applied sciences

MDPI

Article

Design of Minimal Model-Free Control Structure for Fast Trajectory Tracking of Robotic Arms

Baptiste Toussaint and Maxime Raison *

Department of Mechanical Engineering, Polytechnique Montreal, Montreal, QC H3T 1J4, Canada; baptiste.toussaint@polymtl.ca
* Correspondence: maxime.raison@polymtl.ca

Featured Application: Following straightforward guidelines, feedforward neural networks enable to design an accurate model-free control structure for the fast trajectory tracking of robotic arms.

Abstract: This paper designs a minimal neural network (NN)-based model-free control structure for the fast, accurate trajectory tracking of robotic arms, crucial for large movements, velocities, and accelerations. Trajectory tracking requires an accurate dynamic model or aggressive feedback. However, such models are hard to obtain due to nonlinearities and uncertainties, especially in low-cost, 3D-printed robotic arms. A recently proposed model-free architecture has used an NN for the dynamic compensation of a proportional derivative controller, but the minimal requirements and optimal conditions remain unclear, leading to overly complex architectures. This study aims to identify these requirements and design a minimal NN-based model-free control structure for trajectory tracking. Two architectures are compared, one NN per joint (INN) and one global NN (GNN), each tested on two serial robotic arms in simulations and real scenarios. The results show that the architecture reduces tracking errors (RMSE < 2°). The INN is accurate for decoupled joint dynamics and requires fewer training data than the GNN. A table summarizes the design process. Future works will apply this control structure to low-cost robotic arms and micro-movements.

Keywords: three-dimensional printing; low cost; machine learning; uncertainties

check for updates

Citation: Toussaint, B.; Raison, M. Design of Minimal Model-Free Control Structure for Fast Trajectory Tracking of Robotic Arms. *Appl. Sci.* **2024**, *14*, 8405. https://doi.org/10.3390/app14188405

Academic Editors: Yi Wang and Seong Hyeon Hong

Received: 16 July 2024
Revised: 29 August 2024
Accepted: 5 September 2024
Published: 18 September 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

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]. The ability of robotic arms to accurately follow specified trajectories is important in a large field of applications.

Control structure methods for the successful tracking of such trajectories by robotic arms represent a problem that has been studied for decades [2] and require either an accurate dynamic model or aggressive tracking with high-gain feedback. However, for most robotic arms, an accurate dynamic model can be difficult to obtain due to nonlinearities (e.g., friction) and uncertainties (e.g., inertial parameters) in these dynamic systems [3], making implementation on robotic arms difficult. Thus, the inability of these joint servo controllers to address these nonlinearities and uncertainties can lead to the degradation of accuracy in trajectory tracking [4]. Generally, in industrial robots, the solution consists of finding a compromise between a reduction in the cycle time and an improvement in tracking accuracy [5]. This situation is currently accentuated by the attempts to develop 3D-printed and low-cost robotic arms, which can have higher friction and cheaper motors.

The most common control architecture consists of a joint torque control using a linear controller, such as a traditional proportional–integral–derivative (PID) controller. To compensate for the response time of the controller, a feedforward compensation using the dynamic model of the system is usually added. Widespread alternatives can use simplified models [6] or machine learning methods [3]. Notably, iterative methods, such as iterative

learning control, are based on task repetition and allow for efficient trajectory tracking either with the addition of a dynamic model [5] or without any information on the dynamic model [7,8]. However, if these methods are useful for repetitive tasks, such as working on assembly lines, the learned representation is not easily transferable between different trajectories.

Control structures based on neural networks (NNs) have attracted attention with their potential ability to generalize beyond a training set [5]. For trajectory tracking especially, the power of approximation of NNs has been used in control structures to learn either the forward or inverse dynamics of the system [9–12]. According to [5], when using an NN architecture, the control problem falls either in reinforcement learning [13,14], adaptive control [15], or optimal control [16].

To approximate a system's forward dynamics, several NN architectures, such as recurrent neural networks (RNNs), feedforward networks, or radial basis function (RBF) neural networks, have been used, e.g., [12,17–19]. In [20], NNs were used to learn inverse dynamic models, and they were included as a feedforward component for dynamic compensation. It has already been shown that NNs—especially RNNs and Long Short-Term Memory (LSTM) neural networks—can outperform other techniques, such as Gaussian Processes, to learn inverse dynamics when there are sufficient training data [21,22]. However, most of these works operate at a torque-level control, which usually requires many parameters for the NNs—in the order of 10^5 in [23], for a 7-degrees of freedom (DOF) robot arm—or several neural networks to approximate different elements of the dynamic model. Moreover, when torque control is required, a bridge exists between the results in simulations and in reality. In [24], an adaptive controller based on a neural network was used for robot manipulator trajectory tracking, but solely the simulation results were presented.

Even if the design of a controller plays a major role in control structures, Ref. [25] reminds us that the design of the control structure also answers the following questions: "Which variables should be controlled and measured, and which inputs should be manipulated?" "Which control configurations (e.g., cascade control, feedforward control, etc.) should be used, how must the different controls be organized hierarchically, and how many degrees of freedom must the controller have to achieve the desired performance?".

In [4], as in [26], another architecture is proposed for the control structure. An NN-based control structure is used in an outer loop (i.e., for position control) as feedforward compensation on either a quadrotor with a deep NN or a 7-DOF robot arm with an RNN. These papers highlighted three main advantages of combining an NN system as feedforward compensation with a position PD controller. First, NN architectures can be applied to various systems with complex dynamics while ensuring their stability [26]. Secondly, this approach is easy to use because it does not need any prior information on the system and can be applied to unseen trajectories without any adaptation process [5,11]. Thirdly, with a good training process, this approach demonstrates good accuracy in trajectory tracking while being computationally efficient with a small model [4].

In [4], the learning of the direct dynamic model of a flexible Baxter robot was conducted at the joint position control level by combining a proportional–integral–derivative (PID) controller with either a bidirectional RNN or an RNN associated with an iterative learning control (ILC) algorithm, both with around 10^3 parameters, directly as one integrated model. The results showed a reduction in tracking errors between 12% and 54% depending on the joints, compared to the reference for random trajectories.

