**Jonathan Asensio**
Department of Systems Engineering & Control,
Polytechnic University of Valencia,
Valencia 46022, Spain
e-mail: jonathan.asensio@gmail.com

**Wenjie Chen**[2]
Department of Mechanical Engineering,
University of California,
Berkeley, CA 94720
e-mail: wjchen@berkeley.edu

**Masayoshi Tomizuka**
Department of Mechanical Engineering,
University of California,
Berkeley, CA 94720
e-mail: tomizuka@me.berkeley.edu

# Feedforward Input Generation Based on Neural Network Prediction in Multi-Joint Robots[1]

*Learning feedforward control based on the available dynamic/kinematic system model and sensor information is generally effective for reducing the tracking error for 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 feedforward input generation scheme based on neural network (NN) prediction. Learning/training is performed for the NNs for a set of trajectories in advance. Then the feedforward torque input for any trajectory in the predefined workspace can be calculated according to the predicted error from multiple NNs managed with expert logic. Experimental study on a 6-DOF industrial robot has shown the superior performance of the proposed NN based feedforward control scheme in the position tracking as well as the residual vibration reduction, without any further learning or end-effector sensors during operation.* [DOI: 10.1115/1.4025986]

## 1 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 for 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) [2] and repetitive control [3]. In the way of ILC, the learning controller learns the feedforward control input for a particular trajectory. 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* before initial run for new trajectories based on the information from past learned trajectories. These predictions can be then utilized to modify the feedforward compensation torque 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 [4], neural network [5], adaptive fuzzy logic [6], and experience-based input selection [7]. 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 feedforward input generation scheme using radial basis function NN [8–10] to learn and predict the *model following error*. Learning/training is performed for the NNs prior to the prediction for real-time control stage. The prediction and control problem is properly decoupled into subproblems 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 additional learning and sensing for each new trajectory.

## 2 System Overview

Consider an $n$-joint robot with gear compliance, which is commonly used in industrial applications. The robot is equipped with motor side encoders for real-time feedback, and an end-effector sensor (e.g., accelerometer) for off-line training use[3].

**2.1 Robot Dynamic Model.** The dynamics [12] of this robot can be formulated as[4]

$$M_\ell(q_\ell)\ddot{q}_\ell + C(q_\ell,\dot{q}_\ell)\dot{q}_\ell + G(q_\ell) + D_\ell\dot{q}_\ell + F_{\ell c}\mathrm{sgn}(\dot{q}_\ell) + J^{\mathrm{T}}(q_\ell)f_{ext} = K_J\left(N^{-1}q_m - q_\ell\right) + D_J\left(N^{-1}\dot{q}_m - \dot{q}_\ell\right) \quad (1)$$

$$M_m\ddot{q}_m + D_m\dot{q}_m + F_{mc}\mathrm{sgn}(\dot{q}_m) = u - N^{-1}\left[K_J\left(N^{-1}q_m - q_\ell\right) + D_J\left(N^{-1}\dot{q}_m - \dot{q}_\ell\right)\right] \quad (2)$$

where $q_\ell, q_m \in \mathbb{R}^n$ are the load side and the motor side position vectors, respectively. $u \in \mathbb{R}^n$ 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

[1]This work was supported by FANUC Corporation, Japan, and the UPV PROMOE exchange program, Spain. This paper was presented in part as the paper (DSC2012-8726) [1] in the 2012 ASME Dynamic System Control Conference.

[2]Corresponding author.

Contributed by the Dynamic Systems Division of ASME for publication in the Journal of Dynamic Systems, Measurement, and Control. Manuscript received November 11, 2012; final manuscript received November 2, 2013; published online January 20, 2014. Assoc. Editor: Won-jong Kim.

[3]If the computing resource and the sensor configuration allow, the end-effector sensor can also be used online [11]. This paper, however, will address the conservative case where the end-effector sensor is for off-line training use only, which is preferred in industry due to the cost saving and the limited real-time computational power.

[4]The dynamic model and the controller introduced later will be greatly simplified if the joint compliance is absent (i.e., $K_J = \infty, D_J = \infty$, and $q_\ell = q_m/N$). The proposed controller may not be necessary in this special case since the problems (such as mismatched dynamics and the residual vibration behavior) arising from the joint compliance may not exist.

the Coriolis and centrifugal force matrix, and $G(q_\ell) \in \mathbb{R}^n$ is the gravity vector. $M_m$, $K_J$, $D_J$, $D_\ell$, $D_m$, $F_{\ell c}$, $F_{mc}$, 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 $M_n = \text{diag}(M_{n1}, M_{n2}, \ldots, M_{nn}) \in \mathbb{R}^{n\times n}$, 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

$$M_m\ddot{q}_m + D_m\dot{q}_m = u - F_{mc}\text{sgn}(\dot{q}_m) - N^{-1}\left[K_J\left(N^{-1}q_m - q_\ell\right) + D_J\left(N^{-1}\dot{q}_m - \dot{q}_\ell\right)\right] \tag{3}$$

$$M_n\ddot{q}_\ell + D_\ell\dot{q}_\ell = d^\ell(q) + K_J\left(N^{-1}q_m - q_\ell\right) + D_J\left(N^{-1}\dot{q}_m - \dot{q}_\ell\right) \tag{4}$$

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

$$d^\ell(q) = \left[M_nM_\ell^{-1}(q_\ell) - I_n\right]\left[K_J\left(N^{-1}q_m - q_\ell\right) + D_J\left(N^{-1}\dot{q}_m - \dot{q}_\ell\right) - D_\ell\dot{q}_\ell\right] - M_nM_\ell^{-1}(q_\ell)\left[C(q_\ell,\dot{q}_\ell)\dot{q}_\ell + G(q_\ell) + F_{\ell c}\text{sgn}(\dot{q}_\ell) + J^{\text{T}}(q_\ell)f_{ext}\right] \tag{5}$$

where $q = \left[q_m^{\text{T}}, \quad q_\ell^{\text{T}}\right]^{\text{T}}$ 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

$$q_m(j) = P_{mu}(z)u(j) + P_{md}(z)d(j) \tag{6}$$

$$q_\ell(j) = P_{\ell u}(z)u(j) + P_{\ell d}(z)d(j) \tag{7}$$

where $j$ is the time index, $z$ is the one step time advance operator in the discrete time domain, and the fictitious disturbance input is defined as $d(j) = \left[d^1(j), \quad \ldots, \quad d^n(j)\right]^{\text{T}}$ and

$$d^i(j) = d^i(q(j)) = \left[-F_{mc}^i\text{sgn}(\dot{q}_m^i(j)), \quad d^{\ell i}(q(j))\right] \tag{8}$$

where the superscript $i$ denotes the $i$-th joint component of the variable. As shown later, we will use feedforward torque to compensate for this fictitious disturbance input $d$. 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 Eqs. (3) and (4) as in Ref. [13].

