ROBOT LEARNING CONTROL BASED ON NEURAL NETWORK PREDICTION *

Jonathan Asensio
Dept. of Systems Engineering & Control
Polytechnic University of Valencia
Valencia, 46022, Spain
Email: jonathan.asensio@gmail.com

Wenjie Chen †
Dept. of Mechanical Engineering
University of California
Berkeley, California 94720, USA
Email: wjchen@me.berkeley.edu

Masayoshi Tomizuka
Dept. of Mechanical Engineering
University of California
Berkeley, California 94720, USA
Email: tomizuka@me.berkeley.edu

ABSTRACT

Learning feedforward control based on the available dynamic/kinematic system model and sensor information is generally effective for reducing the repeatable errors of a learned trajectory. For new trajectories, however, the system cannot benefit from previous learning data and it has to go through the learning process again to regain its performance. In industrial applications, this means production line has to stop for learning, and the overall productivity of the process is compromised. To solve this problem, this paper proposes a learning control scheme based on neural network (NN) prediction. Learning/training is performed for the neural networks for a set of trajectories in advance. Then the feedforward compensation torque for any trajectory in the set can be calculated according to the predicted error from multiple neural networks managed with expert logic. Experimental study on a 6-DOF industrial robot has shown the superior performance of the proposed NN based learning scheme in the position tracking as well as the residual vibration reduction, without any further learning or end-effector sensors during operation after completion learning/training of motion trajectories in advance.

INTRODUCTION

End-effector performance in industrial robots suffers from the undesired discrepancy between the expected output and the actual system output, known as the *model following error*. Usually, the complete dynamics to define this discrepancy cannot be modeled accurately due to its complexity and uncertainty. Thus, it is hard to compensate it by standard model based feedforward control or model based adaptive control techniques.

If the robot is to execute repetitive tasks, and the robot repeatability is good, the error information from past iterations/periods can be utilized to reduce the error for the next iteration/period using the learning control techniques, such as iterative learning control (ILC) [1] and repetitive control [2]. For new trajectories, however, the learned knowledge cannot be directly applied and the system has to go through the learning process again to regain its performance, which is undesired in industrial applications.

On the other hand, if clear patterns appear in the error behavior, black box identification techniques can be applied to estimate the *model following error* for new trajectories based on the information from past learned trajectories. Then, learning control techniques can be applied to modify the feedforward compensation torque based on these predictions for new trajectories.

Several earlier works for this case have attempted to extend the learning knowledge to other varying motions using the techniques such as approximate fuzzy data model approach [3], neural network [4], adaptive fuzzy logic [5], and experience-based input selection [6]. Most of these algorithms are, however, either too complicated, or not suitable for highly nonlinear systems, and none of them have explored the multi-joint robot characteristics with joint elasticity or proven their performance in the actual robot setup.

In this paper, by studying the robot dynamics and error characteristics, we propose a learning control scheme using radial basis function neural network (NN) [7–9] based approach to learn and predict the *model following error*. Learning/training is performed for the neural networks prior to the prediction for real-time control stage. The prediction and control problem is prop-

*THIS WORK WAS SUPPORTED BY FANUC LTD., JAPAN, AND THE UPV PROMOE EXCHANGE PROGRAM, SPAIN

†Address all correspondence to this author.

erly decoupled into sub-problems for each individual joint to reduce the algorithm complexity and computation requirements. The performance of the proposed approach is experimentally evaluated and compared with the sensor based learning control, which requires learning for each new trajectory.

SYSTEM OVERVIEW

Consider an n-joint robot with gear compliance. The robot is equipped with motor side encoders for real-time feedback, and an end-effector sensor (e.g., accelerometer) for off-line use. Note that, if the computing resource and the sensor configuration allow, the end-effector sensor can also be used online. This paper, however, will address the conservative case where the end-effector sensor is for off-line and training use only, which is usually preferred in industry due to the cost saving and the limited real-time computational power.

Robot Dynamic Model

The dynamics of this robot can be formulated as [10]

$$\begin{aligned} &M_\ell(q_\ell)\ddot{q}_\ell + C(q_\ell,\dot{q}_\ell)\dot{q}_\ell + G(q_\ell) + D_\ell\dot{q}_\ell + F_{\ell c}\mathrm{sgn}(\dot{q}_\ell) \qquad (1)\\ &\quad + J^{\mathrm{T}}(q_\ell)f_{ext} = K_J\left(N^{-1}q_m - q_\ell\right) + D_J\left(N^{-1}\dot{q}_m - \dot{q}_\ell\right)\\ &M_m\ddot{q}_m + D_m\dot{q}_m + F_{mc}\mathrm{sgn}(\dot{q}_m) = \tau_m \qquad (2)\\ &\quad - N^{-1}\left[K_J\left(N^{-1}q_m - q_\ell\right) + D_J\left(N^{-1}\dot{q}_m - \dot{q}_\ell\right)\right] \end{aligned}$$

where $q_\ell, q_m \in \mathbb{R}^n$ are the load side and the motor side position vectors, respectively. $\tau_m \in \mathbb{R}^n$ is the motor torque vector. $M_\ell(q_\ell) \in \mathbb{R}^{n\times n}$ is the load side inertia matrix, $C(q_\ell, \dot{q}_\ell) \in \mathbb{R}^{n\times n}$ is the Coriolis and centrifugal force matrix, and $G(q_\ell) \in \mathbb{R}^n$ is the gravity vector. M_m, K_J, D_J, D_ℓ, D_m, $F_{\ell c}$, F_{mc}, and $N \in \mathbb{R}^{n\times n}$ are all diagonal matrices. The (i,i)-th elements of these matrices represent the motor side inertia, joint stiffness, joint damping, load side damping, motor side damping, load side Coulomb friction, motor side Coulomb friction, and gear ratio of the i-th joint, respectively. $f_{ext} \in \mathbb{R}^6$ denotes the external force acting on the robot due to contact with the environment. The matrix $J(q_\ell) \in \mathbb{R}^{6\times n}$ is the Jacobian matrix mapping from the load side joint space to the end-effector Cartesian space.