In [5], the inverse dynamic learning process of a 7-DOF industrial robot was first conducted in a simulation by implementing the control for many trajectories to collect data for the NN on the real system. However, this technique required a good dynamic simulation of the system, which is rarely available for non-industrial robots because a large amount of data is needed to train the model. In this paper, a trained NN with around $2{\cdot}10^4$ parameters—2 layers of 100 neurons—is employed as a feedforward component to compensate for the dynamic errors of the position controller of an ABB IRB6640-180 robot with 6 DOF. This approach resulted in a diminution of tracking errors by 80% to 90%

compared to tracking errors without compensation, achieving a global mean squared error of 2.145° in their tests. Unlike their previous work [4], which used a single NN for the whole system, the authors mentioned that on the studied ABB robot (Zurich, Switzerland), the joint dynamics were decoupled—i.e., the impact of the joint movements was negligible on the dynamics of the other joints—allowing the possibility to train six separate NN's, each approximating the dynamic inversion for one joint. However, only a global neural network for all joints was implemented.

In [11], we proposed a model-free control structure for high-speed trajectory tracking that combined the advantages of both proportional–derivative (PD) controllers, which are stable and easy to use, and neural networks, which are efficient in estimating the system dynamics. This method differed from [4] and [5] by using one NN per joint (the INN method) instead of one global NN for the entire robot arm (the GNN method). The architecture—illustrated in Figure 1—combines an NN with a PD controller. It consists of learning the dynamic response of the PD controller with an NN and using it in an outer loop as a feedforward compensation to predict and correct the errors of the PD controller, which is effective for robot trajectory tracking [27]. A summary of existing methods for trajectory tracking is presented in Figure 2.

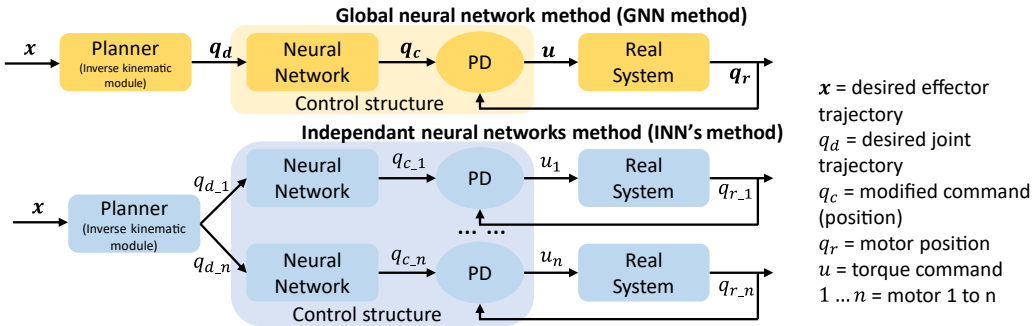


Figure 1. Model-free control structure for high-speed trajectory tracking.

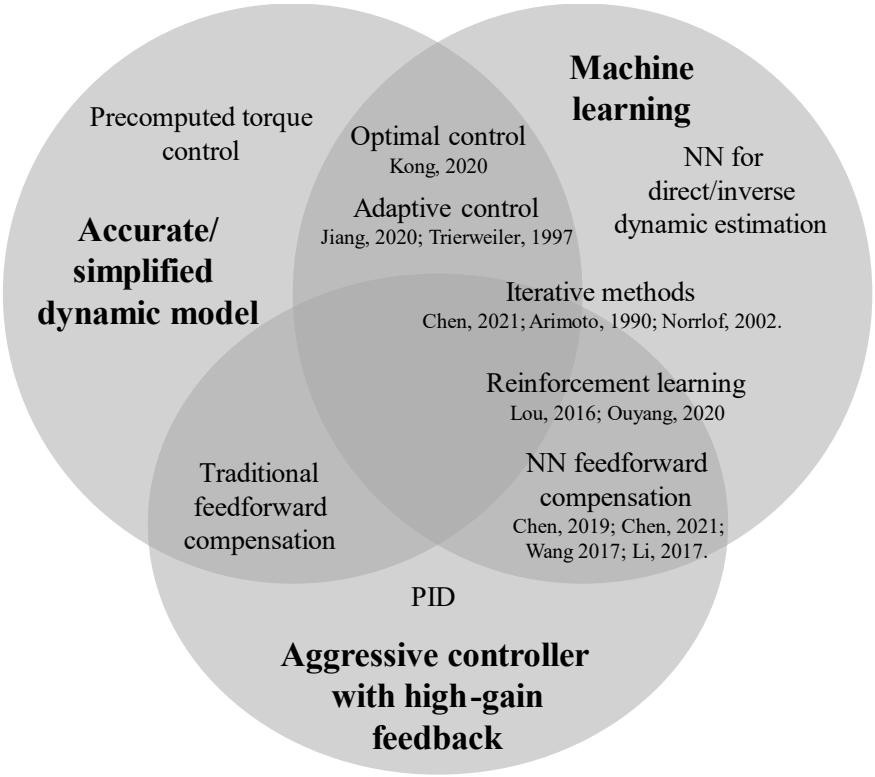
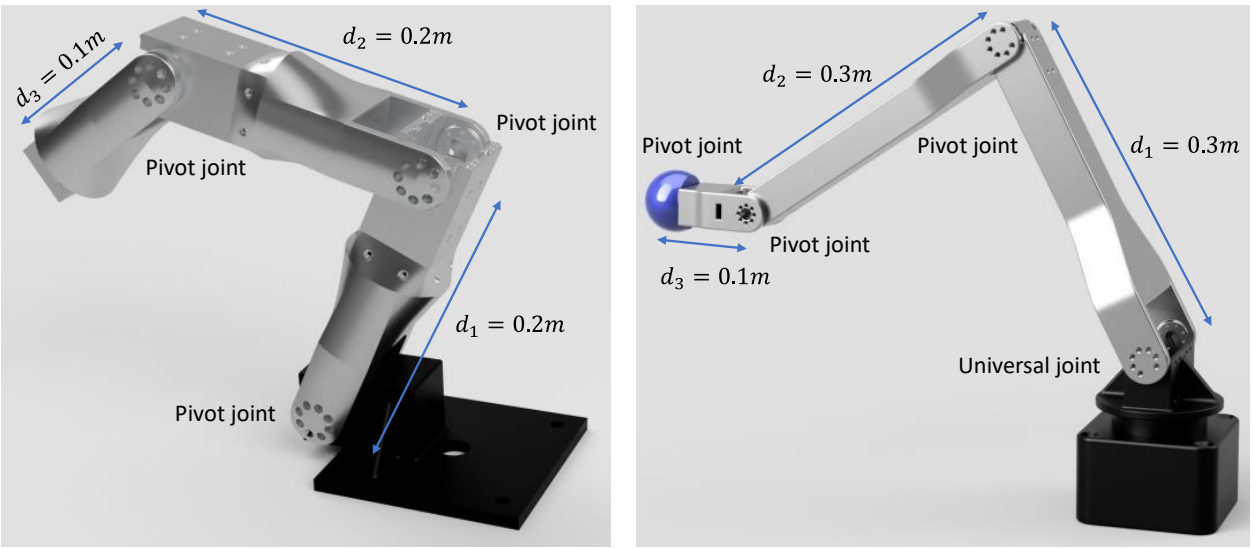


Figure 2. Summary of existing control structures for trajectory tracking [4,5,7,8,12–16,25,26].

