Neural-Learning Trajectory Tracking Control of Flexible-Joint Robot Manipulators with Unknown Dynamics

Shuyang Chen[1], *Student Member, IEEE* and John T. Wen[2], *Fellow, IEEE*

***Abstract*— Fast and precise motion control is important for industrial robots in manufacturing applications. However, some collaborative robots sacrifice precision for safety, particular for high motion speed. The performance degradation is caused by the inability of the joint servo controller to address the uncertain nonlinear dynamics of the robot arm, e.g., due to joint flexibility. We consider two approaches to improve the trajectory tracking performance through feedforward compensation. The first approach uses iterative learning control, with the gradient-based iterative update generated from the robot forward dynamics model. The second approach uses dynamic inversion to directly compensate for the robot forward dynamics. If the forward dynamics is strictly proper or is non-minimum-phase (e.g., due to time delays), its stable inverse would be non-causal. Both approaches require robot dynamical models. This paper presents results of using recurrent neural networks (RNNs) to approximate these dynamical models – forward dynamics in the first case, inverse dynamics (possibly non-causal) in the second case. We use the bi-directional RNN to capture the noncausality. The RNNs are trained based on a collection of commanded trajectories and the actual robot responses. We use a Baxter robot to evaluate the two approaches. The Baxter robot exhibits significant joint flexibility due to the series-elastic joint actuators. Both approaches achieve sizable improvement over the uncompensated robot motion, for both random joint trajectories and Cartesian motion. The inverse dynamics method is particularly attractive as it may be used to more accurately track a user input as in teleoperation.**

I. INTRODUCTION

Motivated by the need for fast and precise motion in manufacturing, robot tracking control has long been of interest [1]. A common joint torque control architecture consists of linear joint position and velocity feedback for stabilization and model-based feedforward for trajectory tracking. If the feedforward is expressed in the linear-in-parameter form, adaptive control may be applied to estimate the uncertain parameters online while achieving perfect tracking asymptotically [2]. To further remove the requirement for the expression of the nonlinear robot dynamics, iterative learning control (ILC) [3] may be used, but only for a specified desired trajectory. Neural networks (NNs) based controller has been proposed to extend the learning paradigm to arbitrary desired trajectory [4]. These approaches have been extended to flexible joint robots, as in some collaborative industrial robots and space robots [5]–[7].

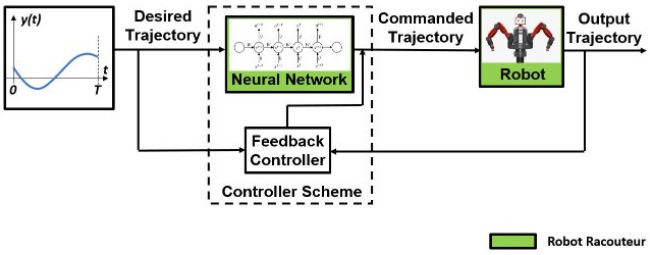


Fig. 1: Block diagram of the proposed trajectory tracking flow.

Most industrial robots already have a joint torque controller and only allow the joint setpoint adjustment at certain rate. This paper considers neural-learning control for the outer loop robot tracking control for a flexible joint robot (Baxter) based on recurrent neural networks (RNNs). It has already been demonstrated that RNNs can be used to approximate dynamical systems due to their internal recurrent structure [8]. Here, we propose and compare two approaches based on deep RNNs. Fig. 1 shows the schematics of the control architecture. The RNN is used as a feedforward controller to produce the control input, which is combined with the feedback term to determine the commanded input to the robot. For the first approach, we train a deep unidirectional RNN to approximate the forward dynamics of the robot manipulator by collecting the response data of the manipulator for a large amount of specified trajectories. Then, we iteratively refine a given desired trajectory offline with the learned model to get the optimal input for the trajectory. Compared to the iterative refinement implemented on the physical robot, our method is much more time efficient. For the second approach, we train another RNN to approximate the inverse dynamics of the robot manipulator. Considering the non-causality of the inverse dynamical system, we train a bidirectional RNN (BRNN) [9] with the same set of collected data. We report these two approaches here out of two reasons. First, they can both improve the trajectory tracking performance, given that the entire desired trajectory is known in advance. Second, we have the similar workload for RNNs training due to the one-time data collection. However, for a desired trajectory that is not determined completely in advance but is generated online via user teleoperation, the dynamical inversion approach is preferred for online dynamics

This research was sponsored in part by the New York State Matching Grant program and by the Center for Automation Technologies and Systems (CATS) at Rensselaer Polytechnic Institute under a block grant from the New York State Empire State Development Division of Science, Technology and Innovation (NYSTAR).

[1]Shuyang Chen is with the Department of Mechanical, Aerospace, and Nuclear Engineering, Rensselaer Polytechnic Institute, 110 8th St, Troy, NY 12180, USA chens26@rpi.edu

[2]John T. Wen is with the Department of Electrical, Computer, and Systems Engineering, Rensselaer Polytechnic Institute, Troy, NY 12180 USA. wenj@rpi.edu

compensation with its capability of real-time inference. The first approach, which requires the knowledge of the entire trajectory for iterative refinement, is not suitable anymore. To our best knowledge, there are few works exploring and comparing the two approaches using RNN/BRNN as feedforward compensation for robot manipulators trajectory tracking control with unknown dynamics.