Define the nominal load side inertia matrix as $M_n = \mathrm{diag}(M_{n1}, M_{n2}, \cdots, M_{nn}) \in \mathbb{R}^{n\times n}$, where $M_{ni} = M_{\ell,ii}(q_{\ell 0})$, and $M_{\ell,ii}(q_{\ell 0})$ is the (i,i)-th element of the inertia matrix $M_\ell(q_{\ell 0})$ at the home (or nominal) position $q_{\ell 0}$. M_n can be used to approximate the inertia matrix $M_\ell(q_\ell)$. The off-diagonal entries of $M_\ell(q_\ell)$ represent the coupling inertia between the joints. Then, the robot dynamic model can be reformulated as follows

$$\begin{aligned} &M_m\ddot{q}_m + D_m\dot{q}_m = \tau_m - F_{mc}\mathrm{sgn}(\dot{q}_m) \qquad (3a)\\ &\quad - N^{-1}\left[K_J\left(N^{-1}q_m - q_\ell\right) + D_J\left(N^{-1}\dot{q}_m - \dot{q}_\ell\right)\right]\\ &M_n\ddot{q}_\ell + D_\ell\dot{q}_\ell = d^\ell(q) \qquad (3b)\\ &\quad + K_J\left(N^{-1}q_m - q_\ell\right) + D_J\left(N^{-1}\dot{q}_m - \dot{q}_\ell\right) \end{aligned}$$

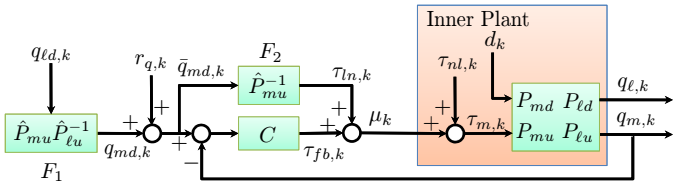


Figure 1. Robot control structure with reference & torque updates

where all the coupling and nonlinear terms, such as Coriolis force, gravity, Coulomb frictions, and external forces, are grouped into a fictitious disturbance torque $d^\ell(q) \in \mathbb{R}^n$ as

$$\begin{aligned} d^\ell(q) = &\left[M_nM_\ell^{-1}(q_\ell) - I_n\right]\left[K_J\left(N^{-1}q_m - q_\ell\right) + D_J\left(N^{-1}\dot{q}_m - \dot{q}_\ell\right)\right.\\ &\left. - D_\ell\dot{q}_\ell\right] - M_nM_\ell^{-1}(q_\ell)\left[C(q_\ell,\dot{q}_\ell)\dot{q}_\ell\right.\\ &\left. + G(q_\ell) + F_{\ell c}\mathrm{sgn}(\dot{q}_\ell) + J^{\mathrm{T}}(q_\ell)f_{ext}\right] \qquad (4) \end{aligned}$$

where $q = \left[q_m^{\mathrm{T}},\ q_\ell^{\mathrm{T}}\right]^{\mathrm{T}}$ and I_n is an $n \times n$ identity matrix.

Thus, the robot can be considered as a MIMO system with $2n$ inputs and $2n$ outputs as follows

$$q_m(j) = P_{mu}(z)\tau_m(j) + P_{md}(z)d(j) \qquad (5)$$
$$q_\ell(j) = P_{\ell u}(z)\tau_m(j) + P_{\ell d}(z)d(j) \qquad (6)$$

where j is the time index, z is the one step time advance operator in the discrete time domain, and the fictitious disturbance input $d(j)$ is defined as

$$d(j) = d(q(j)) = \left[-[F_{mc}\mathrm{sgn}(\dot{q}_m(j))]^{\mathrm{T}},\ [d^\ell(q(j))]^{\mathrm{T}}\right]^{\mathrm{T}} \qquad (7)$$

The transfer functions from the inputs to the outputs (i.e., $P_{mu}, P_{md}, P_{\ell u}$, and $P_{\ell d}$) can be derived from (3) as in [11].

Controller Structure for Iterative Learning Control

Note that the robot dynamic model (3) is in a decoupled form since all the variables are expressed in the diagonal matrix form or vector form. Therefore, the robot controller can be implemented in a decentralized form for each individual joint.

Figure 1 illustrates the control structure for this robot system, where the subscript k is the iteration index. It consists of two feedforward controllers, F_1 and F_2, and one feedback controller, C. Here, C can be any linear feedback controller such as a decoupled PID controller to stabilize the system. The feedforward controllers, F_1 and F_2, are designed using the nominal inverse model as

$$q_{md,k}(j) = \hat{P}_{mu}(z)\hat{P}_{\ell u}^{-1}(z)q_{\ell d,k}(j) \triangleq F_1(z)q_{\ell d,k}(j) \qquad (8)$$
$$\tau_{ln,k}(j) = \hat{P}_{mu}^{-1}(z)\left[q_{md,k}(j) + r_{q,k}(j)\right] \triangleq F_2(z)\bar{q}_{md,k}(j) \qquad (9)$$

where $\hat{\bullet}$ is the nominal model representation of $\bullet$, and $\bullet_d$ is the desired output of $\bullet$. $r_{q,k}$ and $\tau_{nl,k}$ are the additional reference and feedforward torque updates, respectively. The initialization/nominal values of these two updates are designed as

$$r_{q,0} = N\hat{K}_J^{-1}(\hat{\tau}_{\ell d} - \hat{M}_n\ddot{q}_{\ell d} - \hat{D}_\ell\dot{q}_{\ell d}) \tag{10}$$

$$\tau_{nl,0} = \tau_{ff,0} - \hat{\tau}_{ln,0} \tag{11}$$

where

$$\begin{aligned}\hat{\tau}_{\ell d} = {} & \hat{M}_\ell(q_{\ell d})\ddot{q}_{\ell d} + \hat{C}(q_{\ell d},\dot{q}_{\ell d})\dot{q}_{\ell d} + \hat{G}(q_{\ell d}) + \hat{D}_\ell\dot{q}_{\ell d} \\ & + \hat{F}_{\ell c}\mathrm{sgn}(\dot{q}_{\ell d}) + J^{\mathrm{T}}(q_{\ell d})\, f_{ext,d}\end{aligned} \tag{12}$$

$$\hat{\tau}_{md,0} = \hat{M}_m\bar{\ddot{q}}_{md,0} + \hat{D}_m\bar{\dot{q}}_{md,0} + \hat{F}_{mc}\mathrm{sgn}(\bar{\dot{q}}_{md,0}) \tag{13}$$

$$\tau_{ff,0} = \hat{\tau}_{md,0} + N^{-1}\hat{\tau}_{\ell d} \tag{14}$$

If the trajectory is repetitive, the two updates, $r_{q,k}$ and $\tau_{nl,k}$, can be generated iteratively by some ILC scheme such as the two-stage ILC algorithm designed in [11]. Particularly, the feedforward torque update $\tau_{nl,k}$ can be designed to compensate for the model following error of the inner plant (shaded area in Fig. 1) caused by the model uncertainty and the disturbance. This can be realized using the plant inversion learning scheme [11], i.e.

$$\tau_{nl,k+1}(j) = Q_u(z)[\tau_{nl,k}(j) + \hat{P}_{\ddot{\ell}u}^{-1}(z)e_{p\ddot{\ell},k}(j)] \tag{15}$$

where $P_{\ddot{\ell}u}$ is the transfer function from the motor torque τ_m to the load side joint acceleration $\ddot{q}_\ell$. The corresponding model following error is defined as $e_{p\ddot{\ell},k} = \hat{P}_{\ddot{\ell}u}\mu_k - \bar{\ddot{q}}_{\ell,k}$, where μ_k is the torque input to the inner plant and $\bar{\ddot{q}}_{\ell,k}$ is the load side joint acceleration estimated from the end-effector accelerometer measurements using the inverse differential kinematics techniques such as the one proposed in [12]. The low pass Q-filter $Q_u(z)$ is designed to trade off the performance bandwidth with the model uncertainties at high frequencies. The stability assurance for the ILC scheme and the monotonic convergence of $e_{p\ddot{\ell},k}$ are addressed in [11]. It is shown that this torque update scheme is effective in reducing the end-effector vibration, which is captured by the accelerometer.