From our state-of-the-art review, it appears that using a control structure with an NN architecture as a feedforward component for the inner-loop dynamic compensation allows for reducing the errors in position and velocity during the trajectory tracking without any prior knowledge of the dynamic model [4,5,11]. However, there is a lack of studies optimizing the design process of this model-free control structure. This gap necessitates experimentally selecting learning parameters, often leading to unnecessarily complex architectures. Particularly, the current models still require a large number of parameters—ranging from 10^3 to 10^4 parameters—resulting in non-convex loss functions and higher computational costs during the training process. The minimum number of parameters needed for the NN architecture to provide an effective dynamic compensation is also absent from the literature, as well as the amount of data required to train these architectures.

Moreover, there are no established guidelines on the best NN architecture for the use as feedforward compensation—whether a global NN for the whole system (the GNN method) or smaller NNs for each joint (the INN method)—and the necessary conditions to achieve optimal performances for these methods remain unclear.

The objective of this study is to design a minimal model-free control structure based on a neural network for the fast and accurate trajectory tracking of robotic arms. The subsequent objective is to compare 1. one NN per individual joint (the INN method) and 2. one global NN for all joints (the GNN method). To meet this objective, both real 3D-printed robotic arms (3- and 5-DOF) and simulated robotic arms (3-DOF) were used. We focused on the fast trajectories, i.e., trajectories where the PD is unable to accurately track the trajectories due to the response time limitations. Figure 3 presents the design of the studied robots.



(A) **(B)**

Figure 3. Design of the studied robotic arms for (**A**) simulated and real 3-DOF arms and (**B**) a real 5-DOF arm.

The planar 3-DOF setup allows straightforward results while accounting for external forces like gravity, while the 5-DOF setup demonstrates the architecture's applicability to a widely used topology in industry settings.

2. Materials and Methods

To obtain the required results to evaluate the NN architectures used in the control structure across different configurations, both real 3D-printed (3- and 5-DOF) and simulated robotic arms (3-DOF) were used, as shown in Figure 3.

First, the real 3-DOF planar robotic arm consisted of three 3D-printed bodies made from polylactic acid with segment lengths of $d_1 = 0.2$ m, $d_2 = 0.2$ m, $d_3 = 0.1$ m, respectively. Each body was connected by a pivot joint actuated by XM-540 Dynamixel motors.

Secondly, the real 5-DOF robotic arm consisted of five 3D-printed PLA bodies. Two DOFs were in the base (universal joint), while the remaining three were pivot joints, as shown in Figure 3. All joints were actuated by Dynamixel motors from the XM series. The segment lengths were $d_1 = 0.3$ m, $d_2 = 0.3$ m, $d_3 = 0.1$ m, respectively.

The experimental results were used to validate and enhance the simulated results, ensuring that the simulations represented reality. Specifically, we adjusted simulation parameters, including PID gains, inertias, and masses, to ensure that the accuracy of the simulated PD controller matched that of the real PD controller.

We also assumed that the internal dynamics of the robotic arms were deterministic and repeatable.

2.1. Equations of the Simulated Model

The equations for the simulated system were generated using the Robotran software (version 1.22.0) [28]. The semi-explicit formalism, as presented in (1), also referred to as the direct dynamic model in the literature, was used as follows:

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

where $\boldsymbol{M}[n \cdot n]$ is the symmetric generalized mass matrix of the system; $\boldsymbol{c}[n \cdot 1]$ is the nonlinear dynamic vector containing the gyroscopic, centrifugal and gravity terms, as well as the contributions of external forces and torques; $\boldsymbol{q}[n \cdot 1]$ contains the relative generalized coordinates; $\boldsymbol{\delta}[10n \cdot 1]$ contains the dynamic parameters of the system, i.e., masses, centers of masses, inertia; and $\boldsymbol{\tau}[n \cdot 1]$ contains the generalized joint forces.

To be as realistic as possible, the dynamic effects of the motors were considered by accounting for the inertia of the rotor of each motor and viscous friction, which is velocity-dependent [29]. For simplicity, we neglected the nonlinear friction torque given in this article. Thus, the model of the motors was written as follows:

$$\boldsymbol{J_m}\ddot{\boldsymbol{q}}_{\boldsymbol{m}} + \boldsymbol{D_m}\dot{\boldsymbol{q}}_{\boldsymbol{m}} = \boldsymbol{\tau_m} - \boldsymbol{R}\boldsymbol{\tau}, \tag{2}$$

with $\boldsymbol{\tau}$ corresponding to the generalized joint forces from Equation (1); $\dot{\boldsymbol{q}}_{\boldsymbol{m}}[n \cdot 1]$ and $\ddot{\boldsymbol{q}}_{\boldsymbol{m}}[n \cdot 1]$ being the velocities and accelerations of the n motors, respectively; $\boldsymbol{J_m}$, $\boldsymbol{D_m}$, and $\boldsymbol{R}$ being three $[n \cdot n]$ diagonal matrices corresponding to the moments of inertia, the viscous friction coefficients of the motors, and the gear reduction ratios, respectively.