**2.2 Controller Structure for Iterative Learning Control.** The robot dynamic model (3) and (4) 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 a control structure for this robot system to track the load side joint reference trajectory $q_{\ell r}$. The proposed control structure has two nested loops. The inner loop (i.e., feedback controller $C$ and feedforward torque signal consisting of the linear component $\tau_{ln}$ from $F_2$ and the nonlinear component $\tau_{nl}$) uses the motor torque $u$ as the control to the robot plant to achieve motor side tracking, which is the common practice in industrial applications. The joint flexibility due to the gear transmissions is addressed in the outer loop (i.e., feedforward

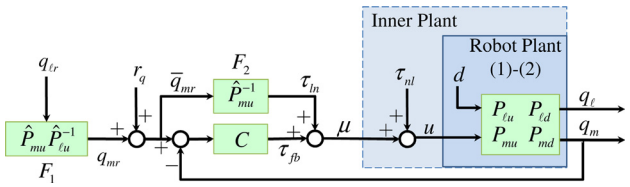



**Fig. 1 Robot control structure with reference and torque update. The ultimate objective is to make robot plant output $q_\ell$ track the load side reference trajectory $q_{\ell r}$. Only motor side position output $q_m$ is available for the real-time feedback. $r_q$ and $\tau_{nl}$ are the additional reference and feedforward torque updates to further compensate for the joint flexibility and the fictitious disturbance $d$, respectively. See Sec. 2.2 for more controller details.**

reference controller $F_1$ and feedforward reference signal $r_q$), which uses the motor side reference $\bar{q}_{mr}$ as the control to the inner loop to achieve the ultimate objective, i.e., load side tracking. 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 model inverse as

$$q_{mr}(j) = \hat{P}_{mu}(z)\hat{P}_{\ell u}^{-1}(z)q_{\ell r}(j) \triangleq F_1(z)q_{\ell r}(j) \tag{9}$$

$$\tau_{ln}(j) = \hat{P}_{mu}^{-1}(z)\left[q_{mr}(j) + r_{q,k}(j)\right] \triangleq F_2(z)\bar{q}_{mr}(j) \tag{10}$$

where $\hat{\bullet}$ is the nominal model representation of $\bullet$. $r_q$ and $\tau_{nl}$ are the additional reference and feedforward torque updates, respectively, with the initial/nominal values designed as

$$r_{q,0} = N\hat{K}_J^{-1}(\hat{\tau}_{\ell r} - \hat{M}_n\ddot{q}_{\ell r} - \hat{D}_\ell\dot{q}_{\ell r}) \tag{11}$$

$$\tau_{nl,0} = \tau_{ff,0} - \tau_{ln,0} \tag{12}$$

where the subscript 0 denotes the calculation prior to the first experiment iteration, and

$$\tau_{ff,0} = \hat{\tau}_{mr,0} + N^{-1}\hat{\tau}_{\ell r} \tag{13}$$

$$\hat{\tau}_{\ell r} = \hat{M}_\ell(q_{\ell r})\ddot{q}_{\ell r} + \hat{C}(q_{\ell r},\dot{q}_{\ell r})\dot{q}_{\ell r} + \hat{G}(q_{\ell r}) + \hat{D}_\ell\dot{q}_{\ell r} + \hat{F}_{\ell c}\text{sgn}(\dot{q}_{\ell r}) + J^{\text{T}}(q_{\ell r})f_{ext,r} \tag{14}$$

$$\hat{\tau}_{mr,0} = \hat{M}_m\ddot{\bar{q}}_{mr,0} + \hat{D}_m\dot{\bar{q}}_{mr,0} + \hat{F}_{mc}\text{sgn}(\dot{\bar{q}}_{mr,0}) \tag{15}$$

*Remark 1*. Note that (13)–(15) are obtained from Eqs. (1) and (2) using the nominal model representations. With this nominal torque calculation, $r_{q,0}$ in Eq. (11) is aimed to account for the joint torsion due to the nominal nonlinear torque at the load side.

Equation (12) and Fig. 1 show that the overall nominal feedforward torque $\tau_{ff,0}$ is divided into two parts. The nonlinear part $\tau_{nl,0}$ is injected inside the inner plant to compensate for the fictitious disturbance d and to make the inner plant behave as the chosen nominal linear model. With this in mind, the linear feedforward torque $\tau_{ln,0}$, which is computed by Eq. (10) using the nominal linear model, can be injected outside the inner plant to achieve the nominal performance. The model uncertainty and inaccuracy resulted from the use of the nominal model are addressed by the following schemes to further refine the feedforward torque update $\tau_{nl}$.

If the trajectory is repetitive, the two updates, $r_{q,k}$ and $\tau_{nl,k}$, where the subscript $k$ is the experiment iteration index, can be generated iteratively by some ILC scheme such as the two-stage ILC algorithm designed in Ref. [13]. Particularly, the feedforward torque update $\tau_{nl,k}$ is designed to compensate for the model following error of the inner plant (shaded area with dashed outline in Fig. 1) caused by the model uncertainty and the fictitious

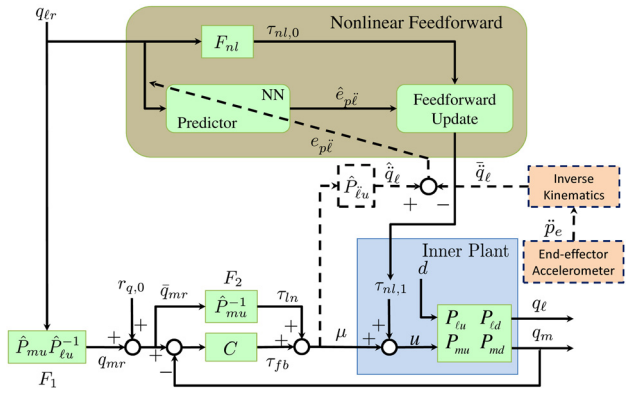



**Fig. 2 Control diagram with NN predictor**

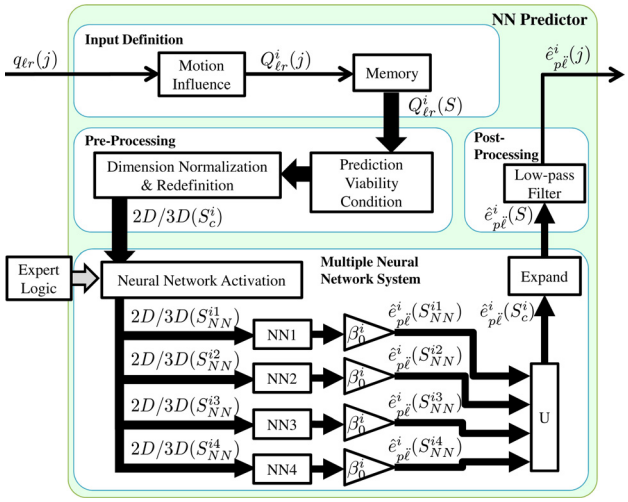



**Fig. 3 NN predictor structure**

disturbance $d$. This can be realized using the plant inversion learning scheme [13], i.e.,