To validate the effectiveness and generalization capability of the proposed approaches. We conduct experiments for tracking an unseen multi-joint sinusoidal trajectory, a random joint trajectory, and a Cartesian trajectory using two approaches. For those entirely given desired trajectories, the corresponding feedforward controller input trajectory is obtained either through ILC with the trained RNN (the first approach), or directly filtered through the BRNN (the second approach). We test that, for either of the two approaches, we can apply additional iterations of ILC with the resulted feedforward trajectory on the physical manipulator to further improve the tracking performance. We also validate the effectiveness of the dynamical inversion approach for real-time online compensation of a user-teleoperated trajectory. Instead of obtaining the entire feedforward input trajectory offline, the feedforward input at each time step is obtained by the trained BRNN in real-time. For all above experiments, at each time step, the feedforward control input is combined with a feedback term obtained by a proportional feedback controller, and the combined input will be used as the commanded joint setpoint to the robot manipulator. By comparing with the performance of a baseline feedback controller, we demonstrate that both of two approaches work effectively in reducing the trajectory tracking error.

The rest of the paper is organized as follows. Section II summarizes related work of advanced approaches on trajectory tracking control, including NNs, ILC, and differential dynamic programming (DDP), and highlights the difference from our approaches. Section III states the problem, followed by the detailed description of methodology in Section IV. The experimental results are presented in Section V. Finally, Section VI concludes the paper and adds some new insight.

II. Related Work

There have been a lot of works for improving trajectory tracking performance of robot manipulators through controllers design based on learning dynamics models or inverse dynamics through NNs [10]. NNs, with their strong generalization and approximation capability given enough training data, have attracted more and more attention in control community for controller design. A detailed review of neural-learning control can be referred in [11].

Candidate NNs, including radial basis function (RBF) NN, RNN, and multi-layer feedforward NN, have been explored to approximate the robot forward dynamics model for adaptive trajectory tracking controller design [12]–[16]. In [17], an adaptive controller designed using RBF NN producing joint torque input was proposed to compensate the unknown dynamics and a payload for a Baxter robot. Compared to their work, instead of using torque level control, we use joint setpoint control. Also, it is a non-trivial task to determine the number of Gaussian kernels and their centers as well as shapes for RBF NN. They trained a large amount of parameters (in the order of 10^5) for RBF NN to approximate the robot dynamics model, whereas we have much less amount of parameters (in the order of 10^3) considering that the parameters are shared among all RNN time step cells. In addition, in their work, Lyapunov stability theory was applied to guarantee the bounded tracking error and also derive the NNs parameters update law. However, there is no guarantee that the model estimation error will also converge. They used separate RBF NNs to approximate robot dynamical model components individually including inertial matrix M, Coriolis matrix C, and gravity torque G whereas they did not guarantee the properties of the robot dynamics model such as the positive-definiteness of M. On the contrary, we use supervised learning to train the RNNs given collected input/response data to directly approximate the robot internal dynamics as one integrated model. Finally, the dynamics model they tried to approximate is for rigid-joint robot manipulators instead of flexible-joint ones.

Works have also been reported of using NNs to learn an inverse dynamics model for dynamics compensation [18]. Li et al. [19] proposed an NN-based control architecture with an NN pre-cascaded to a quadrotor under a feedback controller. Compared to their approach, our method is different in the following aspects. First of all, we apply the trained RNNs to an articulated robot manipulator, which has very different dynamics from the quadrotors. Next, we train RNNs to approximate the feedfoward and inverse dynamics while they use multi-layer feedforward NNs. And specifically we train a BRNN to address the noncausality of the inverse of a strictly proper system. It has been also shown that RNN can outperform Gaussian Processes [20] in performance of learning the inverse dynamics of a robot manipulator with linear time complexity, given sufficient training data [21].

ILC is another approach to improve the tracking performance by searching for the optimal input trajectory for a desired trajectory via iteratively updating the control input with previous tracking errors. The main drawback of ILC is its non-transferability. For each new desired trajectory, ILC must be re-implemented to search for the optimal input. In our work [22], we trained multi-layer feedforward NNs offline to find a good estimate of the inverse dynamics of an industrial ABB robot for trajectory tracking control, by implementing ILC for a large number of desired trajectories to collect training data in a high-fidelity simulator. We also applied transfer learning to transfer the learned representations in simulator to the real-world by fine-tuning the NNs with real-world data. However, the above techniques cannot be applied to Baxter robots due to the following reasons. First, there is no good dynamic simulator for Baxter, which makes it tedious to implement ILC on the physical robot for a large amount of trajectories. Then, the joints of ABB robot are decoupled, thus we can train 6 NNs, with each NN approximating the dynamical inversion for one joint. But the joints of Baxter are coupled as shown in Fig. 2.

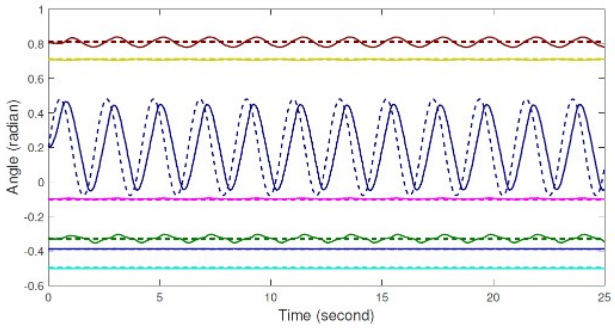


Fig. 2: Proof of coupling of Baxter joints. We only command active motion of joint 4 (the blue sinusoid in the middle of the figure) with other joints fixed. The dashed and solid lines indicate desired and actual joint angles, respectively. The lines of different colors indicate different joints. The figure shows that the output of some joints have similar motion pattern to joint 4, although they are commanded not moving.

