Improvement of trajectory tracking accuracy in serial robotic arms based on feed-forward neural networks per servomotor for position control

Toussaint Baptiste[1], **Raison Maxime**[2],

[3] Department of mechanical engineering
Polytechnique Montreal
2900, Edouard-Montpetit, Montréal, Canada
[baptiste.toussaint, maxime.raison]@polymtl.ca

ABSTRACT

Fast and accurate motion control can be important with robotic arms performing large movements. However a compromise should be made between the reduction of cycle time and tracking accuracy. This is due to the inability of joint servomotors controllers to address nonlinearities and uncertainties of the dynamic models of the robotic arms.

In this paper, we consider an architecture which does not requires any prior information about the dynamic model. It consists in learning the dynamic response of a reference position PD controller by using a neural network (NN) system in an outer loop to predict/correct the tracking errors of this PD. This paper aims at determining the optimal architecture for the NN system and identify the best practices to maximize its performances. Especially, we compare two different methods for the NN system. Either one NN per servomotor or a single global NN for the whole robot. To evaluate these two approaches, we use both simulated and real robotic arms (3D-printed).

Keywords: robotics, control, machine learning, neural networks, trajectory tracking

1 INTRODUCTION

The transfer of robotics into everyday life is currently accelerating, with 20 million robots expected to be in use worldwide by 2030 [1]. In this field, fast and accurate motion control is usually required with robotic arms performing large movements.

Control methods for a successful tracking of such trajectories by robot arms is a well-studied problem [2]. The most-current control architecture consists in a joint torque control using a linear controller such as a traditional proportional–integral–derivative (PID) controller. To compensate the response time of the controller, a feedforward compensation using the dynamic model of the system is usually added. However yet, for most robotics arms, an accurate dynamic model can be difficult to obtain, due to the nonlinearities (e.g. friction) and uncertainties (e.g. inertial parameters) in these dynamic systems [3] which makes it challenging to efficiently implement on robot arms. Thus, the inability of joint servo controllers to address these nonlinearities and uncertainties can lead to a degradation of a trajectory tracking accuracy [4]. In general, the chosen solution is to find a compromise between the reduction of cycle time and improvement of tracking accuracy for industrial robots [5]. This situation is currently accentuated with the attempts to develop 3D printed low-cost robotic arms, which usually have high friction and cheaper motors.

In this case, it is either possible to use a simplified model [6] or machine learning methods [3]. Notably, iterative methods such as the iterative learning control, are based on the repetition of tasks and allow efficient trajectory tracking without any information on the dynamic model [7][8]. However, the learned representation is not transferable between different trajectories.

Recently, controllers based on neural networks have attracted attention with its potential ability

to generalize beyond the training set [5]. For trajectory tracking especially, the power of approximation of NNs has been used in controllers to learn either forward or inverse dynamics of the system. [9][10][11][12].

To approximate the systems forward dynamics, several NNs such as recurrent neural networks (RNNs), feedforward networks or radial basis function neural network (RBF) have been used [13][14][15][16][17]. Most of these works are operating at a torque level control, which usually requires many parameters for the NNs – in the order of 10^5 in [18], for a 7DOF robot arm – and several neural networks to approximate different elements of the dynamic model. In [4], the learning of the direct dynamic model is made at the joint position control level combining a PID controller with a RNN with only around 10^3 parameters, directly as one integrated model.

NNs have also been used to learn inverse dynamic models and included as a feedforward controller for dynamic compensation [19]. It already has been shown that NNs - especially RNNs and LSTM - can outperform other techniques such as Gaussian Processes to learn inverse dynamics when there are sufficient training data [20][21]. In [4] and [22], a NN architecture is used in an outer-loop on either a quadrotor with a deep neural network or a 7DOF robot arm with an RNN. These papers highlighted three main advantages of using NNs in an outer control loop summarized in [4]. First, NNs architectures can be applied to various systems with complex dynamics while ensuring its stability. Second, this approach doesn't need any prior information on the system and can be applied to unseen trajectories without any adaptation process. Third, with a good training process, this approach demonstrates good accuracy in trajectory tracking, while being computationally efficient with a small model.

In [5], the inverse dynamics learning process of an industrial ABB robot is done in simulation by implementing iterative learning control (ILC) for many trajectories to collect data for the NN on the real system. However, this technique requires a good dynamics simulator of the system, which is rarely available for non-industrial robots, because a lot of data is needed to train the model. In this paper, unlike the previous article which used only one NN for the whole system, the authors mentioned that on their studied ABB robot, the joints dynamics are decoupled – i.e. the movements of the joints do not impact the dynamics of other joints – so it's possible to train 6 different NNs, with each one approximating the dynamical inversion for one joint.