Controller Structure with Neural Networks

If a new motion trajectory is desired, the prior ILC learning knowledge cannot be directly applied. Moreover, if the end-effector sensor is not available for executing the new task, the model following error $e_{p\ddot{\ell},k}$ cannot be obtained for new learning process. In this paper, we propose a neural network scheme to predict the model following error for a set of trajectories without further learning or end-effector sensor.

Figure 2 shows the control diagram with neural network (NN) predictor for the feedforward torque update, where F_{nl} denotes the nominal nonlinear feedforward controller designed in

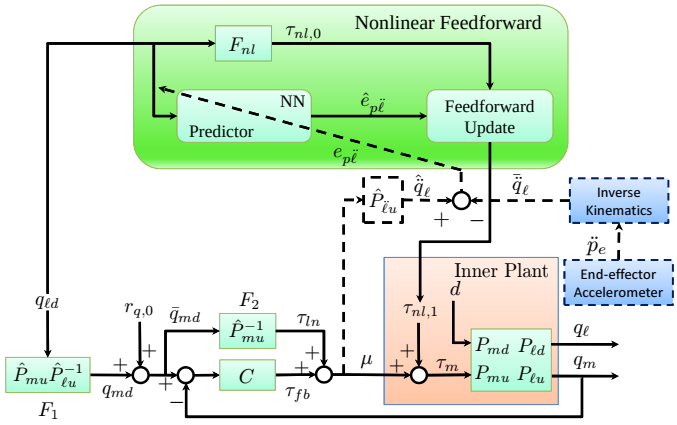


Figure 2. Control diagram with neural network predictor

(11). The dashed lines indicate the parts when the end-effector sensor is available and NN training can be conducted (e.g., in robot factory tuning/testing stage). The training of the neural networks will be detailed later. The other parts with solid lines indicate the nominal control structure at the operation stage, where the NN predictor provides the model following error estimate $\hat{e}_{p\ddot{\ell}}$ for each joint for any trajectory in the set. The feedforward torque for a new trajectory is then computed as

$$\tau_{nl,1}(j) = Q_u(z)[\tau_{n\ell,0}(j) + \hat{P}_{\ddot{\ell}u}^{-1}(z)\hat{e}_{p\ddot{\ell}}(j)] \tag{16}$$

Note that if the end-effector sensor is available, the ILC process (15) with newly measured/calculated error information can still continue after this initial run.

NEURAL NETWORK PREDICTOR

In this section a prediction system based on previously acquired training data is presented to estimate the joint acceleration model following error, which exhibits repeatable patterns under certain conditions in the robot. Figure 3 shows the predictor structure with all the parts detailed below.

Predictor Input Definition

The first step in this prediction problem is to choose the appropriate input signals that define the model following error. In this paper, we propose to define the predictor inputs as the trajectory references of either 2 dimensions (2D, velocity and acceleration) or 3 dimensions (3D, velocity, acceleration, and position). Moreover, due to the coupling dynamics on the multi-joint robot, a further study is carried out to identify the model following error as a proper combination of the trajectory references from all joints together, which is termed as the *movement cost* in this paper.

Proposition 1. *The model following error $e_{p\ddot{\ell},0}$ when applying*

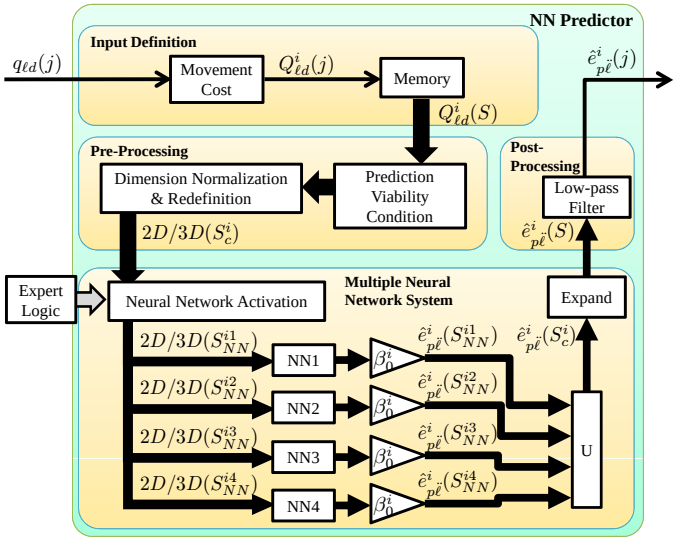


Figure 3. Neural network predictor structure

nominal feedforward torque $\tau_{nl,0}$ is a function of the joint trajectory reference if the robot dynamics is repeatable.

Proof. According to the control structure detailed in Fig. 1, the model following error $e_{p\ddot{\ell},0}$ can be derived as

$$e_{p\ddot{\ell},0} = -\Delta P_{\ddot{\ell}u} S_p (C + \hat{P}_{mu}^{-1})(\hat{P}_{mu}\hat{P}_{\ell u}^{-1} q_{\ell d} + r_{q,0}) - T_u S_p \tau_{nl,0} + (\Delta P_{\ddot{\ell}u} S_p C P_{md} - P_{\ddot{\ell}d}) d_0 \quad (17)$$

where $S_p = (I_n + CP_{mu})^{-1}$ is the sensitivity function of the closed loop system, $T_u = \hat{P}_{\ddot{\ell}u} C P_{mu} + P_{\ddot{\ell}u}$, and $\Delta P_{\ddot{\ell}u} = P_{\ddot{\ell}u} - \hat{P}_{\ddot{\ell}u}$ ([11]).

From (8)-(14), $r_{q,0}$ and $\tau_{nl,0}$ are also the functions of $q_{\ell d}$. Moreover, if the robot dynamics is repeatable and the controller setting remains the same, d_0 will also be the function of $q_{\ell d}$. Thus, the model following error $e_{p\ddot{\ell},0}$ is a function of the joint reference $q_{\ell d}$. □

This proposition implies that during the identification of $e_{p\ddot{\ell},0}$ based on $q_{\ell d}$, the feedback/feedforward controller and the robot working environment should remain consistent for all the training trajectories as well as the future desired task trajectories.