Thus, we must train one NN to couple all 7 joints instead of training separate NNs. To address this, we train RNNs with 2D inputs, which are composed of roll-out of a segment of a joint trajectory for all 7 joints. Finally, since we directly collect training data from the physical robot, no transfer learning is needed.

DDP [23] is another gradient-based algorithm that uses a locally-quadratic dynamics model to search for local optimal control inputs. For nonlinear systems, the control inputs will be iteratively updated by applying the linear quadratic regulator (LQR) to the linearized system about the current trajectory. In general, it is impractical to solve DDP at runtime.

III. Problem Statement

Here we aim at achieving improved trajectory tracking for the left arm of a Baxter robot. The arm is under closed loop joint servo control with input $u \in \mathbb{R}^n$ (here we use the joint position command q_c), and the output $q \in \mathbb{R}^n$, the measured joint position. Our goal is to compensate the robot inner loop dynamics $\mathcal{G}$, by learning a mapping from the desired trajectory q_d to the feedforward control q_f (as shown in Fig. 3), which will be combined with real-time feedback control input to produce the commanded input q_c to achieve improved tracking performance of the overall system, compared to the baseline feedback controller.

IV. Methodology

A. Proposed Control Law

We propose the controller composed of feedforward and feedback components indicated in Fig. 3. The associated control law is as below:

$$q_c(t+1) = q_f(t+1) - k \cdot (q(t) - q_d(t)) \qquad (1)$$

where $q_d(t)$ is the desired joint position at time t, $q(t)$ is the real time output of robot, and $q_f(t+1)$ is the feedforward command at time $t+1$, which is obtained

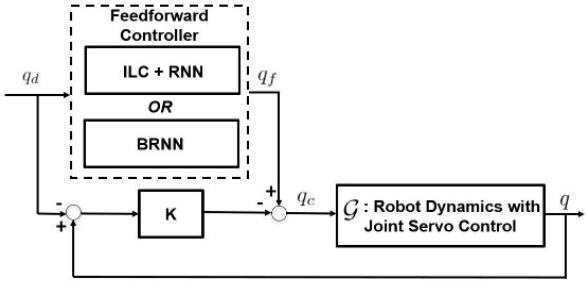


Fig. 3: Overall control architecture.

either by implementing ILC with the trained unidirectional RNN that approximates the manipulator forward dynamics, or directly filtering by the dynamical inversion approximated by the BRNN. $q_c(t+1)$ is the resulting commanded joint input for robot manipulator at time $t+1$.

On the other hand, the stable baseline feedback controller is:

$$q_c(t+1) = q_d(t+1) - k \cdot (q(t) - q_d(t)) \qquad (2)$$

B. Feedforward Controller using RNN and ILC

The feedforward control input q_f can be obtained by implementing ILC with the trained RNN. In order to train an RNN with large capacity of memory by supervised learning, we need a large amount of input/output training data.

1) Data Collection: We collect raw response joint position data of the left arm of Baxter for in total 500 joint trajectories q_d, among which 100 trajectories are purely random and another 400 are sinusoids with varying magnitudes and frequencies. We choose the trajectories by obeying the joint limits of Baxter, as well as trying to cover a feasible portion of Baxter joint space. The manipulability measure [24] of 500 training joint trajectories as well as the 4 testing joint trajectories (described in Section V) are plotted in Fig. 4. Each trajectory q_d contains 2500 joint setpoints for all 7 joints (thus a 7 by 2500 matrix) and is commanded to Baxter at 100 Hz. At the same time, the output joint trajectory q of the manipulator is collected. Then, our goal is to obtain an approximation of the forward dynamics $\mathcal{G}$ by learning from a set of (q_d, q).

2) Unidirectional RNN Training: We train a unidirectional RNN as shown in Fig. 5a to represent the forward dynamics of the manipulator considering the causality of the dynamical system, and the coupling of Baxter joints.

In specific, during training, the input to the RNN is a sequence of a desired joint trajectory containing T time steps setpoints $\underline{q}_d(t) := \{q_d(\tau) : \tau \in [t, t+T-1]\}$. Each $q_d(\tau)$ is a 7 by 1 vector denoting the joint position setpoint on a desired trajectory q_d for all 7 joints at time τ. The output of the RNN is the joint position $q(t+T)$ (also a 7 by 1 vector) of the output trajectory q. Thus, for each input q_d and output q trajectory containing 2500 joint position setpoints, we can get $2500 - T$ training samples $(\underline{q}_d(t), q(t+T))$.