In [12], we proposed a control architecture for high-speed trajectory tracking using the advantages of both PD – stable and easy to use – and neural networks – efficient to estimate the dynamics of systems. Contrary to [4], we used one NN per joint, instead of one global NN for the whole robot arm. The architecture – illustrated in Figure 1 – consists in learning the dynamic response of a reference PD controller by using a NN and use it in an outer loop as a feedforward compensation to predict/correct the errors of this PD. This method allows reducing the errors in position and speed during the trajectory tracking, even without any prior information about the dynamic model [12]. However, no detail was provided to take maximum benefits of this method, forcing one to experimentally choose learning parameters. Especially, there is no guideline about the best architecture for the learning system when used as a feedforward controller – either a unique NN for the whole system or one smaller NN for each joint – as well as the conditions of performance for this method. The objective of this study is to determine the optimal architecture for the learning system and identify the best practices to maximize its performances. To meet this objective, we use both real and simulated robotic arms in various configurations.

The rest of the paper is organized as follows. Section II presents the problem statement and the methodology. Section III and IV show the results of the study and the discussions. Finally, conclusions are summarized in Section V.

2 METHODOLOGY

The equations of the simulated system are generated with the Robotran software. The used formalism is the semi-explicit formulation, also referred as the direct dynamic model in the literature.

$$\boldsymbol{M}(\boldsymbol{q},\boldsymbol{\delta})\,\ddot{\boldsymbol{q}} + \boldsymbol{c}(\boldsymbol{q},\dot{\boldsymbol{q}},\boldsymbol{\delta},\boldsymbol{frc},\boldsymbol{trq},\boldsymbol{g}) = \boldsymbol{\tau}(\boldsymbol{q},\dot{\boldsymbol{q}}) \qquad (\mathbf{1})$$

With $\boldsymbol{M}[n*n]$ the symmetric generalized mass matrix of the system, $\boldsymbol{c}\,[n*1]$ the non-linear dynamic vector which contains the gyroscopic, centrifugal and gravity terms, as well as the contributions of external forces and torques, $\boldsymbol{q}[n*1]$ the relative generalized coordinates, $\boldsymbol{\delta}[10n*1]$ which contains the dynamics parameters of the system - masses, center of masses, inertia -, and $\boldsymbol{\tau}[n*1]$ the generalized joint forces. For more details, you can read the Robotran documentation [23].

To be as realistic as possible, the effect of the motors in the dynamic model is considered with the inertia of the rotor of each motor and a viscous friction, velocity-dependant [24]. For simplicity, we neglect the nonlinear friction torques given in this article. Thus, we can write the model of the motor as follow:

$$\boldsymbol{J_m\ddot{q}_m + D_m\dot{q}_m = \tau_m - R\tau} \quad \textbf{(2)}$$

With $\boldsymbol{\tau}$ corresponding to the generalized joint forces of equation **(1)**, $\boldsymbol{\dot{q}_m}[n*1]$ and $\boldsymbol{\ddot{q}_m}[n*1]$ the velocities and accelerations of the n motors, $\boldsymbol{J_m}$, $\boldsymbol{D_m}$ and $\boldsymbol{R}$ three $[n*n]$ diagonal matrices corresponding respectively to the inertia moments, the viscous friction coefficients of the motors and the gear reduction ratios. Finally, $\boldsymbol{\tau_m}[n*n]$ is the vector of torques supplied by the motors. Based on [24], we use the equation **(1)** to replace $\boldsymbol{\tau}$ in equation **(2)** and the relation $\boldsymbol{q = Rq_m}$ to combine the equations to obtain the whole model of the simulated arm:

$$\boldsymbol{J_m\ddot{q}_m + D_m\dot{q}_m = \tau_m - R(M\ddot{q} + c)} \quad \textbf{(3)}$$

$$\boldsymbol{(J_m + RMR)\ddot{q}_m + D_m\dot{q}_m + Rc = \tau_m} \quad \textbf{(4)}$$

$$\boldsymbol{\ddot{q} = R\ddot{q}_m = R[(J_m + RMR)^{-1}(\tau_m - D_m\dot{q}_m - Rc)]} \quad \textbf{(5)}$$

We consider an n-dof robot arm with a joint servocontrol with input $\boldsymbol{q_c} \in R^n$, the joint position command and the output $\boldsymbol{q} \in R^n$, the measured joint position. Our goal is to find a feedforward compensation such that for a given desired trajectory $\boldsymbol{q_d}$, the corrected input command $\boldsymbol{q_c}$ would result in the perfect tracking of this desired trajectory: $\boldsymbol{q_c \rightarrow q = q_d}$. The architecture of the control system is presented in Figure 1.