Finally, $\boldsymbol{\tau_m}[n \cdot n]$ is the vector of torques supplied by the motors, which corresponds to the output of the PD controller. Based on [29], we used (1) to replace $\boldsymbol{\tau}$ in (2) and applied the relation $\boldsymbol{q} = \boldsymbol{R}\boldsymbol{q_m}$ to combine the equations to obtain the complete model of the simulated arm:

$$\boldsymbol{J_m}\ddot{\boldsymbol{q}}_{\boldsymbol{m}} + \boldsymbol{D_m}\dot{\boldsymbol{q}}_{\boldsymbol{m}} = \boldsymbol{\tau_m} - \boldsymbol{R}\left(\boldsymbol{M}\ddot{\boldsymbol{q}} + \boldsymbol{c}\right) \tag{3}$$

$$(\boldsymbol{J_m} + \boldsymbol{RMR})\ddot{\boldsymbol{q}}_{\boldsymbol{m}} + \boldsymbol{D_m}\dot{\boldsymbol{q}}_{\boldsymbol{m}} + \boldsymbol{Rc} = \boldsymbol{\tau_m} \tag{4}$$

$$\ddot{\boldsymbol{q}} = \boldsymbol{R}\ddot{\boldsymbol{q}}_{\boldsymbol{m}} = \boldsymbol{R}(\boldsymbol{J_m} + \boldsymbol{RMR})^{-1}\left(\boldsymbol{\tau_m} - \boldsymbol{D_m}\dot{\boldsymbol{q}}_{\boldsymbol{m}} - \boldsymbol{Rc}\right) \tag{5}$$

An n-DOF robot arm was considered with a joint servocontrol with input $\boldsymbol{q_c} \in R^n$, the joint position commands, and the output $\boldsymbol{q} \in R^n$, the measured joint positions. Our goal was to find a feedforward compensation such that, for desired joint trajectories $\boldsymbol{q_d}$, the corrected input commands $\boldsymbol{q_c}$ would result in the perfect tracking of these desired trajectories: $\boldsymbol{q_c} \rightarrow \boldsymbol{q} = \boldsymbol{q_d}$. The architecture of the control structure is presented in Figure 1.

2.2. Implementation of the Control System

Once the gains of the PD controller were tuned by using traditional tuning methods, the NN architecture for feedforward compensation was designed. All NN architectures

were built by using a multi-layer perceptron regressor, implemented with the "MLPRegressor" function from the Python library "sklearn" [30]. We tested architectures with varying numbers of neurons, starting from the simplest with a single layer of one neuron to more complex configurations, to determine the minimal requirements for achieving good accuracy. In simulation, we evaluated different motor reduction ratios and noise magnitudes to analyze their impact on the control structure accuracy. The range parameter values were set to reflect realistic values of physical systems.

Table 1 sums up parameters tested across different configurations. We provided an open source implementation and training procedure for the architectures discussed in this section in [31].

Table 1. Summary of tested parameters.

Parameter	Values
Arm	Real (3/5 DOF), Simulated (5 DOF)
NN Architecture	INN, GNN
Number of neurons	1, 3, 16, 3 + 3, 16 + 16, 3 × 16
Additional parameters studied in simulation	
PD parameters	RNN
Motor reduction ratio	1, 0.5, 0.25, 0.16, 0.1, 0.01
Noise magnitude (m)	0, 0.001, 0.002, 0.003, 0.005, 0.01, 0.02, 0.04

2.3. Test Procedure

The main steps for designing an accurate control structure with a feedforward neural network were identified prior to the study. To provide a guide for the design of our minimal model-free control structure by using neural networks for feedforward compensation, we studied each of these steps in detail.

Using both real robotic arms and simulated models, we followed these steps:

Build a dataset of trajectories, all with the same duration representing the entire workspace of the arm, by using a simple PD controller on each joint, with the following control law.

$$\boldsymbol{c}(t) = \mathrm{P}(\mathbf{q}(\mathrm{t}) - \mathbf{u}(\mathrm{t})) + \mathrm{D}(\dot{\mathbf{q}}(\mathrm{t}) - \dot{\mathbf{u}}(\mathrm{t})),$$

with $\boldsymbol{c}$ the vector of commands (in current) of the n motors, $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$ the position and velocity of the n motors, $\boldsymbol{u}$ and $\dot{\boldsymbol{u}}$ are the desired position and velocity of the n motors, and P and D the gains of the internal PD controller.

The dataset comprised 1000 trajectories, a number selected to provide enough data to ensure optimal training while being feasible for real systems.

Joint trajectories were generated from a home position by using quintic polynomials by choosing the desired random position and velocity in the workspace of each motor, as follows:

$$q_i(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5, \tag{6}$$

with

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 1 & t_d & t_d^2 & t_d^3 & t_d^4 & t_d^5 \\ 0 & 1 & 2t_d & 3t_d^2 & 4t_d^3 & 5t_d^4 \\ 0 & 0 & 2 & 6t_d & 12t_d^2 & 20t_d^3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} = \begin{bmatrix} q_{i0} \\ \dot{q}_{i0} \\ \ddot{q}_{i0} \\ q_{id} \\ \dot{q}_{id} \\ \ddot{q}_{id} \end{bmatrix}, \tag{7}$$

with q_{i0} being the home position of the ith motor, $\dot{q}_{i0} = 0$ and $\ddot{q}_{i0} = 0$ being the initial velocity and acceleration of the motor, q_{id}, $\dot{q}_{id}$ being the desired position and velocity, and $\ddot{q}_{id} = 0$ being the desired acceleration.

The same equations were used to generate the second half of the trajectory between q_{id} and $q_{if} = q_{i0}$.

All these trajectories were generated at 100 Hz to ensure regular commands were sent to the motors with our computer.

Once the trajectories were executed, the dataset was built using a desired number of trajectories (between 20 and 1000). Each datum of the dataset consisted of five variables for each motor: the position $\boldsymbol{q}$ and the velocity $\dot{\boldsymbol{q}}$ of the motor at two consecutive timestamps—$\boldsymbol{q}(t)$, $\boldsymbol{q}(t+\delta)$, $\dot{\boldsymbol{q}}(t)$, $\dot{\boldsymbol{q}}(t+\delta)$—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. By giving the NN a lot of data from the trajectories, it learned which command had to be sent to each motor to go from a state—$\boldsymbol{q}(t)$, $\dot{\boldsymbol{q}}(t)$—to the next one—$\boldsymbol{q}(t+\delta)$, $\dot{\boldsymbol{q}}(t+\delta)$—and implicitly, the dynamic model of the robotic arm. The multi-layer perceptron regressor was trained with a maximum of 500 iterations by using the "MLPRegressor.fit ()" function from the Python library "sklearn" [30]. This function used the square error as the loss function:

$$L = \sum(\boldsymbol{u}(t) - \boldsymbol{q}(t+\delta)^2)$$

The loss function was minimized by using the Adam solver.