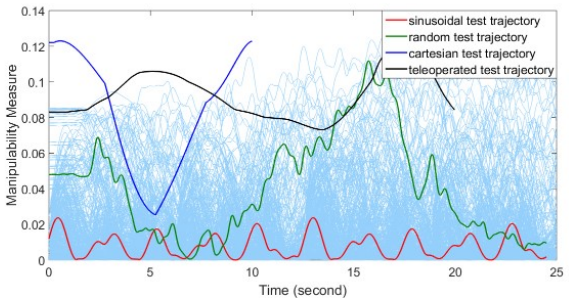
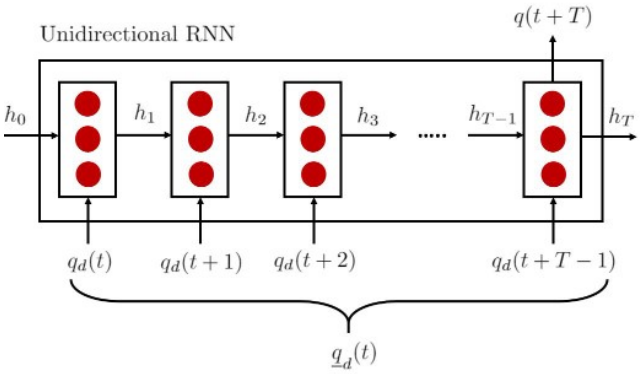
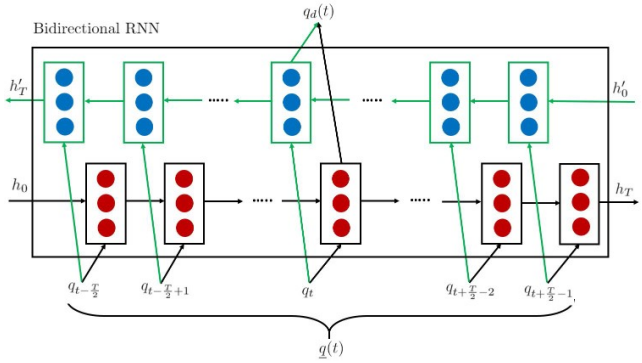


Fig. 4: Manipulability measure over 500 training joint trajectories (shown as the light blue background) and 4 testing joint trajectories.



(a) Unrolled unidirectional RNN with T time steps of inputs.



(b) Unrolled bidirectional RNN with T time steps of inputs.

Fig. 5: Unrolled RNNs structures. h_i represents the internal states of each time step.

We choose $T = 50$ to guarantee that the input of RNN contains enough dynamics information for the RNN to model. In total we have 1225000 samples, and we use 80 % of the samples for training and 20 % for testing. We train a 4-layer unidirectional RNN with Gated Recurrent Units (GRUs) instead of LSTM cells after comparison in `TensorFlow`. We use `AdamOptimizer` with an initial learning rate of 1×10^{-3} to tune the parameters of RNN by minimizing the mean squared error (MSE) between the predicted output of RNN and the expected output over a randomly selected batch of 256 training samples in each training iteration. Also, we use dropout [25] of 0.5 to avoid overfitting. The training process converges after 10000 training iterations.

After training, the RNN can provide a mapping from a sequence of 50 time steps ($T = 50$) joint position inputs (a 7 by 50 matrix) $\underline{q}_d(t)$ to the predicted output $q_o(t+50)$ (a 7 by 1 vector):

$$\underline{q}_d(t) := \{q_d(\tau) : \tau \in [t, t+49]\} \rightarrow q_o(t+50) \qquad (3)$$

3) Iterative Learning Control: We can implement ILC with the trained RNN that emulates the forward dynamics of the manipulator to search for the optimal input for a given desired trajectory. We use the multiple-input multiple-output (MIMO) gradient-based ILC algorithm as developed in [22] based on the work in [26]. The key idea of the algorithm is that, we iteratively update the input u^k (k represents the number of iterations) until convergence with the gradient of the tracking error with respect to the current input u^k as below:

$$u^{k+1} = u^k - \alpha_k G^*(s) e_q \qquad (4)$$

where $G^*(s)$ represents the adjoint of a linear time invariant system $G(s)$ approximating the internal dynamics $\mathcal{G}$, and $e_q := (q^k - q_d)$, which is the tracking error using u^k as input. $G^*(s)e_q$ represents the gradient of the tracking error with respect to u^k. α_k is the optimal learning rate for k^{th} iteration and can be obtained by a 1-d line search. The above iterative refinement algorithm is easily implemented with the trained RNN. u^0 is simply chosen to be the desired output trajectory q_d to start the iteration.

The drawback of the above method is that although it is much faster (compared to implementation on the physical system) to implement ILC with the RNN offline to find the feedforward control input for a given desired trajectory, it still takes several seconds until the convergence of ILC. Thus, it is only suitable for an entirely given trajectory and will not work for real-time compensation.

C. Feedforward Controller using BRNN

The feedforward control input q_f can also be obtained by directly filtering q_d through the trained BRNN, which is used to approximate the manipulator inverse dynamics. In this way, there are no iterations required thus it is suitable for real-time compensation.

1) Data Collection: Instead of collecting training data through ILC as we did in [22], we can use the same collected training data above but use the output trajectory q as input and the desired trajectory q_d as output to the BRNN, inspired by [19]. The idea behind this is that, if q_d and q are corresponding input and output trajectories for a dynamical system $\mathcal{G}$, then with input of q, the inverse dynamical system would produce output of q_d.

2) Bidirectional RNN Training: Here, we train a BRNN as shown in Fig. 5b to represent the inverse dynamics of the manipulator considering the possible noncausality of the inverse of a non-minimum-phase system.

In specific, the input to the BRNN during training is a sequence of an output joint trajectory q containing T time steps $\underline{q}(t) := \{q(\tau) : \tau \in [t - T/2, t + T/2 - 1]\}$. The output of the BRNN is the input joint position $q_d(t)$. Thus, for each input q_d and output trajectory q containing 2500 joint position setpoints, we can get 2501-T training samples $(\underline{q}(t), q_d(t))$.