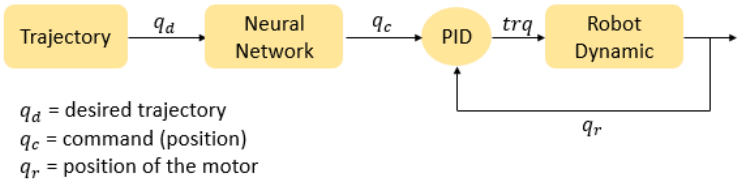

Figure 1. *Control architecture for high-speed trajectory tracking [4][12]*

To obtain the required results to compare the neural networks architectures, we used two 3D-printed robot arms, with 3 and 5 degrees of freedom (DOF) and a simulated model of a 3DOF robotic arm. The experimental results support and enhance the simulated results by ensuring that they reflect the reality. We also assume that the internal dynamics of the robotic arms is deterministic and repeatable.

Using both real robotic arms and simulated models, we used the following method:

- Choose a configuration to test. Either the real or the simulated robot arm, the number and the position of degrees of freedom (DOF), the duration of the trajectories and the configuration of the NN system. In the rest of the paper, the use of one independent NN for each servomotor is designed as "method 1". The use of a unique global NN for all the system is designed as "method 2". Especially, more parameters are available for the simulated model: the PD parameters, the reduction ratio of the servomotors and the noise magnitude.
- Build a dataset of trajectories that represents the whole workspace of the arm by using a simple PD controller on each joint. Each dataset is composed of five variables for each of the five motors: position $\boldsymbol{q}$ and velocity $\boldsymbol{\dot{q}}$ of the motor at two consecutive timestamps – $\boldsymbol{q}(t)$, $\boldsymbol{q}(t+\delta)$, $\boldsymbol{\dot{q}}(t)$, $\boldsymbol{\dot{q}}(t+\delta)$ – and the command sent between these two timestamps $\boldsymbol{u}(t)$.

- Train the NN architecture to learn the dynamic model through the response of the PD controller. Giving the NN a lot of data of trajectories, it will learn which command it has to send to each motor to go from a state – $\boldsymbol{q}(t)$, $\dot{\boldsymbol{q}}(t)$ – to the next – $\boldsymbol{q}(t+\delta)$, $\dot{\boldsymbol{q}}(t+\delta)$ – and in an implicit way, the dynamic model of the robotic arm.
- Generate a new set of commands for the motors, which considers the response time of the controller to send the required series of commands to each motor so that these ones execute the desired trajectory.
- Store the tracking errors for n test trajectories ($n = 40$, chosen number to have low variation in the results).
- Repeat steps 3 to 5 for a different number of training trajectories and different NN architectures to compare their effectiveness to track the test trajectories with accuracy.

The evaluation of the accuracy of the control system for trajectory tracking has been done by comparing the root-mean-square (RMS) errors all along the trajectory: $e = \sqrt{mean(\sum(\boldsymbol{q} - \boldsymbol{q_d})}$

3 RESULTS

3.1 Results with the simulated model without reduction ratio

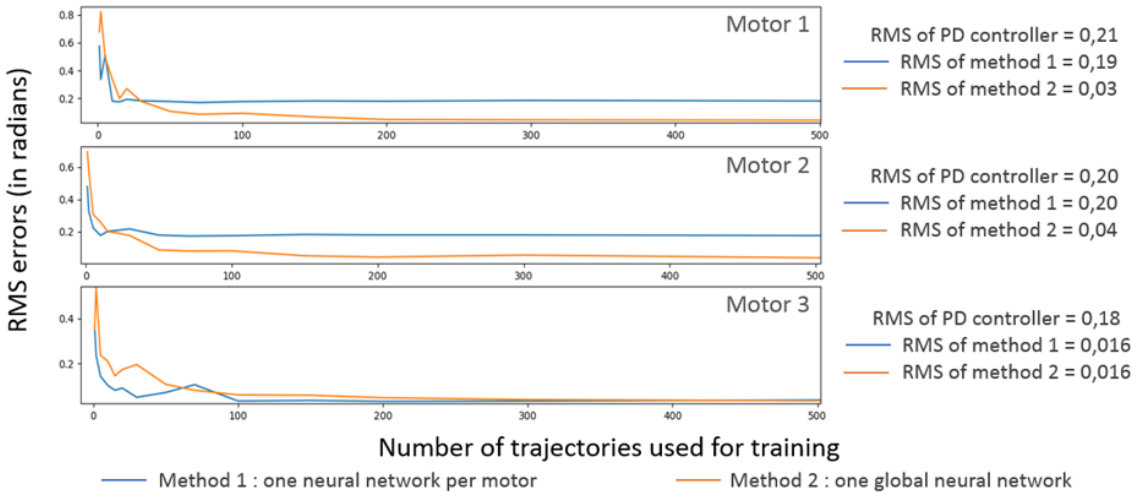

Figure 2. *Learning curve with both methods 1 and 2 without reduction ratio*

Figure 2 illustrates a typical learning curve obtained with the simulated model using both methods. We observe that the method 1 reaches its best performances with on average 3 times less data than method 2 (10 to 100 trajectories needed for the first one, versus 100 to 250 for the last approach).