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

where $P_{\ddot{\ell}u}$ is the transfer function from the motor torque $u$ to the load side joint acceleration $\ddot{q}_\ell$. The corresponding model following error is defined as $e_{p\ddot{\ell},k} = \hat{P}_{\ddot{\ell}u}\mu_k - \ddot{q}_{\ell,k}$, where $\mu_k$ is the torque input to the inner plant. Due to the unavailability of the load side joint acceleration measurement $\ddot{q}_{\ell,k}$, we utilize the estimate $\bar{\ddot{q}}_{\ell,k}$ from the end-effector accelerometer measurements using the inverse differential kinematics technique proposed in Ref. [11]. The low pass $Q$-filter $Q_u(z)$ is designed to trade off the performance bandwidth with the model uncertainty at high frequencies. It was shown in [13] that this torque update scheme is effective in reducing the end-effector vibration captured by the accelerometer.

**2.3 Controller Structure With NNs.** 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 NN scheme to predict the model following error at the first iteration for a set of trajectories without further learning or end-effector sensor. By doing this, the unmodeled dynamics (e.g., uncertainty and disturbances) can be approximated in an indirect way by the NNs of model following error, which later can be used to refine the feedforward torque to compensate for such unmodeled dynamics.

Figure 2 shows the control diagram with NN predictor for the feedforward torque update, where $F_{nl}$ denotes the nominal nonlinear feedforward controller designed in Eq. (12). 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 NNs 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 trained workspace. The feedforward torque for a new trajectory is then computed based on such NN prediction as

$$\tau_{nl,1}(j) = Q_u(z)[\tau_{n\ell,0}(j) + \hat{P}_{\ddot{\ell}u}^{-1}(z)\hat{e}_{p\ddot{\ell}}(j)] \tag{17}$$

Note that the prediction $\hat{e}_{p\ddot{\ell}}$ and the updated feedforward torque $\tau_{nl}$ remain the same for all the iterations after the initial run. However, if the end-effector sensor is available, the ILC process (16) with newly measured/calculated error can still continue after this initial run.

*Remark 2*. In this paper, we focus on the generation of the nonlinear feedforward torque $\tau_{nl}$. This feedforward torque update aims at reducing the model following error $e_{p\ddot{\ell}}$ of the inner plant instead of the tracking error $e = q_{\ell r} - q_\ell$ for the overall system. However, by making the inner plant behave as the nominal linear model, the tracking performance will be effectively improved, since the linear feedforward torque signal $\tau_{ln}$ injected outside the inner plant is computed by Eq. (10) using the nominal linear model.

*Remark 3*. A training stage involving certain effort (as shown later) is required for the proposed NN approach. However, this training is one time only and can be performed in the robot factory tuning/testing stage. After that no learning effort is required for any new trajectory within the trained workspace. This offers a significant advantage over the typical ILC schemes, which need to go through the learning process whenever the trajectory is changed.

## 3 NN 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 of the robot. Figure 3 shows the predictor structure with all the parts detailed below.

**3.1 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, position, velocity, and acceleration). Moreover, due to the coupling dynamics (i.e., the grouped disturbance term $d^\ell(q)$) of the multi-joint robot, we identify the model following error as a function output from a proper combination of the trajectory references from all joints together, which is termed as the *motion influence* in this paper.

PROPOSITION 1. *The model following error $e_{p\ddot{\ell},0}$ is a function of joint trajectory references, if robot dynamics is repeatable and the nominal feedforward inputs $\tau_{nl,0}$ and $r_{q,0}$ are applied.*

*Proof*. By taking the load side joint acceleration $\ddot{q}_\ell$ as the output, Eq. (7) can be rewritten as

$$\ddot{q}_\ell(j) = P_{\ddot{\ell}u}(z)u(j) + P_{\ddot{\ell}d}(z)d(j) \tag{18}$$

where $P_{\ddot{\ell}u}$ and $P_{\ddot{\ell}d}$ are the transfer functions from the motor torque $u$ and the disturbance $d$ to the load side joint acceleration $\ddot{q}_\ell$, respectively.

According to the control structure in Fig. 1, we denote $S_p = (I_n + CP_{mu})^{-1}$ as the sensitivity function of the closed loop

system, where $I_n$ is an $n \times n$ identity matrix. Then the load side acceleration output $\ddot{q}_\ell$ can be derived as

$$\begin{aligned}\ddot{q}_\ell &= P_{\ddot{\ell}u}u + P_{\ddot{\ell}d}d \\ &= P_{\ddot{\ell}u}S_p\left[(C+\hat{P}_{mu}^{-1})(q_{mr}+r_q)+\tau_{nl}-CP_{md}d\right]+P_{\ddot{\ell}d}d \\ &= \hat{P}_{mu}^{-1}P_{\ddot{\ell}u}S_p\hat{S}_p^{-1}r_q + P_{\ddot{\ell}u}S_p \\ &\quad\cdot\left(\hat{P}_{\ell u}^{-1}\hat{S}_p^{-1}q_{\ell r}+\tau_{nl}-CP_{md}d\right)+P_{\ddot{\ell}d}d\end{aligned} \tag{19}$$

where we have noted that $C$, $P_{mu}$, $P_{\ddot{\ell}u}$, and their nominal models ($\hat{P}_{mu}$ and $\hat{P}_{\ddot{\ell}u}$) are diagonal due to our decoupled dynamic modeling (3) and (4).

The input–output equation of the nominal inner plant can be derived as

$$\begin{aligned}\ddot{q}_p \triangleq \hat{P}_{\ddot{\ell}u}\mu = \hat{P}_{\ddot{\ell}u}S_p[(C+\hat{P}_{mu}^{-1})(\hat{P}_{mu}\hat{P}_{\ell u}^{-1}q_{\ell r}+r_q) \\ -CP_{mu}\tau_{nl}-CP_{md}d]\end{aligned} \tag{20}$$

Thus, the model following error $e_{p\ddot{\ell}}$ becomes

$$\begin{aligned}e_{p\ddot{\ell}} &= \ddot{q}_p - \ddot{q}_\ell \\ &= -T_uS_p\tau_{nl} - \Delta P_{\ddot{\ell}u}S_p(C+\hat{P}_{mu}^{-1})\cdot(\hat{P}_{mu}\hat{P}_{\ell u}^{-1}q_{\ell r}+r_q) \\ &\quad + (\Delta P_{\ddot{\ell}u}S_pCP_{md}-P_{\ddot{\ell}d})d\end{aligned} \tag{21}$$

where $T_u = \hat{P}_{\ddot{\ell}u}CP_{mu} + P_{\ddot{\ell}u}$, and $\Delta P_{\ddot{\ell}u} \triangleq P_{\ddot{\ell}u} - \hat{P}_{\ddot{\ell}u}$.