Similar to the training of the unidirectional RNN, we choose $T = 50$, which will result in 1225500 samples, and we use 80 % for training and 20 % for testing. We train a 2-layer BRNN with GRU cells in `TensorFlow` by `AdamOptimizer` with an initial learning rate of 1×10^{-3} to tune the parameters by minimizing the MSE between the predicted output of BRNN and the expected output over a randomly selected batch of 256 training samples in each training iteration.

After training, the BRNN can provide a mapping from a sequence of 50 time steps of desired joint position inputs (a 7 by 50 matrix) $\underline{q}_d(t)$ to a feedforward control input $q_f(t)$ (7 by 1):

$$\underline{q}_d(t) := \{q_d(\tau) : \tau \in [t - 25, t + 24]\} \rightarrow q_f(t) \quad (5)$$

D. Resolved Velocity Controller for Human Teleoperation

To facilitate the human teleoperation of the redundant manipulator, we develop the resolved velocity controller based on our work in [27] as below:

$$\min_{\dot{q}, \alpha_r, \alpha_p} ||J\dot{q} - \alpha v_d||^2 + \epsilon_r(\alpha_r - 1)^2 + \epsilon_p(\alpha_p - 1)^2 \quad (6)$$

subject to equality constraints such as orientation control: $h_E(q) = 0$, and inequality constraints such as joint limits: $h_I(q) > 0$, where J is the Baxter left arm Jacobian and $\dot{q}$ is the resolved robot joint velocity for the user commanded velocity v_d. And

$$\alpha = \begin{bmatrix} \alpha_r I_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & \alpha_p I_{3\times3} \end{bmatrix}$$

in which α_r scales the angular velocity part of the human commanded velocity v_d, and α_p scales the linear velocity part. ϵ_r, ϵ_p are two scaling factors. The resulted trajectory will be used as input to the BRNN to produce the feedforward control input q_f in real-time.

E. Robot Raconteur Bridge

We use Robot Raconteur (RR) [28] as middleware to coordinate robot manipulator control and communication, joystick teleoperation, and feedforward controller input inference using the trained RNNs with separate RR services, as shown by the green blocks in Fig. 1. The overall coordination is conducted by a MATLAB script that connects to these services as a client. The RR-Baxter bridge software is available at https://github.com/rpiRobotics/baxter_rr_bridge.

V. Results

A. Tracking a Multi-Joint Trajectory

Fig. 6 shows the results of tracking an unseen sinusoidal joint trajectory using two approaches. Table I compares the tracking errors in terms of ℓ_2 and ℓ_∞ norms (using the error vectors at each sampling instant) for each joint, using the baseline feedback controller, the first approach and the second approach, respectively. From the figure and the table, we can see that the feedforward control input obtained by either of the two approaches can improve the tracking performance a lot compared to the baseline feedback controller (in average over 50 % of improvement). Also, the second approach works slightly better than the first method. One possible reason is the initial transient process at the beginning when using ILC [22].

B. Tracking a Random Joint Trajectory

We did another test of tracking an unseen random joint trajectory using two approaches. Table II summarizes the results of tracking errors. From the table, we can see that the feedforward control input obtained by both approaches can improve the tracking performance obviously compared to the baseline feedback controller (in average over 40 % of improvement). Also, the second approach still works slightly better than the first one.

C. Tracking a Cartesian Trajectory

Fig. 7 shows the results of using two approaches to track a Cartesian trajectory, which is a square in the $x - y$ plane with $z = 0.2\ m$. During the motion, the orientation of the end-effector is fixed. Table III compares the corresponding tracking errors for each Cartesian direction. From the figure and the table, we show that the tracking performance is significantly improved with the feedforward control inputs (in average over 50 %), as compared to the baseline feedback controller.

We also find that the tracking performance for all test cases above can be further improved by implementing additional ILC iterations on the physical robot, using the feedforward-compensated trajectory as the initial input u^0 in (4).

D. Tracking with Human-Teleoperated Motion

We may also encounter the case that the desired trajectory is not given completely and the feedforward controller must work in real-time to produce the control input. In this case we can only use the BRNN. To verify the feasibility of the BRNN for online compensation, we use an Xbox joystick to teleoperate the left arm of Baxter using the resolved velocity controller in (6), in which case the desired trajectory is randomly planned online by a user. Note that at each time step t, we need to collect future 24 user inputs (as from (5)) to get the required 50 ($T = 50$) input joint setpoints of the BRNN, which will cause about 0.24 s delay corresponding to 100 Hz communication rate. The delay can be reduced by using a smaller T, but if T is too small, there will not be enough information for the BRNN to capture the inverse dynamics of the manipulator.