Note that the robot dynamics is coupled among joints. Thus, due to $r_{q,0}, \tau_{nl,0}$, and d_0 in (17), the model following error for each joint depends on the reference trajectories of other joints as well as that particular joint. In order to implement the decentralized predictor for each joint, the NN predictor input for the i-th joint is designed as the *movement cost* position vector $Q^i_{\ell d}$, velocity vector $\dot{Q}^i_{\ell d}$, and acceleration vector $\ddot{Q}^i_{\ell d}$, which are defined as the linear combination of the reference trajectories across all joints, i.e.

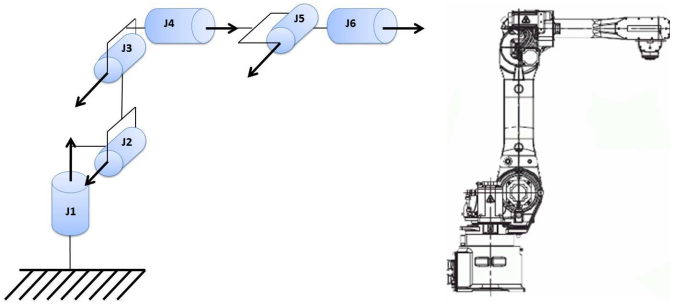
Table 1. Non-diagonal elements in matrix Φ

if $\vec{J}_A$ **and** $\vec{J}_B$ **are aligned**	$\vec{J}_A \equiv \vec{J}_B$	$\vec{J}_A \equiv -\vec{J}_B$
$\text{Dir\_iK}(\vec{J}_A) \equiv \text{Dir\_iK}(\vec{J}_B)$	1	-1
$\text{Dir\_iK}(\vec{J}_A) \equiv -\text{Dir\_iK}(\vec{J}_B)$	-1	1
if $\lvert\angle \vec{J}_A \vec{J}_B\rvert \equiv \frac{\pi}{2}$	0	

$\vec{J}_A$: Joint axis for which the movement cost is evaluated

$\vec{J}_B$: Joint axis the effect of which over $\vec{J}_A$ is assessed

Dir_iK : Rotation direction when applying inverse kinematics



(a) Axis Direction Convention (b) Robot Home Position

Figure 4. 6-DOF robot example

$$\underbrace{\begin{bmatrix} Q^1_{\ell d}(j) \\ Q^2_{\ell d}(j) \\ \vdots \\ Q^n_{\ell d}(j) \end{bmatrix}}_{Q_{\ell d}(j)} = \underbrace{\begin{bmatrix} 1 & a_{12} & \dots & a_{1n} \\ a_{21} & 1 & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & 1 \end{bmatrix}}_{\Phi} \underbrace{\begin{bmatrix} q^1_{\ell d}(j) \\ q^2_{\ell d}(j) \\ \vdots \\ q^n_{\ell d}(j) \end{bmatrix}}_{q_{\ell d}(j)} \quad (18)$$

$$\dot{Q}_{\ell d}(j) = \Phi \dot{q}_{\ell d}(j) \quad (19)$$

$$\ddot{Q}_{\ell d}(j) = \Phi \ddot{q}_{\ell d}(j) \quad (20)$$

where the design of Φ can be determined given a robot configuration. In this paper, we study the case of a 6-joint robot where the end-effector orientation is fixed as the home position shown in Fig. 4. The diagonal elements of the matrix Φ are set to 1 since the movement cost on each joint depends directly on its own movement. Non-diagonal elements are determined as shown in Tab. 1 according to the convention of axis direction shown in Fig. 4 and the desired direction of joint rotation to move the end-effector with fixed orientation.

Following this idea, the matrix Φ becomes

$$\Phi = \begin{bmatrix} 1 & 0 & 0\,1 & 0\,1 \\ 0 & 1 & -1\,0 & 1\,0 \\ 0 & -1 & 1\,0 & -1\,0 \\ 1 & 0 & 0\,1 & 0\,1 \\ 0 & 1 & -1\,0 & 1\,0 \\ 1 & 0 & 0\,1 & 0\,1 \end{bmatrix} \tag{21}$$

Finally, the inputs to the prediction system are defined as the velocity (19) and acceleration (20) movement costs for 2D neural networks, or position (18), velocity (19), and acceleration (20) movement costs for 3D neural networks. The dimension selection depends on the available training data and the computation power. It can be expected that a 3D network will generally provide more accurate prediction than a 2D network. However, more training data is required for a 3D network to perform effective learning. This also implies that more memory storage and computation capability are required for 3D networks.

Data Pre-processing

For the neural network system to be effective, it is crucial that there is a proper match between the data and neuron distribution in the input space. Hence, additional data pre-processing is developed to ensure that the training data covers all possible input values at which the neural network is intended to provide a prediction. This pre-processing stage consists of a magnitude normalization and variable redefinition of the input data, as well as a filtering of the output data, i.e., the predicted model following error. It is aimed to simplify the complexity of the function that defines the model following error from the movement costs, and to standardize the neural network learning for optimal performance.

Denote S as the set containing all the time steps, and T as the total number of time steps for the executing trajectory. The data pre-processing stage is to setup the input signals $\upsilon_{d,s}$ for the NN predictor f_{NN}, i.e.

$$\hat{e}_{p\ddot{\ell}}(j) = \begin{cases} f_{NN}(\upsilon_{1,1}(j), \upsilon_{2,2}(j), \upsilon_{3,3}(j)), & \text{3D NN (22a)} \\ f_{NN}(\upsilon_{2,2}(j), \upsilon_{3,3}(j)), & \text{2D NN (22b)} \end{cases}$$

where $j \in S = \{0, 1, 2, \cdots, T\}$, the subscript d of $\upsilon_{d,s}$ denotes the d-th dimension of the inputs, and s denotes the number of pre-processing steps as explained below applied to this input dimension.

Magnitude Normalization. First, for a given trajectory movement costs (i.e., $Q_{\ell d}$, $\dot{Q}_{\ell d}$, and $\ddot{Q}_{\ell d}$), and the model following error $e_{p\ddot{\ell}}$ to be learned by the neural network, a normalization

Table 2. Logic and boolean operator symbols

$\neg$	Logic Not	$\wedge$	Logic And	$\vee$	Logic Or
$\langle\bullet\rangle$	Boolean brackets with the output of $\bullet$ as 0 or 1				

is applied to each input variable for the i-th joint, i.e.

$$\beta_0^i = \max\left(e^i_{p\ddot{\ell}}(S)\right), \quad \upsilon^i_{1,1}(j) = \frac{Q^i_{\ell d}(j)}{\max\left(Q^i_{\ell d}(S)\right)},$$
$$\upsilon^i_{2,1}(j) = \frac{\dot{Q}^i_{\ell d}(j)}{\max\left(\dot{Q}^i_{\ell d}(S)\right)}, \quad \upsilon^i_{3,1}(j) = \frac{\ddot{Q}^i_{\ell d}(j)}{\max\left(\ddot{Q}^i_{\ell d}(S)\right)} \tag{23}$$

where $\max(\bullet)$ denotes the maximum absolute value across the time series $\bullet$.