The learning curves for motors 1 and 2 highlight a difference of the tracking accuracy of the two studied architectures when more than 100 trajectories are available. The method 1 does not exceed performances of the PD controller with an RMS around 0.2 radians whereas the second method manages to get an RMS error less than 0.04 radians – or 2.3 degrees. On the opposite, the learning curve for the third motor shows similar performances for the trajectory tracking between the two methods with an RMS error of 0.016 radians – less than 1 degree – which represent a 91% diminution of tracking errors compared to the PD controller.

3.2 Results with the simulated model with reduction ratio

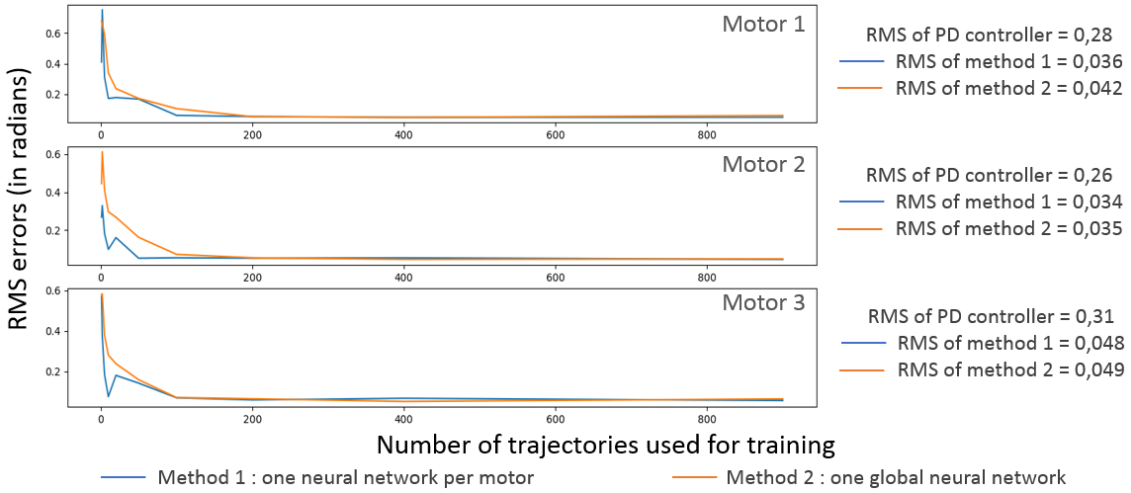

Figure 3. *Learning curve with both methods 1 and 2 with a reduction ratio R=0.01*

Figure 3 presents the learning curves for the two architectures with a reduction ratio R of 0.01. This time, the performances are similar for all the three motors with an RMS error under 0.05 radians – 3 degrees. As for the Figure 2, we still observe that the method 1 reaches its best performance with less data than method 2 (30 to 100 trajectories needed, versus 100 to 200 for the method 2).

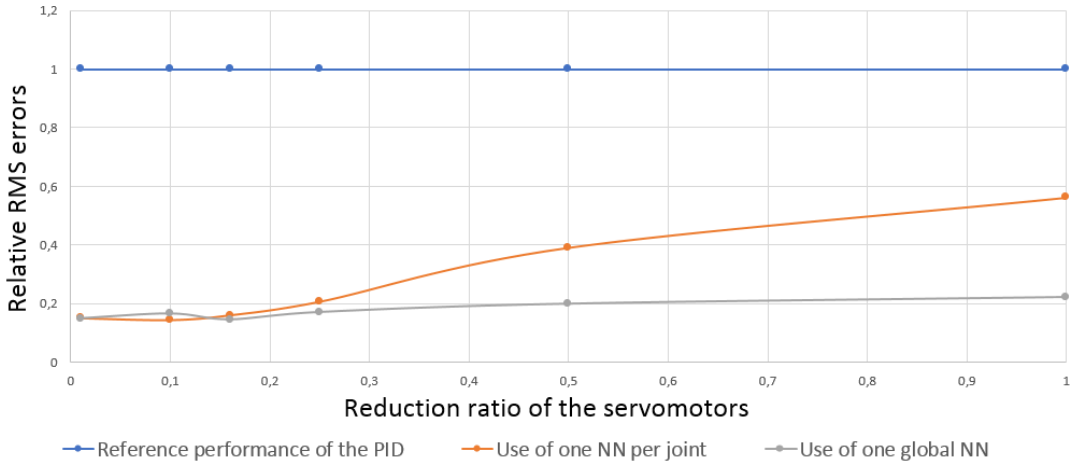

Figure 4. *Relative performances of the neural networks compared to the reference PD controller as a function of the reduction ratio*

Figure 4 presents the average relative performances for both methods, with the reference PD controller in function of the reduction ratio of the servomotors. It shows similar performances for both methods for a ratio below 0.15 with a relative RMS errors of 0.15 compared to the PD controller and a progressive degradation of performances of the method 1 when the ratio increases. For a reduction ratio of 1 – i.e no reduction – the method 2 demonstrates better accuracy than method 1 with a relative RMS errors of 0.22 versus 0.56 compared to the PD controller, i.e 2.5 times better.