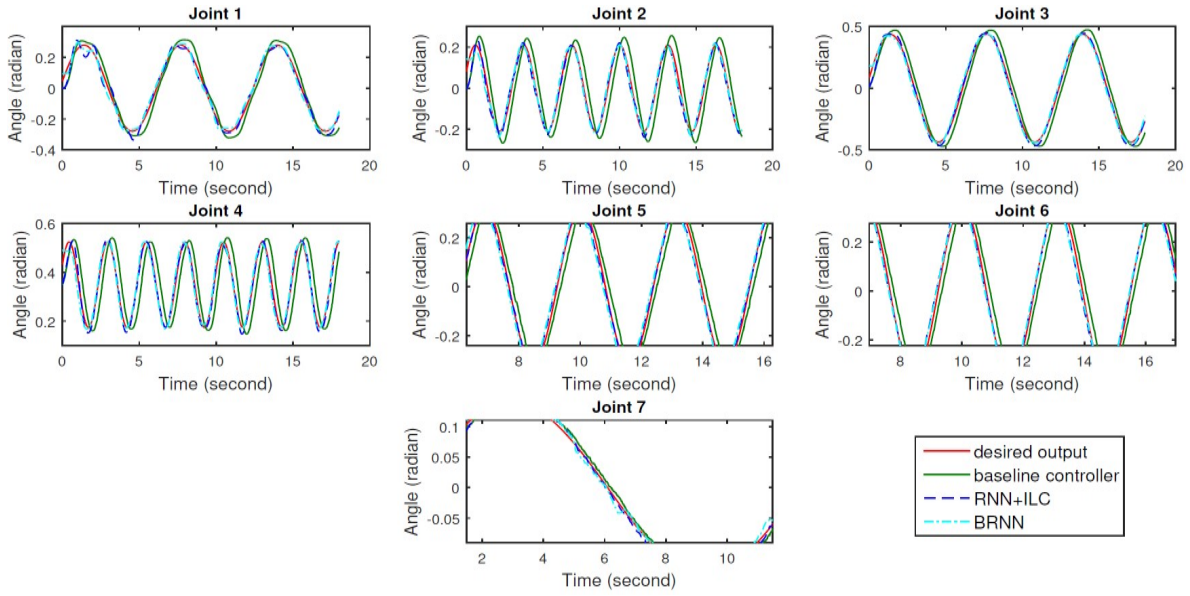


Fig. 6: Comparison of tracking performance without and with the RNN/BRNN feedforward controllers of sinusoidal joint trajectories. The plots of joint 5 to 7 are zoomed in to better visualize the difference.

TABLE I: ℓ_2 and ℓ_∞ norm of tracking errors of 7-dimensional sinusoidal joint trajectories using baseline controller, the first approach and the second approach.

Joint	Baseline Controller Tracking Error (rad)		RNN + ILC + Feedback Tracking Error (rad)		BRNN + Feedback Tracking Error (rad)	
	ℓ_2	ℓ_∞	ℓ_2	ℓ_∞	ℓ_2	ℓ_∞
1	2.3816	0.0973	1.3622	0.0884	0.8088	0.0590
2	4.0453	0.1296	1.2072	0.1124	0.7163	0.0545
3	3.4910	0.1029	1.2245	0.0942	0.7328	0.0727
4	4.1703	0.1437	0.9874	0.0962	0.5944	0.0391
5	2.5834	0.0917	1.0075	0.0672	0.7247	0.0530
6	2.9346	0.0955	1.1818	0.0808	0.7121	0.0593
7	0.3359	0.0133	0.2845	0.0178	0.2523	0.0155

TABLE II: ℓ_2 and ℓ_∞ norm of tracking errors of 7-dimensional random joint trajectories using baseline controller, the first approach and the second approach.

Joint	Baseline Controller Tracking Error (rad)		RNN + ILC + Feedback Tracking Error (rad)		BRNN + Feedback Tracking Error (rad)	
	ℓ_2	ℓ_∞	ℓ_2	ℓ_∞	ℓ_2	ℓ_∞
1	4.1746	0.2515	2.1068	0.1495	2.1974	0.1478
2	3.8348	0.2129	1.9292	0.1131	1.7717	0.1357
3	3.7421	0.2193	2.1502	0.1642	1.9836	0.1458
4	2.9880	0.1614	1.9791	0.1366	1.5006	0.1015
5	1.9861	0.1156	1.7605	0.0974	1.3623	0.1179
6	2.2318	0.167	1.7216	0.1153	1.3887	0.1223
7	1.8733	0.1448	1.3336	0.1104	1.2803	0.0948

Fig. 8 shows the results of tracking one random teleoperated joint trajectories using the BRNN feedforward controller. Table IV compares the tracking errors. The figure and the table highlight that the feedforward control input obtained by BRNN can effectively improve the tracking performance for a randomly generated trajectory compared to the baseline feedback controller (in average over 30 % of improvement). This real-time compensation feature makes it suitable for tracking a trajectory in a timely manner.

VI. Conclusions and Future Work

In this paper, we explored two approaches for trajectory tracking control of a flexible-joint robot manipulators with

TABLE III: ℓ_2 and ℓ_∞ norm of tracking errors of a Cartesian trajectory in three axes using baseline controller, the first approach and the second approach.

Axis	Baseline Controller Tracking Error (m)		RNN + ILC + Feedback Tracking Error (m)		BRNN + Feedback Tracking Error (m)	
	ℓ_2	ℓ_∞	ℓ_2	ℓ_∞	ℓ_2	ℓ_∞
x	0.4478	0.0363	0.1306	0.0124	0.1223	0.0129
y	0.5694	0.0459	0.1808	0.0199	0.1685	0.0200
z	0.0582	0.0084	0.0455	0.0068	0.0273	0.0023

TABLE IV: ℓ_2 and ℓ_∞ norm of tracking errors of 7-dimensional user teleoperated joint trajectories using baseline controller and the second approach.