Prediction Viability Prediction is only viable while the end-effector is moving, since it is based on velocity and acceleration references. The viability condition, χ^i_c, is defined as

$$\chi^i_c(j) = \left\langle \left|\upsilon^i_{2,1}(j)\right| > \varepsilon^i_{2,1} \right\rangle \ \vee \ \left\langle \left|\upsilon^i_{3,1}(j)\right| > \varepsilon^i_{3,1} \right\rangle \tag{24}$$
$$j^i_c \in S^i_c = \left\{ j : \chi^i_c(j) \right\} \subset S \tag{25}$$

where the logic and boolean operators are defined as in Tab. 2. $\varepsilon^i_{2,1}$ and $\varepsilon^i_{3,1}$ are small positive numbers to check if a number is close to zero. S^i_c, as a subset of S, encloses the time steps j^i_c that are eligible to be processed for the prediction at the i-th joint.

Redefinition on Acceleration Dimension In practice, reference trajectories are generated to ensure smooth motion. Thus acceleration and velocity pose a parabolic relationship on the plane where the horizontal and the vertical axes are velocity and acceleration respectively (see Fig. 7(a) for example). By studying the experimental error characteristics, we note that the model following error depends on the ratio between the velocity and the acceleration movement costs more significantly than on any of these two inputs separately. To utilize this pattern, the third dimension (acceleration movement cost) is redefined below.

1. Change the third dimension to the arctangent between the acceleration and the velocity movement costs, which gives the result in the range of $(-\frac{\pi}{2}, \frac{\pi}{2})$, i.e., $\upsilon^i_{3,2}(j^i_c) = \arctan\left(\dfrac{\upsilon^i_{3,1}(j^i_c)}{\upsilon^i_{2,1}(j^i_c)}\right)$.
2. Normalize the resulting third input dimension, i.e., $\upsilon^i_{3,3}(j^i_c) = \frac{2}{\pi}\upsilon^i_{3,2}(j^i_c)$.

Normalize Second Input Dimension To distribute data uniformly along all the input space, we need to normalize the second input dimension (velocity movement cost) at each

Table 3. Neural network activation rule

NN Type	Velocity	Acceleration	Motion Stage
1	Positive	Positive	Accelerating
2	Positive	Negative	Decelerating
3	Negative	Positive	Decelerating
4	Negative	Negative	Accelerating

sampled level of the third input dimension, i.e.

$$\upsilon^i_{2,2}(j^i_{c,l}) = \frac{\upsilon^i_{2,1}(j^i_{c,l})}{\max\left(\upsilon^i_{2,1}(S^i_{c,l})\right)} \tag{26}$$

$$\forall j^i_{c,l} \in S^i_{c,l} = \left\{ j^i_c : a_l \le \upsilon_{3,3}(j^i_c) < b_l \right\} \subset S^i_c \tag{27}$$

where l is the level number, a_l and b_l are the bounds of the third input dimension for the l-th level.

This concludes the data pre-processing and the final predictor is formulated as in (22a)-(22b). Note that for the time steps where the joint remains static, no joint prediction is available, i.e., $\hat{e}^i_{p\ell}(j^i_{nc}) = 0$, $\forall j^i_{nc} \notin S^i_c$.

Multiple Neural Network Activation

In order to enhance the prediction performance, the problem is divided into smaller prediction problems by means of multiple neural networks. Prior knowledge about the error behavior based on experience is formulated as several expert logic rules, whereby each network is specialized to a selected set of input data characterized by similar model following error behaviors under the movement cost definition. Here, each neural network is confined to a different motion stage according to the signs of the velocity and the acceleration movement costs as described in Tab. 3. In this way, the neural networks can learn all nonlinearities (e.g., Coulomb friction effect) more effectively in different motion stages.

By exploring the robot dynamics and error characteristics, it is noted that the $e_{p\ell}$ peaks normally appear when motion starts/stops where strong acceleration is imposed, or when the joint ends accelerating or starts decelerating where acceleration varies exponentially. In addition, an exception is studied where velocity remains almost constant for long periods. In this region, $e_{p\ell}$ tends to zero since the standard feedback and feedforward controller is normally designed to achieve satisfactory steady-state performance.

Define the pseudo-gradient and the pseudo-hessian for the third dimension input signal as

$$\nabla\left[\upsilon^i_{3,3}(j^i_c)\right] = \upsilon^i_{3,3}(j^i_c) - \upsilon^i_{3,3}(j^i_c - 1) \tag{28}$$

$$\nabla^2\left[\upsilon^i_{3,3}(j^i_c)\right] = \upsilon^i_{3,3}(j^i_c) - 2\,\upsilon^i_{3,3}(j^i_c - 1) + \upsilon^i_{3,3}(j^i_c - 2) \tag{29}$$

Then the boolean functions $\chi^i_{p,\bullet}$ are introduced to describe the following specific circumstances in the input data using the logic and boolean operators defined in Tab. 2:

- $\chi^i_{p,AZ}$: **Acceleration is constant & close to zero**, i.e., $\chi^i_{p,AZ}(j^i_c) = \left\langle \left| \upsilon^i_{3,3}(j^i_c) \right| \le \varepsilon^i_{3,3} \right\rangle \wedge \left\langle \left| \nabla\left[\upsilon^i_{3,3}(j^i_c)\right] \right| \le \varepsilon^i_{\nabla,3,3} \right\rangle$, where $\varepsilon^i_{3,3}$ and $\varepsilon^i_{\nabla,3,3}$ are small numbers.
- $\chi^i_{p,VC}$: **Velocity remains constant for a long period**, i.e., acceleration is close to zero for a long period, which can be obtained by analyzing $\chi^i_{p,AZ}$ via *Dilate* and *Erode* image processing methods [13].
- $\chi^i_{p,AS}$: **Acceleration changes sign**, i.e., acceleration is close to zero for a short period, $\chi^i_{p,AS}(j^i_c) = \left\langle \neg\chi^i_{p,VC}(j^i_c) \right\rangle \wedge \chi^i_{p,AZ}(j^i_c)$
- $\chi^i_{p,Vp}$: **Positive velocity**, i.e., $\chi^i_{p,Vp}(j^i_c) = \left\langle \upsilon^i_{2,2}(j^i_c) > 0 \right\rangle$.
- $\chi^i_{p,Vn}$: **Negative velocity**, i.e., $\chi^i_{p,Vn}(j^i_c) = \left\langle \upsilon^i_{2,2}(j^i_c) < 0 \right\rangle$.
- $\chi^i_{p,A}$: **Initial acceleration, or concave acceleration when velocity is close to be constant**, i.e., $\chi^i_{p,A}(j^i_c) = \left\langle \upsilon^i_{3,3}(j^i_c) \gg 0 \right\rangle \vee \left\langle \nabla^2\left[\upsilon^i_{3,3}(j^i_c)\right] < 0 \wedge \chi^i_{p,AZ}(j^i_c) \right\rangle$
- $\chi^i_{p,D}$: **Final deceleration, or convex acceleration when velocity is close to be constant**, i.e., $\chi^i_{p,D}(j^i_c) = \left\langle \upsilon^i_{3,3}(j^i_c) \ll 0 \right\rangle \vee \left\langle \nabla^2\left[\upsilon^i_{3,3}(j^i_c)\right] > 0 \wedge \chi^i_{p,AZ}(j^i_c) \right\rangle$