Figure 5 shows relative performances of both methods compared to the reference PD controller depending on its accuracy – or parameters – for two different ratios between the frequency of the dynamic integration and the frequency of the neural network controllers. When the step time corresponds to the Δt between two commands from the neural network, we can note that the RMS errors of the NN are almost proportional to those of the PD controller, with a diminution of around 90% of its tracking errors. When the frequency of the dynamic integration is ten times the frequency of the NN, the diminution of the tracking errors with respect to the PD controller is between 80% and 85%, i.e the RMS errors are 2 times more important than when the step time are the same. Moreover, when the reference PD is more aggressive – i.e manage to track the trajectories with an RMS error less than 3 degrees – we can see that the performances of the NN

are no longer proportional to those of the PD. But the feedforward neural network is more efficient when the frequencies of both neural network system and dynamic integration matches, with the NN system struggling to reduce the error and has an RMS error under 0.5 degree when the two frequencies are different.

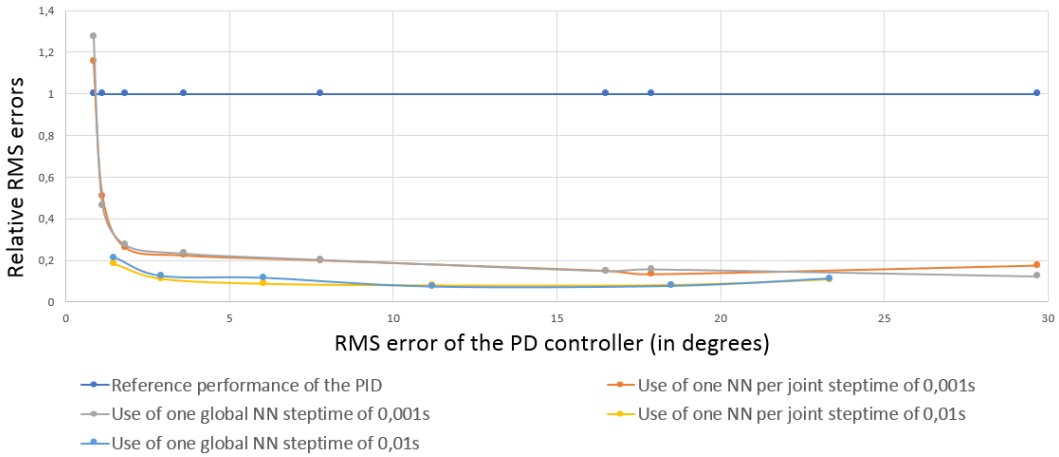

Figure 5. *Relative performances of the neural networks to the reference PD controller as a function of the accuracy of the PD controller and the step time of the direct dynamic integration*

Table 1. *Comparison of the median relative RMS errors in function of the size of the NNs*

Method	One NN for each servomotor						One global NN for the whole system					
Size of the NN	1	3	16	2x3	2x16	3x16	1	3	16	2x3	2x16	3x16
Relative RMS	1	1.057	0.978	0.935	0.992	0.996	24.26	0.970	1.014	1.027	0.989	1.063

Table 1 shows a comparison of the median relative RMS error for different sizes of neural networks for both methods 1 and 2, tested for different PID parameters. All tested architectures which have at least the same number of neurons as the number of associated motors on each layer, have a relative median accuracy between 0.935 and 1.063 compared to the smallest neural network which is less than 10% difference on the accuracy. On the contrary, when the number of neurons on each layer is smaller than the number of associated motors, the NN does not manage to converge.

Figure 6 shows relative performances of both methods compared to the reference PD controller depending on the noise present in the data used for training. When there is no noise, we still have 90% improvement of trajectory tracking compared to the reference PD controller with an RMS error of 0.86 degrees. First, we can observe performances and a diminution of the accuracy of the system which is (very !) similar for both methods 1 and 2. Moreover, when the noise is greater than 0.5 degrees – which is already really important, for example the noise of our real system is estimated to around 0.2 degrees (one step of the encoder) – it doesn't seem to influence much on the performance of the neural networks with an RMS errors between 1.40 and 1.65 degrees, which is still 5 times less than the RMS errors of the PD controller (although it is 1.8 times larger than when there is no noise in the data.

Finally, an interesting result (not visible in this figure) is that when the noise is increasing – more than 0.5 degree – NN with more neurons seem to have better performances with up to 20% improvement of the accuracy compared to the small neural networks.