Joint	Baseline Controller Tracking Error (rad)		BRNN + Feedback Tracking Error (rad)	
	ℓ_2	ℓ_∞	ℓ_2	ℓ_∞
1	1.4939	0.1826	1.2083	0.0954
2	3.0560	0.3547	1.2055	0.0797
3	0.6279	0.0196	0.3023	0.0599
4	5.4097	0.6381	2.5719	0.1529
5	0.2238	0.0265	0.1946	0.0140
6	1.4105	0.1241	1.3121	0.0932
7	0.7065	0.0592	0.6404	0.0344

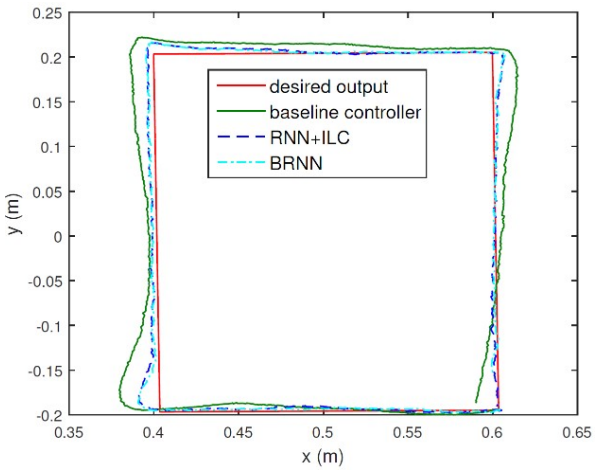


Fig. 7: Comparison of tracking performance without and with the RNN/BRNN feedforward controllers for a Cartesian square trajectory in $x-y$ plane with z constant.

unknown dynamics through feedforward compensation by recurrent neural networks. The first approach is to implement iterative learning control with the trained RNN which approximates the forward dynamics of the robot for a specified desired trajectory offline. The second one is to filter the desired trajectory by the BRNN which emulates the dynamical system inversion. We first demonstrated the results of two approaches for the manipulator trajectory tracking control when a desired trajectory is entirely given in advance. We also showed the results of tracking a human teleoperated random joint trajectory which is generated online using the BRNN feedforward controller producing real-time compensation control input. By comparing with a baseline feedback controller, the effectiveness of our approaches is verified.

Future work includes the development of an adaptive controller with generalized basis network that can approximate multiple dynamical systems via online parameters tuning. Also, it is worthwhile to explore the transferability of the trained model to another Baxter robot. The tracking performance may be further improved by combining the feedforward control inputs produced by RNNs with more sophisticated feedback control laws instead of a pure proportional control as described here.

REFERENCES

[1] Z. Qu and D. M. Dawson, *Robust tracking control of robot manipulators*. IEEE press, 1995.

[2] J.-J. E. Slotine and W. Li, "On the adaptive control of robot manipulators," *The international journal of robotics research*, vol. 6, no. 3, pp. 49–59, 1987.

[3] S. Arimoto, "Learning control theory for robotic motion," *International Journal of Adaptive Control and Signal Processing*, vol. 4, no. 6, pp. 543–564, 1990.

[4] H. Miyamoto, M. Kawato, T. Setoyama, and R. Suzuki, "Feedback-error-learning neural network for trajectory control of a robotic manipulator," *Neural Networks*, vol. 1, no. 3, pp. 251 – 265, 1988.

[5] B. Brogliato, R. Ortega, and R. Lozano, "Global tracking controllers for flexible-joint manipulators: a comparative study," *Automatica*, vol. 31, no. 7, pp. 941–956, 1995.

[6] D. Wang, "A simple iterative learning controller for manipulators with flexible joints," *Automatica*, vol. 31, no. 9, pp. 1341–1344, 1995.

[7] T. H. Lee and C. J. Harris, *Adaptive neural network control of robotic manipulators*. World Scientific, 1998, vol. 19.

[8] O. Ogunmolu, X. Gu, S. Jiang, and N. Gans, "Nonlinear Systems Identification Using Deep Dynamic Neural Networks," *arXiv e-prints*, p. arXiv:1610.01439, Oct 2016.

[9] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Trans. Signal Processing*, vol. 45, pp. 2673–2681, 1997.

[10] Y. Jiang, C. Yang, J. Na, G. Li, Y. Li, and J. Zhong, "A brief review of neural networks based learning and control and their applications for robots," *Complexity*, vol. 2017, pp. 1–14, 10 2017.

[11] L. Jin, S. Li, J. Yu, and J. He, "Robot manipulator control using neural networks: A survey," *Neurocomputing*, vol. 285, pp. 23 – 34, 2018.

[12] S. J. Yoo, J. B. Park, and Y. H. Choi, "Adaptive output feedback control of flexible-joint robots using neural networks: Dynamic surface design approach," *IEEE Transactions on Neural Networks*, vol. 19, no. 10, pp. 1712–1726, Oct 2008.