Then each of the four neural networks can be activated by the boolean function χ^{it}_{NN} defined by the following rule, where the superscript t is the type of neural network defined in Tab. 3.

$$\chi^{i1}_{NN}(j^i_c) = \left\langle \chi_{p,AS}(j^i_c) \vee \chi_{p,A}(j^i_c) \right\rangle \wedge \left\langle \neg\chi_{p,VC}(j^i_c) \right\rangle \wedge \chi^i_{p,Vp}(j^i_c)$$
$$\chi^{i2}_{NN}(j^i_c) = \left\langle \neg\chi_{p,AS}(j^i_c) \wedge \chi_{p,D}(j^i_c) \right\rangle \wedge \left\langle \neg\chi_{p,VC}(j^i_c) \right\rangle \wedge \chi^i_{p,Vp}(j^i_c)$$
$$\chi^{i3}_{NN}(j^i_c) = \left\langle \neg\chi_{p,AS}(j^i_c) \wedge \chi_{p,D}(j^i_c) \right\rangle \wedge \left\langle \neg\chi_{p,VC}(j^i_c) \right\rangle \wedge \chi^i_{p,Vn}(j^i_c)$$
$$\chi^{i4}_{NN}(j^i_c) = \left\langle \chi_{p,AS}(j^i_c) \vee \chi_{p,A}(j^i_c) \right\rangle \wedge \left\langle \neg\chi_{p,VC}(j^i_c) \right\rangle \wedge \chi^i_{p,Vn}(j^i_c)$$
$$j^{it}_{NN} \in S^{it}_{NN} = \left\{ j^i_c : \chi^{it}_{NN}(j^i_c) \right\} \subset S^i_c$$

The model following error prediction from the t-th neural network on the i-th joint for the time steps enclosed by S^{it}_{NN} is denoted as $\hat{e}_{p\ell}(S^{it}_{NN})$. Note that when velocity is constant for a long period, $\hat{e}_{p\ell}$ prediction is set to zero, i.e, $\hat{e}^i_{p\ell}(j^i_{n_{NN}}) = 0$, $\forall j^i_{n_{NN}} \notin \left\{ S^{i1}_{NN} \bigcup S^{i2}_{NN} \bigcup S^{i3}_{NN} \bigcup S^{i4}_{NN} \right\}$.

Radial Basis Function Neural Network

With the identified error patterns, the radial basis function neural networks [7, 8] can be applied to effectively learn the model following error. The success of prediction relies on the NN learning method utilized to ensure a stable learning process [9].

The neural networks utilized here are composed of two layers, and based on the radial basis function

(RBF) in (30). Define the t-th neural network for the i-th joint as $f^{it}_{NN,RBF}(\vec{x}^{\,i}(j^{it}_{NN}), \vec{\theta}^{\,it}, \mu^{it}, \sigma^{it})$, where $\vec{x}^{\,i}(j^{it}_{NN})$ is either $[\upsilon^i_{2,2}(j^{it}_{NN}), \upsilon^i_{3,3}(j^{it}_{NN})]^{\mathrm{T}}$ for 2D networks, or $[\upsilon^i_{1,1}(j^{it}_{NN}), \upsilon^i_{2,2}(j^{it}_{NN}), \upsilon^i_{3,3}(j^{it}_{NN})]^{\mathrm{T}}$ for 3D networks. Denote D as the number of the input dimensions and m_d as the number of neurons in the d-th dimension. Thus, the total number of RBF neurons at the input layer is $N_{RBF} = \prod_{d=1}^{D} m_d$. For the m-th neuron, the center position of the RBF and its width are preset and denoted respectively as $\vec{\mu}_m \in \mathbb{R}^D$ and $\sigma_m \in \mathbb{R}$. The neural network output, scaled by $\beta^i_0 \in \mathbb{R}$, is then defined in (31) as a product of the neuron regression vector, $\vec{\Gamma}^{it} \in \mathbb{R}^{N_{RBF}+1}$, and the parameter vector, $\vec{\theta}^{\,it} \in \mathbb{R}^{N_{RBF}+1}$, where the last entry θ^{it}_0 corresponds to the offset in the output prediction.

$$f_r(\vec{x}, \vec{\mu}, \sigma) = e^{-\frac{\|\vec{x}-\vec{\mu}\|_2^2}{\sigma^2}} \tag{30}$$

$$\hat{e}^i_{p\ell}(j^{it}_{NN}) = \beta^i_0 \vec{\theta}^{\,it} \vec{\Gamma}^{it}\left(\vec{x}^{\,i}(j^{it}_{NN}), \vec{\mu}^{\,it}, \sigma^{it}\right) \tag{31}$$

$$\vec{\theta}^{\,it} = \left[\, \theta^{it}_1, \cdots, \theta^{it}_{N_{RBF}}, \theta^{it}_0 \,\right] \tag{32}$$

$$\vec{\Gamma}^{it}(\vec{x}^{\,i}(j^{it}_{NN}), \vec{\mu}^{\,it}, \sigma^{it}) = \begin{bmatrix} f_r(\vec{x}^{\,i}(j^{it}_{NN}), \vec{\mu}^{\,it}_1, \sigma^{it}_1) \\ \vdots \\ f_r(\vec{x}^{\,i}(j^{it}_{NN}), \vec{\mu}^{\,it}_{N_{RBF}}, \sigma^{it}_{N_{RBF}}) \\ 1 \end{bmatrix} \tag{33}$$

The parameter vector $\vec{\theta}^{\,it}$ is tuned to minimize the following quadratic cost function, V^{it}_T, using the training data with the actual model following error collected by the end-effector sensor and the inverse differential kinematics method proposed in [12], as the dashed part in Fig. 2.

$$V^{it}_T = \frac{1}{2} \sum_{j^{it}_{NN} \in S^{it}_{NN}} \left(e^i_{NN}(j^{it}_{NN})\right)^2 \tag{34}$$

where $e^i_{NN}(j^{it}_{NN})$ is the prediction error given the current $\vec{\theta}^{\,it}$, i.e., $e^i_{NN}(j^{it}_{NN}) = e^i_{p\ell}(j^{it}_{NN}) - \hat{e}^i_{p\ell}(j^{it}_{NN})$. The optimized $\vec{\theta}^{\,it}$ for this least squares problem can be numerically obtained by gradient method with momentum [14–16] using heuristically adaptive step size and momentum gain.