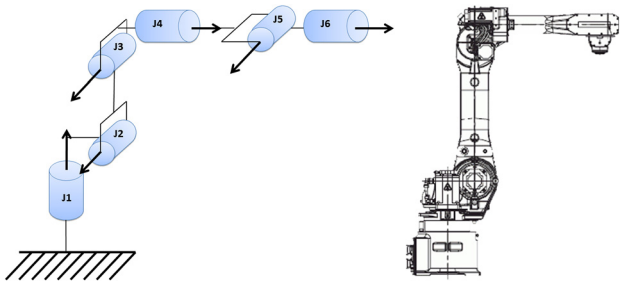
Therefore, $e_{p\ddot{\ell}}$ is a function of $q_{\ell r}, \tau_{nl}$, $r_q$, and $d$. Equations (9)–(13) show that the nominal feedforward inputs $r_{q,0}$ and $\tau_{nl,0}$ are also functions of $q_{\ell r}$. Moreover, if the robot dynamics is repeatable and the controller setting remains the same, $d$ will also be a function of $q_{\ell r}$. Thus, the model following error $e_{p\ddot{\ell},0}$ with nominal feedforward inputs is a function of the joint reference $q_{\ell r}$. □

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

Note that the robot dynamics is coupled among joints. Thus, due to $r_q, \tau_{nl}$, and $d$ in Eq. (21), the model following error for each joint depends on the reference trajectories of other joints besides that particular joint. In order to implement the decentralized predictor for each joint, the NN predictor input for the $i$-th joint is synthesized as the *motion influence* position vector $Q_{\ell r}^i$, velocity vector $\dot{Q}_{\ell r}^i$, and acceleration vector $\ddot{Q}_{\ell r}^i$, which are defined as the linear combination of the reference trajectories across all joints, i.e.,

$$\underbrace{\begin{bmatrix}Q_{\ell r}^1(j)\\ Q_{\ell r}^2(j)\\ \vdots \\ Q_{\ell r}^n(j)\end{bmatrix}}_{Q_{\ell r}(j)} = \underbrace{\begin{bmatrix}1 & \phi_{12} & \cdots & \phi_{1n}\\ \phi_{21} & 1 & \cdots & \phi_{2n}\\ \vdots & \vdots & \ddots & \vdots \\ \phi_{n1} & \phi_{n2} & \cdots & 1\end{bmatrix}}_{\Phi} \underbrace{\begin{bmatrix}q_{\ell r}^1(j)\\ q_{\ell r}^2(j)\\ \vdots \\ q_{\ell r}^n(j)\end{bmatrix}}_{q_{\ell r}(j)} \tag{22}$$

$$\dot{Q}_{\ell r}(j) = \Phi\dot{q}_{\ell r}(j) \tag{23}$$




(a) (b)

**Fig. 4 6-DOF robot example (a) axis direction convention and (b) robot home position**

$$\ddot{Q}_{\ell r}(j) = \Phi\ddot{q}_{\ell r}(j) \tag{24}$$

where $\Phi$ can be determined given a robot configuration. The entry $\phi_{i,j}$ of $\Phi$ indicates the influence factor of the $j$-th joint motion on the $i$-th joint performance (e.g., model following error). This influence factor $\phi_{i,j}$ can be approximately determined by the kinematic relationship (see the example later) between the joints and then normalized to be within $[-1, 1]$. The resulting $\Phi$ may not be necessarily symmetric due to different dynamics of each joint. Here, for simplicity, we assume that the influences between the two joints are mutually equivalent and thus $\Phi$ could be symmetric. 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 $\Phi$ are set to 1 since the motion influence on each joint depends directly on its own movement. Nondiagonal elements are determined by the rules in Table 1 according to the convention of axis direction shown in Fig. 4 and the desired joint rotation direction to move the end-effector with fixed orientation.

Following this idea, the matrix $\Phi$ becomes

$$\Phi = \begin{bmatrix}1 & 0 & 0 & 1 & 0 & 1\\ 0 & 1 & -1 & 0 & 1 & 0\\ 0 & -1 & 1 & 0 & -1 & 0\\ 1 & 0 & 0 & 1 & 0 & 1\\ 0 & 1 & -1 & 0 & 1 & 0\\ 1 & 0 & 0 & 1 & 0 & 1\end{bmatrix} \tag{25}$$

Finally, the inputs to the prediction system are defined as velocity (23) and acceleration (24) motion influences for 2D NNs, or position (22), velocity (23), and acceleration (24) motion influences for 3D NNs. 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 (also more memory storage and computation capability) will be required for a 3D network to perform effective learning.

**3.2 Data Preprocessing.** For the NN 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

**Table 1 Nondiagonal 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 | $\vec{J}_A$: Joint axis for which the motion influence is evaluated |
| $\text{Dir\_iK}(\vec{J}_A) \equiv -\text{Dir\_iK}(\vec{J}_B)$ | −1 | 1 | $\vec{J}_B$: Joint axis the effect of which over $\vec{J}_A$ is assessed<br>Dir_iK: Rotation direction when applying inverse kinematics |
| if $\lvert\angle\vec{J}_A\vec{J}_B\rvert \equiv \pi/2$ | 0 | | |

preprocessing is developed to ensure that the training data covers all possible input values at which the NN is intended to provide a prediction. This preprocessing stage consists of a magnitude normalization and variable redefinition of the input data. It is aimed to simplify the complexity of the function that defines the model following error from the motion influences and to standardize the NN learning for optimal performance.

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

$$\hat{e}_{p\ddot{\ell}}(j) = \begin{cases} f_{NN}(\upsilon_{1,1}(j), \upsilon_{2,2}(j), \upsilon_{3,3}(j)), & \text{3D} \quad (26a) \\ f_{NN}(\upsilon_{2,2}(j), \upsilon_{3,3}(j)), & \text{2D} \quad (26b) \end{cases}$$

where $j \in S = \{0, 1, 2, \cdots, T\}$, the subscript $d$ of $\upsilon_{d,s}$ denotes the $d$-th dimension (i.e., $d=1$ for position, $d=2$ for velocity, and $d=3$ for acceleration) of the inputs, and $s$ denotes the number of following preprocessing steps applied to this input dimension.