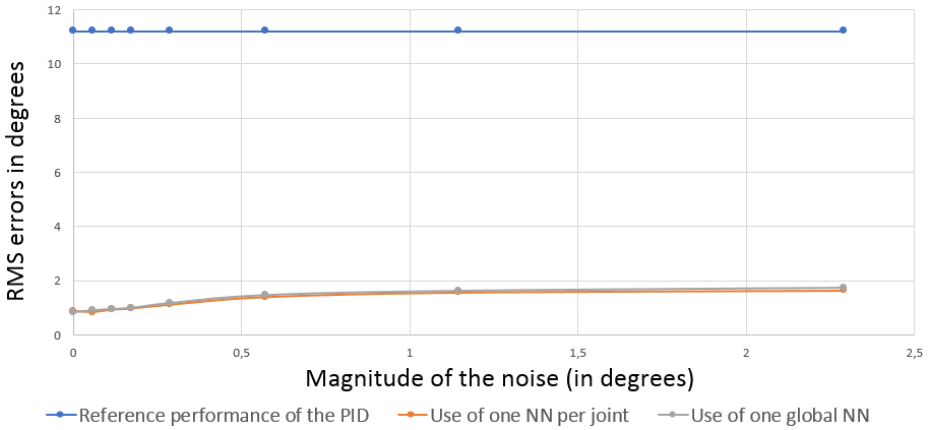

Figure 6. *Relative performances of the neural networks to the reference PD controller as a function of the noise magnitude in the training data*

3.3 Experimental results

Figure 7 compares the accuracy – i.e. the RMS error in trajectory tracking - for three different configurations with real robotic arms, between the two studied NN architectures: A. fast trajectories and B. intermediate trajectories for a 3 DOF robotic arm and C. slow trajectories with a 5-DOF serial robotic arm.

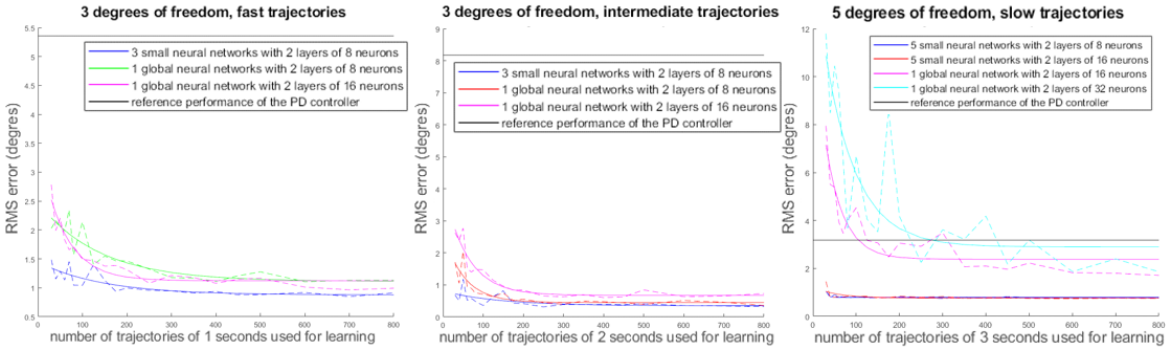

Fig 7A *Fig 7B* *Fig 7C*

Figure 7. *Performance of trajectory tracking with various NNs, for: A. 3 DOF robotic arm, fast trajectories; B. 3 DOF robotic arm, intermediate trajectories; C. 5 DOF robotic arm, slow trajectories.*

Figure 7 shows that method 1 allows to get an accuracy close to those obtained with all training data with around 100 whereas method 2 requires around 300 trajectories to learn the dynamic response of the PD controller, regardless to the used robot arm and the speed of the trajectories.

Table 2. *Comparison of the trajectory tracking accuracy (RMS error) of NNs for trajectory tracking on two real robot arms*

Trajectory velocities (and DOF)	**RMS error with the reference PD controller**	**RMS error with a global NN**	**RMS error with one NN per servomotor**
	In degrees (reduction of tracking errors compared to the reference controller in %)		
Fast (3 DOF)	5.36	1.12 (79%)	0.88 (84%)
Intermediate (3 DOF)	8.17	0.45 (94%)	0.33 (96%)
Slow (5 DOF)	3.18	1.71 (46%)	0.76 (76%)

Table. 2 summarizes the numerical values of RMS errors among the *n* test trajectories and the rate of reduction of the errors compared with the reference PD controller. Especially, the first column shows at least 75% reduction of tracking errors compared to this reference controller, and an RMS error under 1 degree, whatever the configuration when one NN is used for each servomotor (method 1).