Data Post-processing

A zero-phase low-pass filter is applied to smooth the final output of the NN predictor, which may contain discontinuities resulted from the output switching among neural networks and the unavailability of prediction during the static periods. The cut-off frequency for this low-pass filter is set to be higher than that of the Q-filter in (16) to ensure that the predicted information is rich enough for control update.

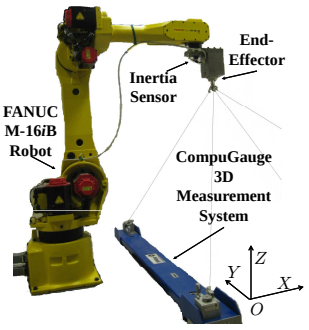


Figure 5. FANUC M-16iB robot system

DISCUSSION OF THE APPROACH

Stability & Safety Analysis

The assurance of stability for this neural network based learning control as well as several safety measures are taken into account during the design. On one hand, optimality and stability of the neural network training are ensured by utilizing radial basis function neurons in a double layer network with a quadratic cost function and momentum gradient method [15, 16]. On the other hand, the stability of plant inversion learning control (15)-(16) is assured as detailed in [11].

Furthermore, in order to increase prediction safety, data redundancy is utilized at the learning stage, and prediction uncertainty is also considered at the error prediction and feedforward correction stage. Thus, as the uncertainty grows, the prediction tends to zero, and no feedforward torque modification is applied.

Memory & Computation Requirements

The presented approach is suited for centralized systems where online equipment such as the data acquisition target and robot controller have very limited computation and storage resources but a computer with higher resources is available for off-line learning/training computation. In this way, one computer could be utilized for neural network training and learning control of several different robots for cost saving. If computation power and storage resources are quite limited (i.e., off-line PC is not available), some extra customization and simplification measures could be taken to facilitate the practical implementation.

EXPERIMENTAL STUDY

Test Setup

The proposed method is implemented on a 6-joint industrial robot, FANUC M-16iB/20, in Fig. 5. The robot is equipped with built-in motor encoders for each joint. An inertia sensor (Analog Devices, ADIS16400) consisting of a 3-axial accelerometer and a 3-axial gyroscope is attached to the end-effector. The three-dimensional position measurement system, CompuGauge 3D, is utilized to measure the end-effector tool center point (TCP) position as a ground truth for performance validation. The sampling

Table 4. Neuron distribution ranges for each neural network

NN Type	1*st* Dimension	2*nd* Dimension	3*rd* Dimension
1	$[-1,+1]$	$[0,+1]$	$[-0.1,+1]$
2	$[-1,+1]$	$[0,+1]$	$[-1,+0.1]$
3	$[-1,+1]$	$[-1,0]$	$[-1,+0.1]$
4	$[-1,+1]$	$[-1,0]$	$[-0.1,+1]$

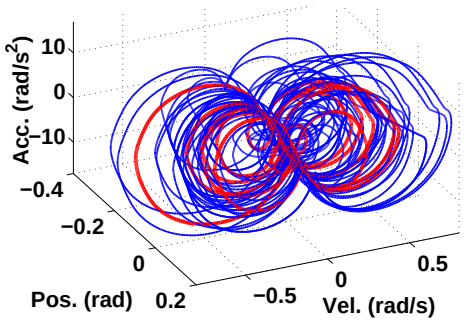


Figure 6. Training (blue) and validation (red) trajectories for Joint 3, based on a set of joint position, velocity, and acceleration references

rates of all the sensor signals as well as the real-time controller implemented through MATLAB xPC Target are set to 1kHz.

Algorithm Setup

For learning control algorithms (15)-(16), the zero-phase acausal low-pass filter Q_u is set with a cut-off frequency of 20Hz. The cut-off frequency for the NN output filtering is set to 50Hz.

In order to train the neural networks, experiments are performed to obtain the model following error variances for a set of different positions, velocities, and accelerations within certain workspace. In this paper, as a demonstration example, training and validation TCP trajectories are designed to move along X-axis for various distances of range of $60-100$cm, with fixed orientation but different varying velocities and accelerations. Only Joint 2, 3, and 5 need to move for these trajectories. Figure 6 shows the trajectories generated for Joint 3, where blue and red colors denote training and validation trajectories, respectively.

As described above, data is pre-processed for the neural network learning. Figure 7 shows the training data distribution before and after applying the movement cost definition and the pre-processing stage for both 2D and 3D networks. Thereafter, four neural networks are trained for each moving joint, with 20 rows of neurons uniformly distributed in the first dimension (only for 3D NN), 10 and 11 rows of neurons for the second and third dimensions respectively for each neural network as a trade-off between the performance and the computational viability. The neuron distribution ranges (Tab. 4) are set to be equal or larger than the expected input range. The RBF width σ for each neuron is set to 0.07 to ensure overlap between neurons.

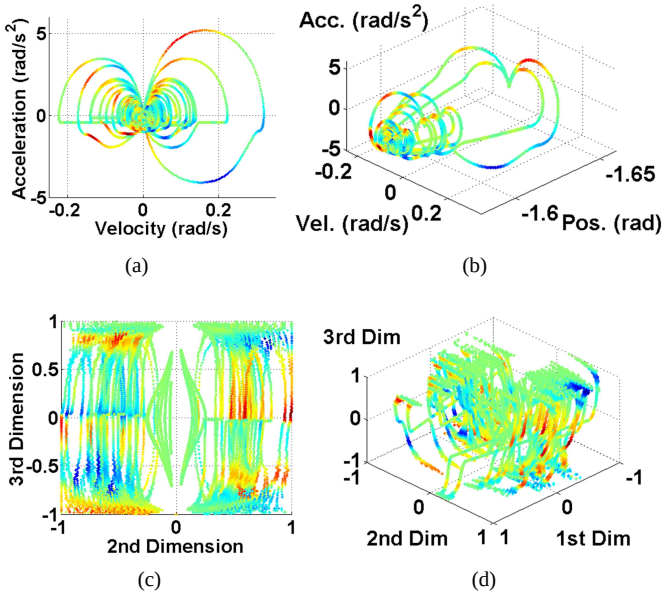


Figure 7. Joint 5 model following error. Color: $Red = 4$, $Blue = -4$, $Green = 0$ in [rad/sec^2]. Figure (a) and (b) are 2D and 3D distributions, respectively, based on Joint 5 reference before pre-processing stage. Figure (c) and (d) are 2D and 3D distributions, respectively, based on Joint 5 movement cost, with regards to the references from all joints, after pre-processing stage.