*3.2.1 Magnitude Normalization.* First, for a given trajectory motion influences (i.e., $Q_{\ell r}, \dot{Q}_{\ell r}$, and $\ddot{Q}_{\ell r}$) and the model following error $e_{p\ddot{\ell}}$ to be learned by the NN, a normalization is applied to each input variable for the $i$-th joint, i.e.,

$$\beta_0^i = \max_{k\in S}\left|e^i_{p\ddot{\ell}}(k)\right|, \quad \upsilon^i_{1,1}(j) = \frac{Q^i_{\ell r}(j)}{\max_{k\in S}\left|Q^i_{\ell r}(k)\right|},$$

$$\upsilon^i_{2,1}(j) = \frac{\dot{Q}^i_{\ell r}(j)}{\max_{k\in S}\left|\dot{Q}^i_{\ell r}(k)\right|}, \quad \upsilon^i_{3,1}(j) = \frac{\ddot{Q}^i_{\ell r}(j)}{\max_{k\in S}\left|\ddot{Q}^i_{\ell r}(k)\right|}$$

where the superscript $i$ indicates the $i$-th joint component.

*3.2.2 Prediction Viability.* Note that prediction is only viable while the end-effector is moving, since it is based on velocity and acceleration references. Thus, the viability condition, $\chi^i_c$, is defined as

$$\chi^i_c(j) = \left\langle \left|\upsilon^i_{2,1}(j)\right| > \in^i_{2,1} \right\rangle \vee \left\langle \left|\upsilon^i_{3,1}(j)\right| > \in^i_{3,1} \right\rangle \quad (27)$$

$$S^i_c = \left\{ j : \chi^i_c(j) \right\} \subset S \quad (28)$$

where the logic and Boolean operators are defined as in Table 2. $\in^i_{2,1}$ and $\in^i_{3,1}$ 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 that are eligible to be processed for the prediction at the $i$-th joint.

*3.2.3 Redefinition on Third Input Dimension.* In practice, reference trajectories are generated to ensure smooth motion. Therefore, the acceleration and the velocity pose a parabolic-like relationship on the velocity-acceleration plane (e.g., Fig. 7(*a*)). By studying the experimental error characteristics, we note that the model following error depends on the ratio between the velocity and the acceleration motion influences more significantly than on either individual input. To utilize this pattern, the third dimension (acceleration motion influence) is redefined below:

(1) Change the third dimension to the arctangent between the acceleration and the velocity motion influences, i.e., $\upsilon^i_{3,2}(j^i_c) = \arctan(\upsilon^i_{3,1}(j^i_c)/\upsilon^i_{2,1}(j^i_c)) \in (-\frac{\pi}{2}, \frac{\pi}{2})$. Notice that $\pm\frac{\pi}{2}$ corresponds to the motion periods of velocity reversal (zero-crossing). This will essentially group uncertainties/changes of friction force together during these velocity reversal periods.

(2) Normalize the resulting third input dimension, i.e., $\upsilon^i_{3,3}(j^i_c) = 2/\pi \cdot \upsilon^i_{3,2}(j^i_c)$.

*3.2.4 Normalize Second Input Dimension.* To distribute data uniformly along all the input space, we need to normalize the second input dimension (velocity motion influence) at each sampled level of the third input dimension, i.e.,

$$\upsilon^i_{2,2}(j^i_{c,l}) = \frac{\upsilon^i_{2,1}(j^i_{c,l})}{\max_{k\in S^i_{c,l}}|\upsilon^i_{2,1}(k)|} \quad (29)$$

$$\forall j^i_{c,l} \in S^i_{c,l} = \left\{ j^i_c : a_l \le \upsilon_{3,3}(j^i_c) < b_l \right\} \subset S^i_c \quad (30)$$

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 preprocessing and the final predictor are formulated as in Eqs. (26*a*) and (26*b*). Note that for the time steps where the joint remains static, no joint prediction is available, i.e., $\hat{e}^i_{p\ddot{\ell}}(j^i_{nc}) = 0, \forall j^i_{nc} \notin S^i_c$.

**3.3 Multiple NN Activation.** In order to enhance the prediction performance, the problem is divided into smaller prediction problems by means of multiple neural networks. To do this, we utilize prior knowledge of the error behavior to formulate 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 motion influence definition. Here, we confine each NN to a different motion stage according to the signs of the velocity and the acceleration motion influences as described in Table 3. In this way, the NNs 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 $e_{p\ddot{\ell}}$ 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, we study an exception where velocity remains almost constant for long periods. In this case, $e_{p\ddot{\ell}}$ tends to zero since the standard feedback and feedforward controller is normally designed to achieve satisfactory steady-state performance.

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

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

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

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

- $\chi^i_{p,AZ}$ : **Acceleration is constant and close to zero**, i.e., $\chi^i_{p,AZ}(j^i_c) = \left\langle \left|\upsilon^i_{3,3}(j^i_c)\right| \le \in^i_{3,3} \right\rangle \wedge \left\langle \left|\nabla\left[\upsilon^i_{3,3}(j^i_c)\right]\right| \le \varepsilon^i_{\nabla,3,3} \right\rangle$, where $\varepsilon^i_{3,3}$ and $\varepsilon^i_{\nabla,3,3}$ are small numbers.

**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 | | |

**Table 3 NN activation rule**

| NN Type | Velocity | Acceleration | Motion stage |
|---|---|---|---|
| 1 | Positive | Positive | Accelerating |
| 2 | Positive | Negative | Decelerating |
| 3 | Negative | Positive | Decelerating |
| 4 | Negative | Negative | Accelerating |

- $\chi^i_{p,VC}$ : **Velocity remains constant for a long period**, i.e., acceleration is close to zero for a long period, which can be obtained by analyzing the color plot of $\chi^i_{p,AZ}$ via *Dilate* and *Erode* image processing methods [14].
- $\chi^i_{p,AS}$ : **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., $\chi^i_{p,Vp}(j^i_c) = \left\langle v^i_{2,2}(j^i_c) > 0 \right\rangle$.
- $\chi^i_{p,Vn}$ : **Negative velocity**, i.e., $\chi^i_{p,Vn}(j^i_c) = \left\langle v^i_{2,2}(j^i_c) < 0 \right\rangle$.
- $\chi^i_{p,A}$ : **Initial acceleration, or concave acceleration when velocity is close to be constant**, i.e., $\chi^i_{p,A}(j^i_c) = \left\langle v^i_{3,3}(j^i_c) \gg 0 \right\rangle \vee \left\langle \nabla^2 \left[ v^i_{3,3}(j^i_c) \right] < 0 \wedge \chi^i_{p,AZ}(j^i_c) \right\rangle$.
- $\chi^i_{p,D}$ : **Final deceleration, or convex acceleration when velocity is close to be constant**, i.e., $\chi^i_{p,D}(j^i_c) = \left\langle v^i_{3,3}(j^i_c) \ll 0 \right\rangle \vee \left\langle \nabla^2 \left[ v^i_{3,3}(j^i_c) \right] > 0 \wedge \chi^i_{p,AZ}(j^i_c) \right\rangle$.