4 DISCUSSION

The first main result highlighted by the study of the simulated model without reduction ratio in Figure 2 is that method 1 – with one NN for each motor – works only when the joint dynamics is not coupled on the other joints dynamics – i.e that the movements of other joints does not have any – or almost any – influence on the dynamics of the joint. In fact, the motor three at the end of the arm is not influenced by the movements of the first two motors. That explains why method 1 works well for this last motor with a diminution of 91% of the tracking errors whereas it does not exceed performances of the PD controller for the first two motors with an RMS error of 0.2 radians equal to that of the PD controller.

The second main result highlighted in Figure 3 is the interest of method 1 when the dynamics of each joint is not coupled with the others. In particular, the decoupling of the joint dynamics is observed when the term RMR of equation **(2)** becomes negligible compared to the inertia of the servomotor J_m. Figure 4 shows it appends for a reduction ratio below 0.15, which corresponds to $J_m > 50 \cdot RMR$. In this situation, method 1 requires less data than method 2 to perform the learning process: 10 to 100 trajectories versus 100 to 250 trajectories. Consequently, method 1 outperforms method 2 when the training dataset is composed by data from less than 100 trajectories and demonstrates similar accuracy when more training data is available with RMS errors between 0.034 and 0.048, similar to the RMS errors of method 2, between 0.035 and 0.049. On the contrary, Figure 4 also shows the interest of using method 2 for systems with a coupled dynamics with a better accuracy when the reduction ratio is 1 with RMS errors 2.5 smaller than method 1. In fact, the global NN can learn the dependencies between the different joints and keeps an accuracy close to the one obtained with a decoupled dynamics.

Figure 6 demonstrates two important facts. First, it shows that when the step time of the data used for training corresponds to the PD frequency, the RMS errors are 2 times less compared to the situations where the step time of the data used for training is 10 times smaller than the PD frequency. The conclusion is that to get the best possible accuracy for both methods, the step time of the data used for training must corresponds to the PD frequency. Second, the relative accuracy of the NN compared to the reference controller is stable whatever the performances of the PD controller. A conclusion is that the better accurate the PD controller, the better accurate the feedforward NN control system.

Table 1 shows that a necessary condition to the use of method 2 is to have at least a number of neurons equal to the number of motors in the system on each layer in order to converge. Furthermore, the comparison of RMS trajectory tracking errors has shown that when this condition is satisfied, there is no significant difference on the accuracy – less than 10 percent – between all sizes of NNs. This result encourages the use of the smallest possible neural networks to limit the number of parameters. For example, the smallest NN presented in Figure 3 is as accurate as the other NNs and only requires 15 parameters for the whole 3 DOF robot arm – 5 parameters per motor – which is around 67 times less than the system in [4] and 6700 times less than the architecture in [18] which operates at the torque level. This allows a proportional diminution of the calculation time which could be used to increase the frequency and the performances of the control system.

Finally, Figure 6 shows that a big magnitude of noise in the data – more than 0.5 degree – does not prevent the NN to improve the trajectory tracking, with RMS errors between 1.4 and 1.65 degrees, 5 times lower than the PD controller, Even if the NN seems to be able to manage this noise during the training, Figure 6 also shows that the best results are obtained when the noise is

zero – or almost zero with a RMS error of 0.86, 1.8 times lower than when the noise is more than 0.5 degree. Thus, even if it is not critical, a good practice is to get the best possible encoder definition to limit the noise in the data.

Figure.7 and Table.2 recall the interest of the use of a NN as a feed-forward controller, by comparing the accuracy with a standard PD controller on a real 3D printed robotic arm. Both methods 1 and 2 allow a good trajectory tracking accuracy for the real robot arms with an RMS error under 2 degrees for all tested configurations. These errors are consistent with the errors of the simulated model. Especially, Figures. 7A and 7B shows that the improvement is even more important with the fastest trajectories i.e. where the response time of the PD controller has a greater influence. Further, in Table.1 the use of method 1 demonstrates at least a 75% reduction of errors for all tested configurations. This result corresponds to the simulation results of Figures 4 and 5 with an 80% to 90% reduction of tracking errors.

Experimental results also show that even for the first two motors, the method 2 doesn't outperform the accuracy of the method 1. These results are consistent with the fact that the dynamics of the joints is decoupled, i.e. the dynamics of the servomotors prevails on the dynamics of the robotic arm due to the real reduction ratio of the servomotors (1/540).

Finally, the experimental results confirm the main results of the simulated model for a real system: the method 1 requires around 3 times less data than the method 2 to learn the dynamics.