Generate a new set of commands to effectively track the desired trajectory $\boldsymbol{q_d}(t)$, $t \in \left[0, t_f\right]$. For the GNN method, inputs consisted of the ensemble of two successive positions and velocities for all motors—$\boldsymbol{q}(t)$, $\dot{\boldsymbol{q}}(t)$, $\boldsymbol{q}(t+\delta)$, $\dot{\boldsymbol{q}}(t+\delta)$. For the INN method, each NN received inputs related solely to its specific motor, i.e., its successive position and speed—$q_i(t)$, $\dot{q}_i(t)$, $q_i(t+\delta)$, $\dot{q}_i(t+\delta)$.

Outputs were the series of commands required for each motor, by considering the response time of the PD controller to ensure the execution of the desired trajectory. Table 2 summarizes how a command sequence was computed for the trajectory of the joint i.

Table 2. Computation of a command sequence.

Step	Inputs	Output
1	$q_i(0)$, $\dot{q}_i(0)$, $q_i(\delta)$, $\dot{q}_i(\delta)$	$u_i(t)$ for $t \in [0;\ \delta]$
2	$q_i(\delta)$, $\dot{q}_i(\delta)$, $q_i(2\delta)$, $\dot{q}_i(2\delta)$	$u_i(t)$ for $t \in [\delta;\ 2\delta]$
3	$q_i(2\delta)$, $\dot{q}_i(2\delta)$, $q_i(3\delta)$, $\dot{q}_i(3\delta)$	$u_i(t)$ for $t \in [2\delta;\ 3\delta]$
n	$q_i((n-1)\,\delta)$, $\dot{q}_i((n-1)\,\delta)$, $q_i(n\,\delta)$, $\dot{q}_i(n\,\delta)$	$u_i(t)$ for $t \in [(n-1)\delta;\ n\delta]$

Store the tracking errors for n test trajectories ($n = 40$). The number of test trajectories was chosen to ensure low variation in the results while maintaining a reduced number of trajectories for practical feasibility. The accuracy of the control structure for trajectory tracking was evaluated by comparing the root-mean-square errors (RMSE) along the trajectory, $e = \sqrt{mean\left(\sum(\boldsymbol{q} - \boldsymbol{q_d})^2\right)}$, with $\boldsymbol{q}$ being the real executed trajectory and $\boldsymbol{q_d}$ being the desired trajectory.

This test procedure was applied by using different values of parameters and NN architectures to compare their effectiveness to track the test trajectories accurately.

3. Results

Figures 4 and 5 illustrate the impact of the motor reduction ratio on the performance of the control structure for both the GNN and INN methods, which must be considered during the design phase to choose the best architecture for the NN. Additionally, these figures present performances for varying amounts of training trajectories, which is important for optimizing the NN training process.

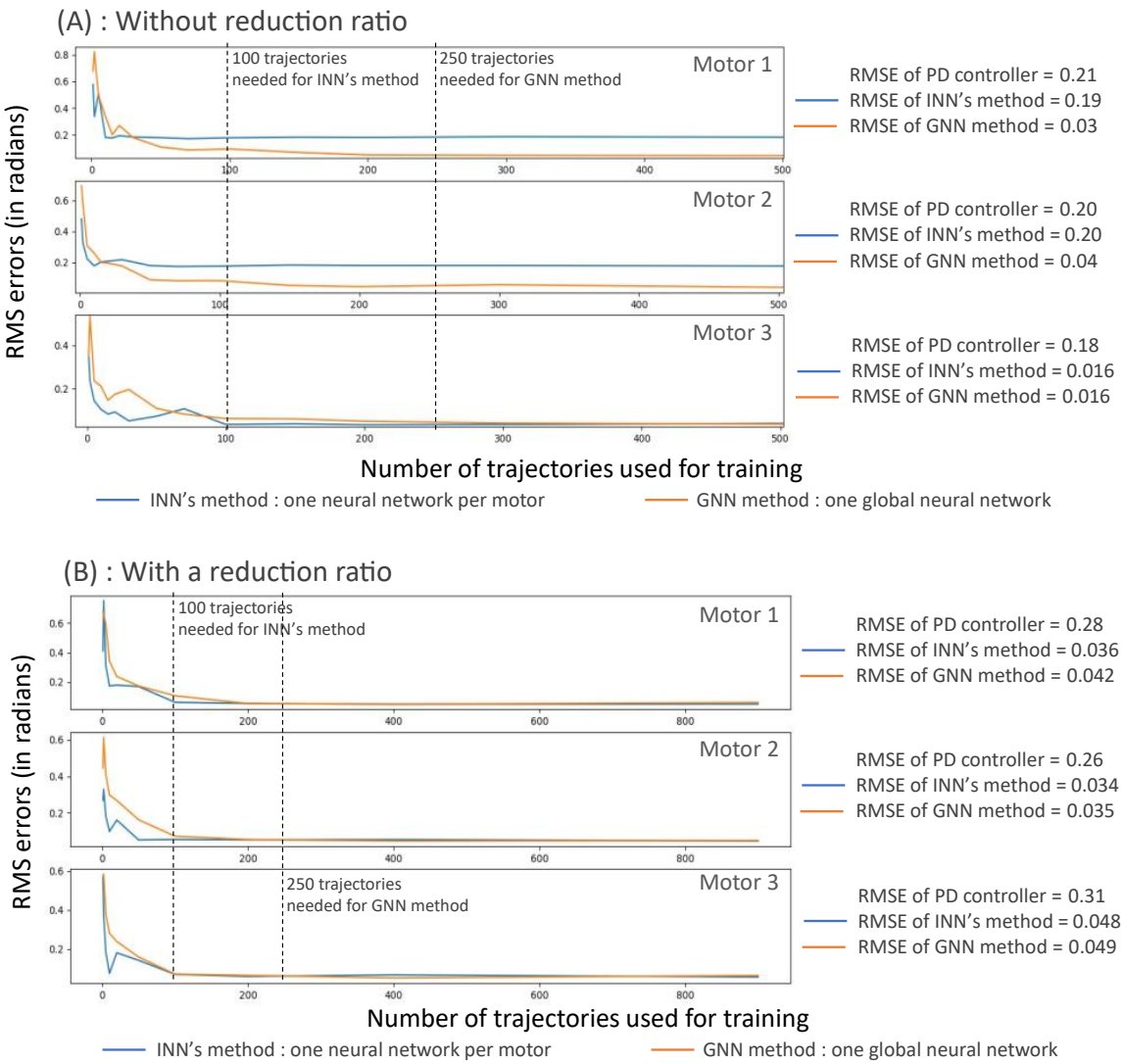


Figure 4. Learning curve with both INN and GNN methods: (**A**) without reduction ratio, (**B**) with a reduction ratio R = 0.01.