Then each of the four NNs defined in Table 3 can be activated by the following Boolean function $\chi^{it}_{NN}$ with the superscript $t$ as the type of NN

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

The $i$-th joint model following error prediction $\hat{e}_{p\ell}(S^{it}_{NN})$ from the $t$-th NN for the time steps enclosed by $S^{it}_{NN}$ can be then computed by the NN function defined in the next section. However, when velocity is constant for a long period, $\hat{e}_{p\ell}$ prediction is set to zero, i.e., $\hat{e}^i_{p\ell}(j^i_{n_{NN}}) = 0, \forall j^i_{n_{NN}} \notin \{ S^{i1}_{NN} \bigcup S^{i2}_{NN} \bigcup S^{i3}_{NN} \bigcup S^{i4}_{NN} \}$.

**3.4 Radial Basis Function NN.** With the identified error patterns, the radial basis function NNs [8,9] 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 [10].

Here, we utilize the two-layer NN structure based on the radial basis function (RBF) as follows

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

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

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

$$f_r(\vec{x}, \vec{\mu}, \sigma) = e^{-\frac{\|\vec{x}-\vec{\mu}\|_2^2}{\sigma^2}} \tag{34}$$

where $f^{it}_r(\vec{x}^i(j^{it}_{NN}), \mu^{it}, \sigma^{it})$ defines the $t$-th NN for the $i$-th joint, with $\vec{x}^i(j^{it}_{NN})$ as $[v^i_{2,2}(j^{it}_{NN}), v^i_{3,3}(j^{it}_{NN})]^{\mathrm{T}}$ for 2D networks, or $[v^i_{1,1}(j^{it}_{NN}), v^i_{2,2}(j^{it}_{NN}), v^i_{3,3}(j^{it}_{NN})]^{\mathrm{T}}$ for 3D networks. $N_{RBF} = \prod_{d=1}^{D} m_d$ denotes the total number of RBF neurons at the input layer, where $D$ (i.e., 2 or 3) is the number of the input dimensions and $m_d$ as the number of neurons in the $d$-th dimension. For the $m$-th neuron, the width of the RBF and its center position are preset and denoted respectively as $\sigma_m$, $^{i}_{3,3} \in \mathbb{R}$ and $\vec{\mu}_m \in \mathbb{R}^D$. Each dimension of $\vec{\mu}_m$ is uniformly distributed within $[-1, 1]$ due to the normalization. Preliminary experimental study is necessary to determine the density of neurons as a tradeoff between the performance and the computation/storage requirements. $\sigma_m$ is selected to ensure the overlap of the Gaussian functions among neurons.

The NN output $\hat{e}^i_{p\ell}(j^{it}_{NN})$, scaled by $\beta^i_0 \in \mathbb{R}$, is then defined 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 $\theta^{it}_0$ corresponds to the offset in the output prediction.

Before the prediction takes place, the parameter vector $\vec{\theta}^{it}$ is tuned in the NN training process. This is done by using the training data to minimize the following quadratic cost function, $V^{it}_T$, as:

$$V^{it}_T = \frac{1}{2} \sum_{j^{it}_{NN} \in S^{it}_{NN}} \left( e^i_{NN}(j^{it}_{NN}) \right)^2 \tag{35}$$

where $e^i_{NN}(j^{it}_{NN})$ is the prediction error by the $t$-th NN given the current $\vec{\theta}^{it}$, i.e., $e^i_{NN}(j^{it}_{NN}) = e^i_{p\ell}(j^{it}_{NN}) - \hat{e}^i_{p\ell}(j^{it}_{NN})$, and $e^i_{p\ell}(j^{it}_{NN})$ is the actual model following error collected by the end-effector sensor and using the inverse differential kinematics method proposed in Ref. [11] (the dashed part in Fig. 2). The optimized $\vec{\theta}^{it}$ for this least squares problem can be numerically obtained by gradient method with momentum [15,16] using heuristically adaptive step size and momentum gain.

From Eq. (35), we can derive the gradient of $V^{it}_T$ as $-(\beta^i_0)^2 \vec{V}^{it}_\Gamma$, where $\vec{V}^{it}_\Gamma$ is the accumulated regression error vector defined in Eq. (36). Thus, the gradient descent adaptation law can be formulated as Eq. (38), with $\gamma_h$ and $\eta_h$ as the adaptation gains for gradient and momentum, and the subscript/superscript $h$ denotes the adaptation iteration index. Notice that the same initial values can be used for $\gamma_h$ or $\eta_h$ in different training processes due to the normalization of $\vec{V}^{it}_\Gamma$ by $\beta^i_0$. A set of heuristic rules for this learning process is listed in Table 4.

$$\vec{V}^h_{\Gamma^{it}} = \sum_{j^{it}_{NN} \in S^{it}_{NN}} \vec{\Gamma}^{it}(\vec{x}^i(j^{it}_{NN}), \vec{\mu}^{it}, \sigma^{it}) \frac{e^{i,h}_{NN}(j^{it}_{NN})}{\beta^i_0} \tag{36}$$

$$\Upsilon_h = \mathrm{diag}\left( \gamma_h, \ldots, \gamma_h, \frac{\gamma_h}{100} \right) \in \mathbb{R}^{(N_{RBF}+1)\times(N_{RBF}+1)} \tag{37}$$

$$\vec{\theta}^{it}_{h+1} = \vec{\theta}_h + \Upsilon_h \vec{V}^h_{\Gamma^{it}} + \eta_h (\vec{\theta}^{it}_h - \vec{\theta}^{it}_{h-1}) \tag{38}$$

**3.5 Data Postprocessing.** 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 different NNs and the prediction unavailability 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 Eq. (17) to ensure that the predicted information is rich enough for control update.