5 CONCLUSION

In this paper, we compared two methods for trajectory tracking control through a neural network used as a feedforward controller for position control: the use of one neural network for each joint of the robot arm or the use of one global neural network for the whole system. We first demonstrated the interest of the use of a neural network as a feedforward controller with various configurations which allows an accurate trajectory tracking – RMS errors always under 3 degrees – without any prior information about the dynamic model of the system. Especially, the objective was to determine the optimal architecture for the NN and identify the best practices to maximize its performances. Thus, we stated that when the dynamics of the joints of the robotic arm is decoupled, the use of one neural network for each joint is advised because it requires less data to learn the dynamics. We also stated about four practices to get the best possible accuracy: 1. optimizing the parameters of the PD controller before training, 2. using a step time to construct the training data equal to the step time of the PD controller, 3. limiting the noise by using a high resolution encoder, 4. privileging the use of one small neural network per joint when the dynamics of the joint is decoupled - i.e when the servomotors have a small reduction ratio – and using small neural networks with few neurons to limit the number of parameters and to reduce the calculation time.

REFERENCES

[1] Taylor, C. Robots could take over 20 million jobs by 2030, study claims. https://www.cnbc.com/2019/06/26/robots-could-take-over-20-million-jobs-by-2030-study-claims.html; last visit: 24th February 2023.

[2] Zhihong, M., Palaniswami, M. (1994). Robust tracking control for rigid robotic manipulators. IEEE Transactions on Automatic Control, 39(1), 154-159.

[3] Koç, O., Maeda, G., & Peters, J. (2019). Optimizing the execution of dynamic robot movements with learning control. IEEE Transactions on Robotics, 35(4), 909-924.

[4] Chen, S., Wen, J. T. (2019, November). Neural-learning trajectory tracking control of flexible-joint robot manipulators with unknown dynamics. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (pp. 128-135).

[5] Chen, S., Wen, J. T. (2021). Industrial robot trajectory tracking control using multi-layer neural networks trained by iterative learning control. Robotics, 10(1), 50.

[6] Spong, M. A. R. K. W., Khorasani, K., & Kokotovic, P. (1987). An integral manifold approach to the feedback control of flexible joint robots. IEEE Journal on Robotics and Automation, 3(4), 291-300.

[7] Arimoto, S. (1990). Learning control theory for robotic motion. International Journal of Adaptive Control and Signal Processing, 4(6), 543-564.

[8] Norrlof, M. (2002). An adaptive iterative learning control algorithm with experiments on an industrial robot. IEEE Transactions on robotics and automation, 18(2), 245-251.

[9] Wang, M., Ye, H., Chen, Z. (2017). Neural learning control of flexible joint manipulator with predefined tracking performance and application to baxter robot. Complexity, 2017.

[10]He, W., Yan, Z., Sun, Y., Ou, Y., & Sun, C. (2018). Neural-learning-based control for a constrained robotic manipulator with flexible joints. IEEE transactions on neural networks and learning systems, 29(12), 5993-6003.

[11]Chen, S., & Wen, J. T. (2020, May). Adaptive neural trajectory tracking control for flexible-joint robots with online learning. In IEEE International Conference on Robotics and Automation (ICRA) (pp. 2358-2364).

[12]Toussaint, B., Raison, (2021) M. High-speed trajectory tracking on robotic arm by learning the dynamic response of the pid controller with a neural network. Book of Abstracts of the 10th ECCOMAS Thematic Conference on Multibody Dynamics, pages 216–217.

[13]Yoo, S. J., Park, J. B., & Choi, Y. H. (2008). Adaptive output feedback control of flexible-joint robots using neural networks: dynamic surface design approach. IEEE Transactions on Neural Networks, 19(10), 1712-1726.

[14]Pérez-Cruz, J. H., Chairez, I., de Jesús Rubio, J., & Pacheco, J. (2014). Identification and control of class of non-linear systems with non-symmetric deadzone using recurrent neural networks. IET Control Theory & Applications, 8(3), 183-192.

[15]He, W., Dong, Y., & Sun, C. (2015). Adaptive neural impedance control of a robotic manipulator with input saturation. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 46(3), 334-344.

[16]Wang, M., Ye, H., & Chen, Z. (2017). Neural learning control of flexible joint manipulator with predefined tracking performance and application to baxter robot. Complexity.

[17]He, W., Yan, Z., Sun, Y., Ou, Y., & Sun, C. (2018). Neural-learning-based control for a constrained robotic manipulator with flexible joints. IEEE transactions on neural networks and learning systems, 29(12), 5993-6003.

[18]Yang, C., Wang, X., Cheng, L., & Ma, H. (2016). Neural-learning-based telerobot control with guaranteed performance. IEEE transactions on cybernetics, 47(10), 3148-3159.

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

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

[21]Rueckert, E., Nakatenus, M., Tosatto, S., & Peters, J. (2017, November). Learning inverse dynamics models in o (n) time with lstm networks. In IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids) (pp. 811-816).

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

[23]https://robotran-doc.git-page.immc.ucl.ac.be/RobotranBasic/Robotran_basics.pdf
last visit: 28th February 2023.

[24]Rocco, P. (1996). Stability of PID control for industrial robot arms. IEEE transactions on robotics and automation, 12(4), 606-614.