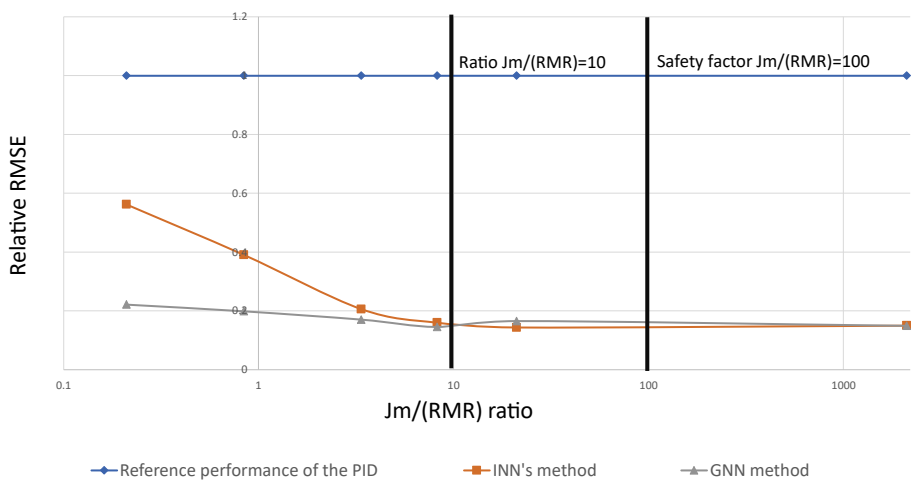


Figure 5. Relative performances of the NN-based control structure compared to the reference PD controller as a function of the $J_m/(RMR)$ ratio.

3.1. A. Results with the 3-DOF Simulated Model without Reduction Ratio

Figure 4A illustrates a typical learning curve obtained with the simulated model from Figure 3A by using both methods. One can observe that the INN method requires 100 trajectories, while the GNN method needs 250 trajectories of 1 s each to maximize their accuracy.

The learning curves for motors 1 and 2 highlight the differences in tracking accuracy between the two studied architectures when more than 100 trajectories are available. The INN method achieves an RMSE of 0.2 radians, whereas the GNN method achieves an RMSE of less than 0.04 radians—or 2.3 degrees. Conversely, the learning curve for the third motor shows an RMSE of 0.016 radians—less than 1 degree—for both methods, representing a 91% reduction in tracking errors compared to the PD controller without feedforward compensation.

3.2. B. Results with the 3-DOF Simulated Model with a Reduction Ratio

Figure 4B presents the learning curves for the two architectures with a reduction ratio R of 0.01. This time, RMSE is under 0.05 radians—3 degrees—for all three motors.

As observed in Figure 4A, the INN method reaches its optimal performance with 100 trajectories vs. 250 for the GNN method.

Figure 5 presents the average relative performance of both methods, with the reference PD controller as a function of the ratio between the inertia $\boldsymbol{J_m}$ of the motor rotor and the mass matrix of the system (***RMR***). The results indicate similar performances for both methods for a ratio above 10 with a relative RMSE of 0.15 compared to the PD controller and a progressive deterioration of performances of the INN method when the ratio decreases. For a ratio of 0.2—obtained when there is no reduction ratio in the motor—the GNN method demonstrates a relative RMSE of 0.22 compared to 0.56 for the PD controller, i.e., 2.5 times less.

Figure 6 shows the relative performances of both methods compared to the reference PD controller depending on its accuracy—or parameters—for two ratios between the frequency of the dynamic integration and the frequency of the feedforward NN compensation to construct an optimal dataset. When the step time corresponds to the interval between two commands from the neural network, the RMSE of the NN is nearly proportional to that of the PD controller, showing a reduction of 90% of its tracking errors. When the frequency of the dynamic integration is ten times higher than the frequency of the NN, the reduction in tracking errors compared to the PD controller is between 80% and 85%, resulting in RMSE values that are two times more important than when the timestamps are the same. Moreover, when the reference PD is more aggressive—i.e., manages to track the trajectories with an RMSE less than 3 degrees—we can see that the performances of the NN are no longer proportional to that of the PD with an RMSE remaining around 0.5 degrees when the two frequencies differ.

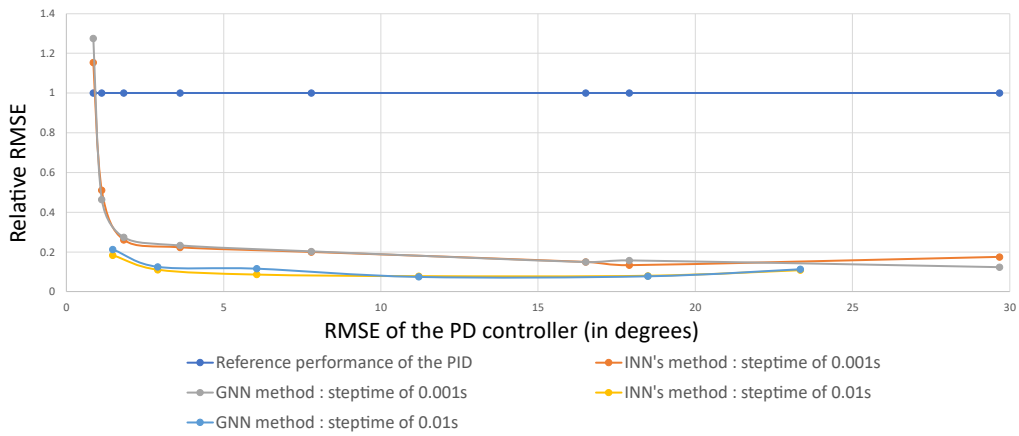


Figure 6. Relative performances of the NN-based control structure compared to the reference PD controller as a function of the accuracy of the PD controller and the step time of the direct dynamic integration.