## 4 Discussion of the Approach

**4.1 Stability and Safety Analysis.** The stability assurance for this NN based control as well as several safety measures are taken

**Table 4 Heuristic rules for adaptation gain selection**

| Type | Consecutive iterations | Action |
|---|---|---|
| Initial | — | $\eta_h = 0, \gamma_h = 7.5 \times 10^{-3}$ |
| $V^{it}_{T,h} < V^{it}_{T,h-1}$ | 1–5 | $\eta_h = 20\gamma_0 + \eta_{h-1}$ |
| | ≥6 | $\eta_h = \eta_{h-1}$ |
| $V^{it}_{T,h} >= V^{it}_{T,h-1}$ | 1–3 | $\eta_h = \eta_{h-1}/2$ |
| | 4 | $\eta_h = 0$ |
| | ≥5 | $\gamma_h = \gamma_{h-1}/2, \eta_h = 100\gamma_h$ |

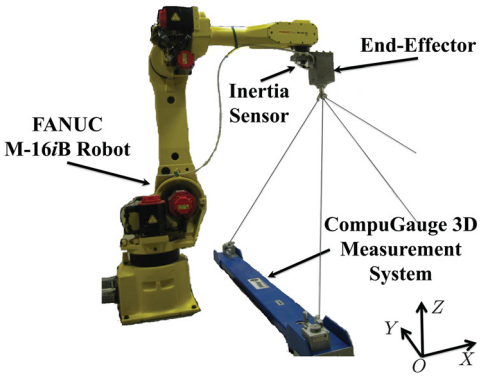



Fig. 5 FANUC M-16*i*B robot system

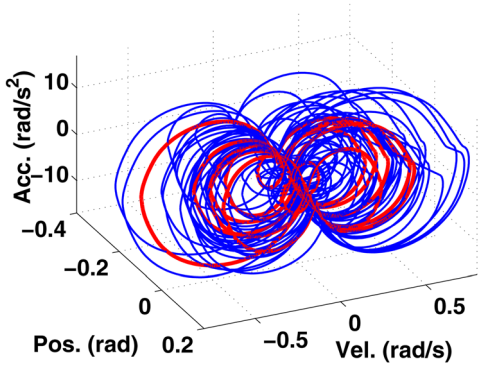



Fig. 6 Training (blue) and validation (red) trajectories for joint 3

into account during the design. On one hand, optimality and stability of the NN training are ensured by utilizing radial basis function neurons in a double layer network with a quadratic cost function and momentum gradient method [16]. On the other hand, the stability of feedforward control (17) can be assured by proper plant inversion filter design.

Furthermore, in order to increase prediction safety, data redundancy is utilized at the learning/training 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.

**4.2 Memory and Computation Requirements.** Note that most computation and experimental efforts occur in the neural network training phase, which is only a one-time practice (in robot factory tuning/testing stage). Thus, 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[5] is available for off-line learning/training computation. In this way, one computer could be utilized for NN training and learning control of several different robots for cost saving. After learning/training (during robot manufacturing/tuning stage), the standard local robot controller can efficiently process the prediction as a simplified look-up table for $\vec{\theta}^{it}$ and a linear regression in Eq. (31) for any general motions within the trained workspace.

## 5 Experimental Study

**5.1 Test Setup.** The proposed method is implemented on a 6-joint industrial robot, FANUC M-16*i*B/20, in Fig. 5. The robot is equipped with built-in motor encoders for each joint. An inertia sensor (Analog Devices, ADIS16400) containing a 3-axial accelerometer 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 rates of all the sensor signals as well as the real-time controller implemented through MATLAB xPC Target are set to 1 kHz.

[5]Quantification of the required computation resources depends on various factors, such as the actual robotic system, the implementation platform, and the tools that are used (e.g. compilers, optimization options, and variable data types).

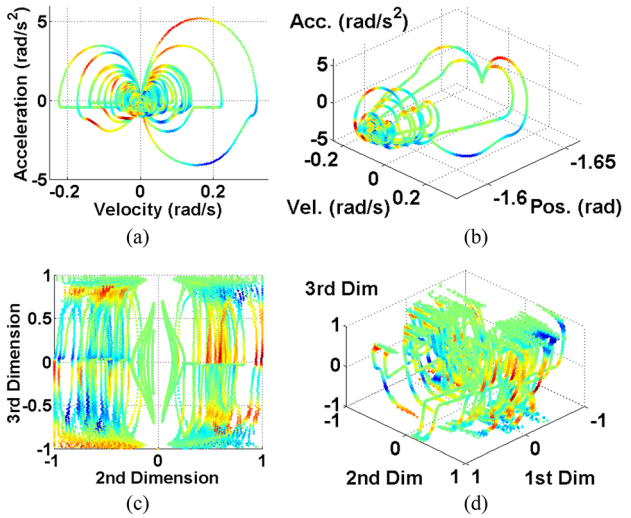



Fig. 7 Joint 5 model following error. Color: *Red* = 4, *Blue* = −4, *Green* = 0 in [rad/s²]. (*a*) and (*b*) are 2D and 3D distributions, respectively, based on joint 5 reference before preprocessing stage. (*c*) and (*d*) are 2D and 3D distributions, respectively, based on joint 5 motion influence (the reference combination from all joints) after preprocessing stage.

**5.2 Algorithm Setup.** For learning control algorithms (16) and (17), the zero-phase acausal low-pass filter $Q_u$ is set with a cut-off frequency of 20 Hz for each joint. The cut-off frequency for the NN output filtering is set to 50 Hz.

In order to train the NNs, experiments are performed to obtain the model following error variations 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 preprocessed for the NN training. Figure 7 shows the training data distribution before and after applying the motion influence definition and the preprocessing 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 NN as a trade-off between the performance and the computational viability (i.e., in general, denser neuron distribution can lead to better performance but more computational and experimental efforts). The neuron distribution ranges (Table 5) are set to be equal or larger than the expected input range. The

Table 5 Neuron distribution ranges for each NN

| NN Type | First dimension | Second dimension | Third 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] |

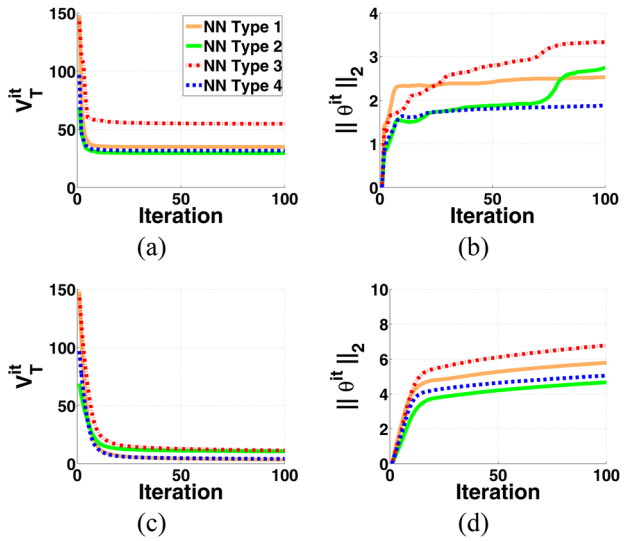



**Fig. 8 Cost function convergence and parameter adaption over iteration during NN training process**

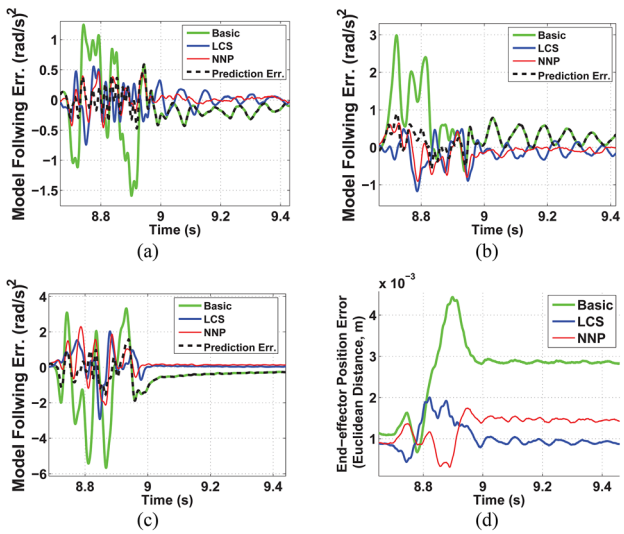



**Fig. 9 Experimental performance comparisons for error reductions. (*a*)–(*c*) the model following error on joint 2, 3, and 5, respectively. (*d*) the end-effector position error (in Euclidean distance, $\sqrt{e_X^2 + e_Y^2 + e_Z^2}$).**