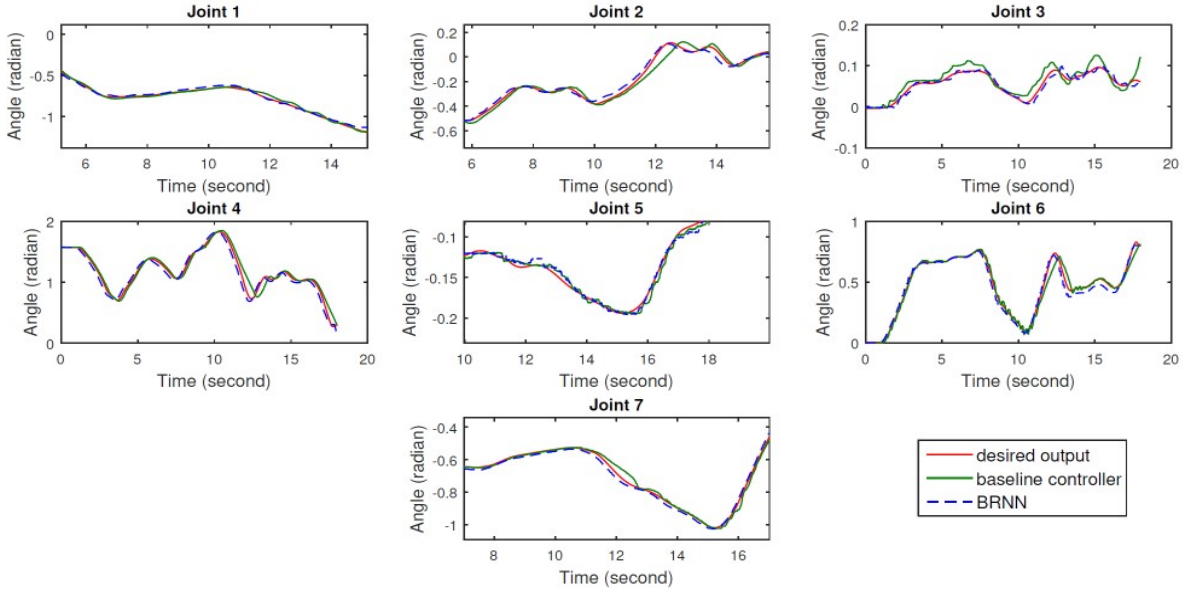


Fig. 8: Comparison of tracking performance without and with the BRNN feedforward controller of a user teleoperated random joint trajectories. The plots of joint 1, 2, 5 and 7 are zoomed in to better visualize the difference.

[13] J. H. Perez-Cruz, I. Chairez, J. de Jesus Rubio, and J. Pacheco, "Identification and control of class of non-linear systems with non-symmetric deadzone using recurrent neural networks," *IET Control Theory Applications*, vol. 8, no. 3, pp. 183–192, Feb 2014.

[14] W. He, Y. Dong, and C. Sun, "Adaptive neural impedance control of a robotic manipulator with input saturation," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 46, no. 3, pp. 334–344, March 2016.

[15] M. Wang, H. Ye, and Z. Chen, "Neural learning control of flexible joint manipulator with predefined tracking performance and application to baxter robot," *Complexity*, vol. 2017, pp. 7 683 785:1–7 683 785:14, 2017.

[16] W. He, Z. Yan, Y. Sun, Y. Ou, and C. Sun, "Neural-learning-based control for a constrained robotic manipulator with flexible joints," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 12, pp. 5993–6003, Dec 2018.

[17] C. Yang, X. Wang, L. Cheng, and H. Ma, "Neural-learning-based telerobot control with guaranteed performance," *IEEE Transactions on Cybernetics*, vol. 47, no. 10, pp. 3148–3159, Oct 2017.

[18] H. A. Talebi, R. V. Patel, and K. Khorasani, "Inverse dynamics control of flexible-link manipulators using neural networks," in *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*, vol. 1, May 1998, pp. 806–811 vol.1.

[19] Q. Li, J. Qian, Z. Zhu, X. Bao, M. K. Helwa, and A. P. Schoellig, "Deep neural networks for improved, impromptu trajectory tracking of quadrotors," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 5183–5189.

[20] R. Calandra, S. Ivaldi, M. P. Deisenroth, E. Rueckert, and J. Peters, "Learning inverse dynamics models with contacts," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 3186–3191.

[21] E. Rueckert, M. Nakatenus, S. Tosatto, and J. Peters, "Learning inverse dynamics models in O(n) time with LSTM networks," in *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, Nov 2017, pp. 811–816.

[22] S. Chen and J. T. Wen, "Industrial Robot Trajectory Tracking Using Multi-Layer Neural Networks Trained by Iterative Learning Control," *arXiv e-prints*, p. arXiv:1903.00082, Feb 2019.

[23] B. D. MAYNE, "A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems," *International Journal of Control*, vol. 3, no. 1, pp. 85–95, 1966.

[24] T. Yoshikawa, "Manipulability of robotic mechanisms," *The International Journal of Robotics Research*, vol. 4, no. 2, pp. 3–9, 1985.

[25] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[26] B. Potsaid, J. T. Wen, M. Unrath, D. Watt, and M. Alpay, "High performance motion tracking control for electronic manufacturing," *Journal of Dynamic Systems Measurement and Control*, vol. 129, p. 767776, 11 2007.

[27] S. Chen, Y. C. Peng, J. Wason, J. Cui, G. Saunders, S. Nath, and J. T. Wen, "Software framework for robot-assisted large structure assembly," in *the ASME 2018 13th International Manufacturing Science and Engineering Conference*, June 2018.

[28] J. D. Wason and J. Wen, "Robot raconteur: A communication architecture and library for robotic and automation systems," in *IEEE International Conference on Automation Science and Engineering*, 08 2011, pp. 761–766.