To design an optimal neural network, the median relative RMSE was compared for different sizes of NNs for both INN and GNN methods tested under various PID parameters. The results are presented in Table 3. All tested architectures, which have at least as many neurons as the number of associated motors on each layer, achieved a median relative accuracy between 0.935 and 1.063 compared to the smallest neural network, representing less than a 10% difference in accuracy. On the contrary, when the number of neurons per layer was fewer than the number of associated motors, the NN failed to converge. This is true for the NN with only 1 neuron in the GNN method, which had an RMSE 24.26 times greater than other NN configurations.

Table 3. Median relative RMSE according to 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	2 × 3	2 × 16	3 × 16	1	3	16	2 × 3	2 × 16	3 × 16
Relative RMSE	1	1.057	0.978	0.935	0.992	0.996	**24.26**	0.970	1.014	1.027	0.989	1.063

Figure 7 shows the relative performances of both methods compared to the reference PD controller depending on the noise present in the training data. In the absence of noise, both INN and GNN methods achieved a 90% improvement in trajectory tracking over the reference PD controller, with an RMS error of 0.86 degrees. A gradual decrease in the accuracy of the system is observed for noise between 0 and 0.5 degrees and is also observed for both INN and GNN methods.

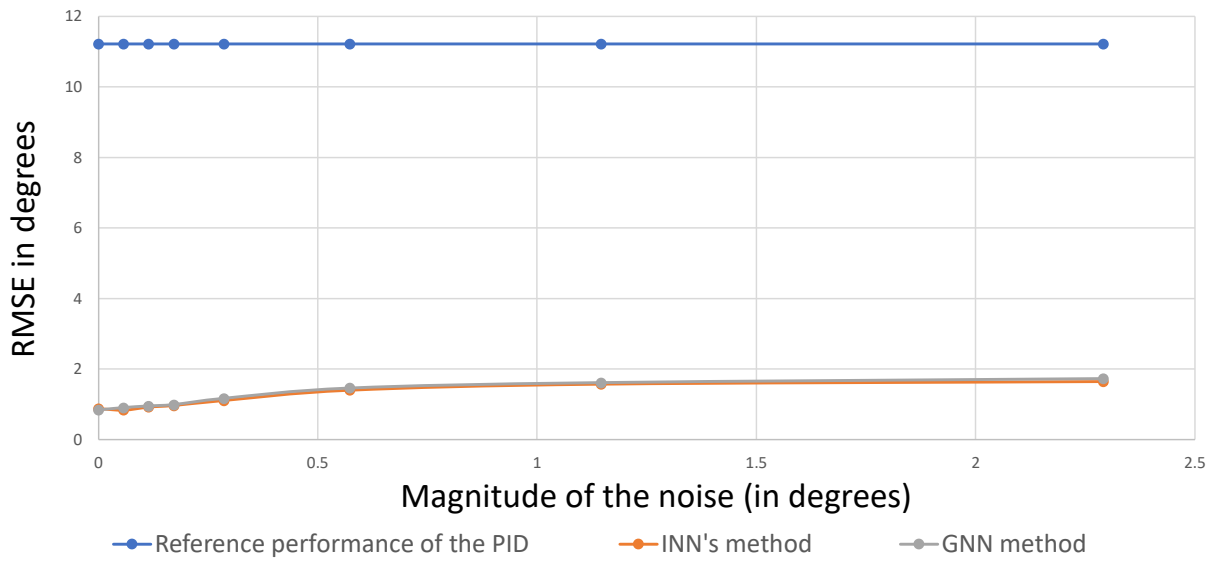


Figure 7. Relative performances of the neural networks to the reference PD controller as a function of the noise magnitude in the training data.

Moreover, when the noise exceeds 0.5 degrees, the RMSE becomes 1.8 times larger than in the absence of noise in the data. However, the RMSE of the NN remains relatively stable—ranging between 1.40 and 1.65 degrees—which is still five times less than the RMSE of the PD controller.

Note that the noise in this figure is quite high; for instance, the noise in the real systems in this study is estimated to be 0.2 degrees (one encoder step).

Finally, with noise levels higher than 0.5 degrees, the NNs with more neurons demonstrated a 20% improvement in accuracy compared to the smallest tested NNs, as summarized in Table 3.

3.3. C. Experimental Results

Figure 8 presents a comparison between trajectory tracking accuracies—measured by RMSE—of the control structure for three configurations using 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.