RBF width $\sigma$ for each neuron is set to 0.07 to ensure overlap between neurons (recall that the input range of each dimension is normalized to be unitless as [–1, 1]). The NN training process (38) is successfully accomplished with the fast convergence of the cost function shown in Fig. 8.

**5.3 Experimental Results.** The performance of executing the validation trajectory (Fig. 6, red trajectory) with the controller structure in Fig. 1 is compared for three different nonlinear feedforward torques: the basic nominal feedforward torque $\tau_{nl,0}$ from Eq. (12) (Basic), the proposed feedforward torque based on neural network prediction (NNP) (i.e., using Eq. (17) where $\hat{e}_{p\ddot{\ell}}$ is obtained from Eq. (31)), and the learning control based on the end-effector sensor (LCS) (i.e., using Eq. (16) once where $e_{p\ddot{\ell},k}$ is obtained from a prior iteration with the end-effector sensor) [13].

Since the feedforward control (17) aims at model matching for the inner plant during moving periods, results in these regions show that the NNP 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 9 shows a graphical comparison of the error reduction[6] on joint 2, 3, 5, and end-effector, using *Basic*, LCS, NNP, and the prediction error $e_{NN}$. This result is confirmed with Table 6, which shows that the NNP achieves a substantial performance enhancement at the end-effector besides the model matching performance. Note that the NNP also improves the performance at the static period (i.e., robot joint reference is not moving) where the prediction is not viable. This static performance improvement is expected as a result of significant improvement at the dynamic period (i.e., robot joint reference is moving).

**Table 6 Percentage of LCP performance compared with LCS, i.e., $|\,\|\,e_{LCP}\,\|_2 - \|\,e_{Basic}\,\|_2\,|/|\,\|\,e_{LCS}\,\|_2 - \|\,e_{Basic}\,\|_2\,| \times 100\%$**

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

## 6 Conclusions

This paper investigated a feedforward input generation scheme based on model following error prediction, which suggested a viable solution in the industry for end-effector performance enhancement when production line requires flexibility and efficiency. The proposed method improved feedforward torque compensation based on predicted error by multiple NNs. The robot dynamics and error characteristics were explored and the NN predictor was accordingly designed with novel input definition and data preprocessing stage. The radial basis function was utilized in the two-layer NNs and the problem was further divided into four smaller NNs for effective learning. Experimental study on a 6-DOF industrial robot showed a noticeable performance improvement over the basic controller for the end-effector position tracking and residual vibration reduction, during both dynamic and static periods without learning for a new trajectory.

## References

[1] Asensio, J., Chen, W., and Tomizuka, M., 2012, "Robot Learning Control Based on Neural Network Prediction," Proceedings of the ASME 2012 Dynamic Systems and Control Conference, pp. 1489–1497.
[2] 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.
[3] Cuiyan, L., and Dongchun, Z., 2004, "A Survey of Repetitive Control," IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 1160–1166.
[4] Gopinath, S., Kar, I., and Bhatt, R., 2008, "Experience Inclusion in Iterative Learning Controllers: Fuzzy Model Based Approaches," Eng. Applic. Artif. Intell., **21**(4), pp. 578–590.
[5] 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, Vol. **2417**, pp. 29–38.
[6] Chien, C.-j., 2008, "A Combined Adaptive Law for Fuzzy Iterative Learning Control of Nonlinear Systems With Varying Control Tasks," IEEE Trans. Fuzzy Syst., **16**(1), pp. 40–51.
[7] Freeman, C. T., Alsubaie, M. A., Cai, Z., Rogers, E., and Lewin, P. L., 2011, "Initial Input Selection for Iterative Learning Control," ASME J. Dyn. Syst., Meas., Control, **133**, p. 054504.
[8] Moody, J., and Darken, C., 1989, "Fast Learning in Networks of Locally-Tuned Processing Units," Neural Comput., **1**, pp. 281–294.
[9] Poggio, T., and Girosi, F., 1990, "Networks for Approximation and Learning," Proc. IEEE, **78**, pp. 1481–1497.
[10] Viñuela, P. I., and Galván, I. M., 2004, *Redes de Neuronas Artificiales. Un Enfoque Práctico*, Pearson Education, Madrid, Spain.

[6]Note that the LCS in Fig. 9 exhibits overcorrection (e.g., LCS error in static periods is about 180 deg phase difference from the Basic setting) because the static error has already been reduced by the correction of the dynamic error. This overcorrection will disappear as the LCS continues learning iteratively. The NNP does not have this behavior since it is turned off during the static periods. However, this switching (around 8.9sec) causes the Cartesian space performance change of NNP (making it worse than LCS).

[11] Chen, W., and Tomizuka, M., 2012, "Load Side State Estimation in Elastic Robots With End-effector Sensing," IEEE/ASME International Conference on Advanced Intelligent Mechatronics, pp. 598–603.
[12] de Wit, C. C., Bastin, G., and Siciliano, B., 1996, *Theory of Robot Control*, 1st ed. Springer-Verlag New York, Inc., Secaucus, NJ.
[13] Chen, W., and Tomizuka, M., 2012, "Iterative Learing Control With Sensor Fusion for Robots with Mismatched Dynamics and Mismatched Sensing," Proceedings of the ASME 2012 Dynamic Systems and Control Conference, pp. 1480–1488.
[14] Bovik, A. C., 2009, *The Essential Guide to Image Processing*, Elsevier, Burlington, MA.
[15] Rumelhart, D., Hintont, G., and Williams, R., 1986, "Learning Representations by Back-Propagating Errors," Nature, **323**(6088), pp. 533–536.
[16] Torii, M., and Hagan, M., 2002, "Stability of Steepest Descent With Momentum for Quadratic Functions," IEEE Trans. Neural Netw., **13**(3), pp. 752–756.