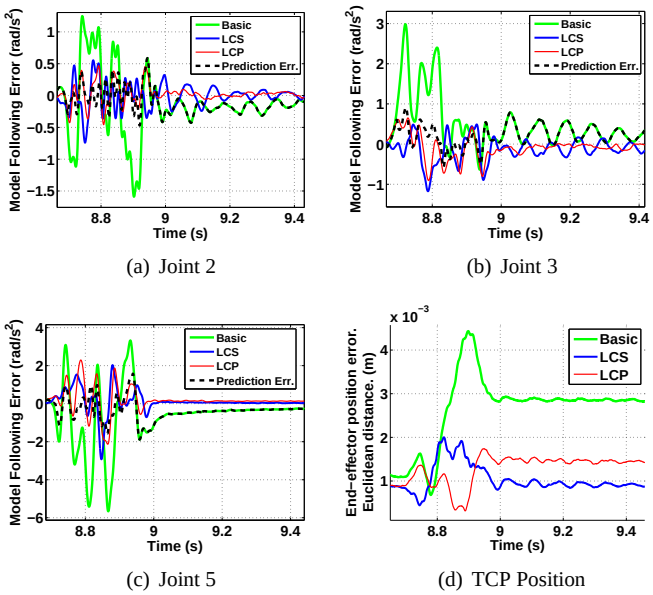
Experimental Results

The performance of the proposed learning control (16) (initial run) based on NN prediction (*LCP*) is compared with the basic controller (initial run in Fig. 1), and with the learning control (15) (second iteration) based on available end-effector sensor (*LCS*), by learning the feedforward control input for the validation trajectory (Fig. 6, red trajectory).

Since the learning control (16) aims at model matching for the inner plant during moving periods, results in these regions show that the *LCP* achieves about 94.5% (calculated by using root-mean-square (RMS) error values) of what the *LCS* achieves in reducing the model following error $e_{p\ddot{\ell}}$. Figure 8 shows a graphical comparison of the error reduction on Joint 2, 3, 5, and end-effector using the basic controller (*Basic*), *LCS*, *LCP*, and the prediction error e_{NN}. This result is confirmed with Tab. 5, which shows that the *LCP* achieves a substantial performance enhancement at the end-effector besides the model matching performance. Note that the *LCP* also improves the performance at the static period where the prediction is not viable. This static performance improvement is expected as a result of significant improvement at the dynamic period.

CONCLUSIONS

This paper investigated a learning control scheme based on model following error prediction, which suggested a viable solution in the industry for end-effector performance enhancement



(a) Joint 2 (b) Joint 3

(c) Joint 5 (d) TCP Position

Figure 8. Experimental performance comparisons for error reductions. Figure (a), (b), and (c) show the model following error on Joint 2, 3, and 5 respectively. Figure (d) shows the end-effector position error $\left(\text{in Euclidean distance, } \sqrt{e_X^2 + e_Y^2 + e_Z^2}\right)$

Table 5. Percentage of LCP performance compared with LCS. [%]

Period	Joint model following error	End-effector acceleration error	End-effector position error
Global	**84.6**	**66.3**	**56.2**
Dynamic	**94.5**	**74.3**	**70.0**
Static	**49.8**	**57.5**	**44.8**

when production line requires flexibility and efficiency. The proposed method improved feedforward torque compensation based on predicted error from multiple neural networks. The robot dynamics and error characteristics were explored and the neural network predictor was accordingly designed with novel input definition and data pre-processing stage. The radial basis function was utilized in the two-layer neural networks and the problem was further divided into four smaller neural networks for effective learning. Experimental study on a 6-DOF industrial robot showed a noticeable performance improvement of the end-effector over the basic controller for both dynamic and static periods without learning for a specific trajectory.

REFERENCES

[1] Bristow, D. A., and Tharayil, M., 2006. "A survey of iterative learning control: A learning-based method for high-performance tracking control". *IEEE Control Systems Magazine*(June), pp. 96–114.

[2] Cuiyan, L., and Dongchun, Z., 2004. "A survey of repetitive control". *Intelligent Robots and Systems (IROS), 2004 IEEE/RSJ International Conference on*, pp. 1160–1166.

[3] Gopinath, S., Kar, I., and Bhatt, R., 2008. "Experience inclusion in iterative learning controllers: Fuzzy model based approaches". *Engineering Applications of Artificial Intelligence*, **21**(4), June, pp. 578–590.

[4] Arif, M., Ishihara, T., and Inooka, H., 2002. "Generalization of iterative learning control for multiple desired trajectories in robotic systems". *PRICAI 2002: Trends in Artificial Intelligence*, **2417**, pp. 29–38.

[5] Chien, C.-j., 2008. "A Combined Adaptive Law for Fuzzy Iterative Learning Control of Nonlinear Systems With Varying Control Tasks". *IEEE Transactions on Fuzzy Systems*, **16**(1), Feb., pp. 40–51.

[6] Freeman, C. T., Alsubaie, M. A., Cai, Z., Rogers, E., and Lewin, P. L., 2011. "Initial Input Selection for Iterative Learning Control". *ASME Journal of Dynamic Systems, Measurement, and Control*, **133**, September.

[7] Moody, J., and Darken, C., 1989. *Fast learning in networks of locally-tuned processing units*. Neural Computation, 1:281-294.

[8] Poggio, T., and Girosi, F., 1990. *Networks for approximation and learning*. in Proceedings of the IEEE, volume 78, pages 1481-1497.

[9] Viñuela, P. I., and Galván, I. M., 2004. *Redes de Neuronas Artificiales. Un Enfoque Práctico*. Pearson Education.

[10] Wang, C.-C., 2008. "Motion Control of Indirect-drive Robots: Model Based Controller Design and Performance Enhancement Based on Load-side Sensors". PhD thesis, University of California at Berkeley.

[11] Chen, W., and Tomizuka, M., 2012. "Iterative Learing Control with Sensor Fusion for Robots with Mismatched Dynamics and Mismatched Sensing". In Proceedings of the ASME 2012 Dynamic Systems and Control Conference.

[12] Chen, W., and Tomizuka, M., 2012. "Load Side State Estimation in Elastic Robots with End-effector Sensing". In Proceedings of the 2012 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM).

[13] Bovik, A. C., 2009. *The Essential Guide to Image Processing*. Elsevier.

[14] Rumelhart, D., Hintont, G., and Williams, R., 1986. "Learning representations by back-propagating errors". *Nature*, **323**(6088), pp. 533–536.

[15] Torii, M., and Hagan, M., 2002. "Stability of steepest descent with momentum for quadratic functions". *Neural Networks, IEEE Transactions on*, **13**(3), pp. 752–756.

[16] Asensio, J., 2012. *UCB-FANUC Project Report - Predicted Acceleration Estimation Error Via Neural Networks For Robot End-effector Performance Enhancement*. Department of Mechanical Engineering, University of California, Berkeley.