = \text{diag}(100, 25) \) and \( K_d = \text{diag}(2, 2) \). So we select this values as PD gains in simulation for both the computed torque method and the NN iterative learning scheme. The control period for each iteration is 2s with sampling time selected as 5ms.

For the first iteration, we employ the computed torque method based on the nominal parameters to bring the robot into the neighbourhood of the desired trajectory. Using the operation data obtained from the results of the computed torque method we employed a three-layer NN with six input neurons, ten hidden neurons and two output neurons to estimate the inverse dynamics of the robot off-line. The proposed weight-tuning algorithm with dead-zone is employed in the training process. After finishing the training of the NN, we implement the iterative learning controller as described in Fig. 1. Note that the weights of the NN are fixed during control process and the desired trajectory is the input to the NN. After the control process, the NN is retrained based on the teaching signal obtained from the previous operation data and the new weights of NN are obtained. Fig. 4 depicts the simulation results of the computed torque method and Fig. 5 displays the tracking results of the seventh iteration. Fig. 6 presents the absolute sums of the position tracking errors and the absolute sums of the velocity tracking errors respectively.

We can see from these figures that there is a significant improvement in tracking performance as the number of the iterations increases. (Note that the performance index of the 0th iteration is recorded from the computed torque method.)

Although most industrial robots perform repetitive tasks, it is usual that the robot systems are subject to a lot of non-repeatable uncertainties from diverse sources such as noise in the sensor readings or errors in the execution of motion commands. To compare the robustness to noise of the standard BP algorithm with our proposed weight-tuning algorithm, we add disturbance to the input-output data of the seventh iteration in the simulation.

The external disturbance, which is added to the torque exerted on each joint, is supposed to be \( d(t) = \sin(\pi t)^2 + \cos(4\pi t) \). A 10% velocity measurement noise is also introduced. Such noise-contaminated data is used to train the NN using standard BP algorithm and our proposed robust weight-tuning algorithm respectively. Fig. 7 (a), (b) show that when fed with the desired trajectory, the feedforward torque produced by the NN trained with BP algorithm is much more different from the desired torque compared with our proposed robust weight-tuning algorithm. This is due to the fact that the noisy data causes large divergence of the NN weights from the ideal ones. Consequently, the tracking performance severely degraded. Fig. 8 and Fig. 9 describe the simulation results of the eighth iteration control using the two difference
algorithms. So that, we conclude that our proposed NN weight-tuning algorithm is robust and less sensitive to noise compared to the standard BP algorithm.

![Fig. 7 Output torque from NN (a) trained with BP algorithm, (b) trained with robust tuning algorithm](image)

**Fig. 7 Output torque from NN (a) trained with BP algorithm, (b) trained with robust tuning algorithm**

![Fig. 8 Tracking performance, using NN trained with BP algorithm (8th iteration)](image)

**Fig. 8 Tracking performance, using NN trained with BP algorithm (8th iteration)**

![Fig. 9 Tracking performance, using NN trained with robust tuning algorithm (8th iteration)](image)

**Fig. 9 Tracking performance, using NN trained with robust tuning algorithm (8th iteration)**

### 5 Conclusions

An iterative learning control scheme comprising a NN feedforward controller and a linear feedback controller is presented. The scheme applies a NN to construct feedforward control signal by making use of the previous operation input-output data. A robust weight-tuning algorithm is proposed using dead-zone technique to assure the convergence of the NN to the actual inverse system. A case study on a realistic two-link direct-drive robot demonstrates that the proposed neural learning controller is very promising, which achieves satisfactory tracking performance after a few iterations. This controller has the following salient features: 1. It has the ability to handle the repeatable structured and unstructured uncertainties. 2. The scheme has a learning-then-control feature thus it avoids the disadvantage of heavy on-line computation. 3. It is robust and less sensitive to noise compared to the standard BP algorithm. 4. It is suitable for real time implementation.

### 6 References