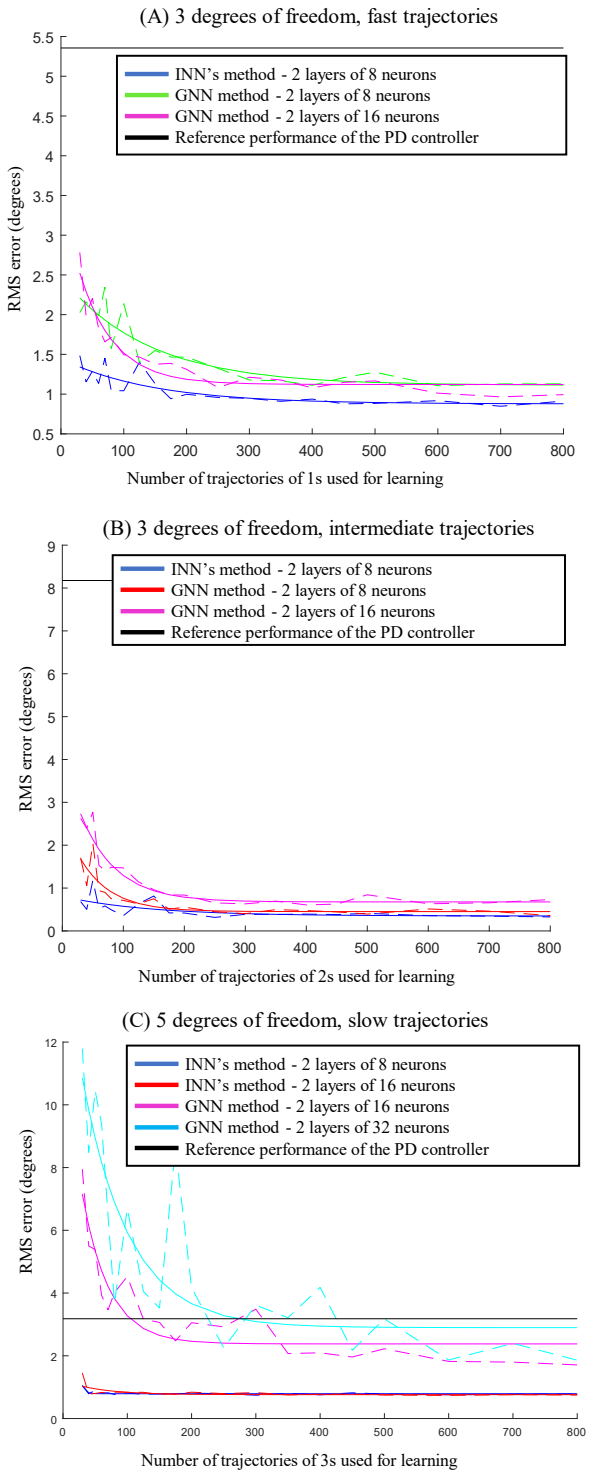


Figure 8. Performance of trajectory tracking with various NNs for (**A**) 3-DOF robotic arm, fast trajectories (**B**) 3-DOF robotic arm, intermediate trajectories; (**C**) 5-DOF robotic arm, slow trajectories.

Figure 8 also shows that the INN method achieves accuracy close to that obtained with all training data by using solely 100 trajectories, whereas the GNN method requires around 300 trajectories to effectively learn the dynamic response of the PD controller, regardless of the robot arm or trajectory speed.

Table 4 summarizes the numerical RMSE values among the n-test trajectories and the reduction rates of the errors compared to the reference PD controller. Specifically, the INN method consistently shows at least a 75% reduction in tracking errors compared to the PD controller alone, with an RMSE under 1 degree across all configurations.

Table 4. Comparison between the trajectory tracking accuracies (RMSE) of NNs for trajectory tracking on two real robot arms.

Trajectory Velocities (and DOF)	RMSE with the Reference PD Controller	RMSE with the GNN Method	RMSE with the INN Method
	In Degrees (Tracking Errors Prediction 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%)

4. Discussion

The first main result is highlighted by the simulated model without reduction ratio in Figure 4A. The INN method is effective solely when the joint dynamics are not coupled with the other joints' dynamics—i.e., the impact of the joint movements on the dynamics of the other joints is negligible. In fact, the INN method performs well for the last motor, with a 91% reduction in tracking errors. This is because the third motor at the end of the arm is not affected by the movements of the first two motors. Contrarily, the INN method does not outperform the PD controller for the first two motors, with an RMSE of 0.2 radians similar to that of the PD controller.

The second main result is highlighted in Figure 4B: the INN method outperforms the GNN method when the dynamics of each joint are not coupled with the other ones and when the training dataset comprises fewer than 250 trajectories. This can be explained by the lower number of parameters required for the INN method. Moreover, both methods present a similar accuracy when more training data are available, with RMSE values ranging from 0.034 and 0.048 for the INN method and from 0.035 and 0.049 for the GNN method. These first two results align with those in [4], where the GNN method was applied to a flexible Baxter robot because of the dependence on the joint dynamics and the statement that it would be possible to use the INN method on an ABB industrial robot because of the decoupled dynamics [4,5].

The decoupling of the joint dynamics occurs when the term RMR in Equation (2) becomes negligible compared to the inertia of the servomotor $\boldsymbol{J_m}$. This means that the influence of the mass matrix on the system is negligible compared to the inertia of the servomotor due to the reduction ratio. This result is consistent with the equations presented in [29]. Specifically, Figure 5 shows that this decoupling occurs when the inertia of the rotor of the motor $\boldsymbol{J_m}$ is more than 10 times larger than the term $\boldsymbol{RMR}$ in Equation (2). In this case, the INN method requires less data than the GNN method to complete the learning process: 100 trajectories vs. 250 trajectories for a 3-DOF robot arm. Moreover, the more complex the controlled robot arm, the more trajectories required for the training of the NN in the GNN method. To our knowledge, no study has explicitly demonstrated this result. Figure 3 also shows that the more trajectories available for training, the more accurate will be the control structure. Notably, the amount of data required for the training is 50 times less than the 500 trajectories of 25 s used in [4]. These ones likely used more trajectories than necessary to ensure an optimal training process of their RNN.

Contrarily, Figures 4A and 5 also show the advantages of using the GNN method for systems with coupled dynamics, i.e., when the inertia of the rotor of the motor $\boldsymbol{J_m}$ is

no more than 10 times larger than $\boldsymbol{RMR}$ in Equation (2). In this case, the GNN method provides better accuracy. For example, when the reduction ratio is 1, the RMSE is 2.5 times smaller than the INN method. In fact, the global NN in the GNN method can learn the dependencies between the different joints and maintain accuracy close to that achieved with decoupled dynamics. To our knowledge, no study has explicitly compared the INN and GNN approaches in systems with coupled dynamics.

To summarize, these first results provide guidelines for both designing the neural network system and its training: if the ratio is greater than 10, a small neural network per servomotor can be used effectively. If the ratio is less than 10, a larger neural network for the entire system will yield better accuracy. Moreover, when the $\boldsymbol{J_m}/(\boldsymbol{RMR})$ ratio is between 10 and 100, the accuracy of the GNN method might be slightly better than the accuracy of the INN method. Thus, if the system accuracy is crucial, it may be beneficial to use both methods to see if the improvement in accuracy justifies the use of the GNN method.

Further, Figure 6 demonstrates that to achieve the best possible accuracy for both the INN and GNN methods, the step time of the training data must match the PD frequency used during the dataset construction. In fact, when the step time of the data used for training corresponds to the PD frequency, the RMSE is twice as low compared to scenarios where the step time is 10 times smaller than the PD frequency. However, to enable the neural network to effectively learn the dynamics of the PD controller, sufficient resolution in the data is required. Particularly, the motor should be able to complete at least 20 encoder steps between 2 consecutive time steps. Thus, the resolution of the encoders can limit the step time of the dataset. To our knowledge, none of the existing literature addresses the impact of the PD frequency on the tracking accuracy and the requirements for encoder resolution for this control structure.

Figure 6 also shows that the more accurate the PD controller, the more accurate the NN-based control structure. Thus, it is strongly recommended to optimize the gain of the PD controller before starting the training process, to ensure the best accuracy for the control structure without the feedforward neural network. In fact, the relative accuracy of the NN-based control structure compared to the PD controller alone remains stable regardless of the accuracy of the PD controller. The tuning can be performed by using manufacturer recommendations when available or through classical tuning methods from the literature.

Table 3 indicates that a necessary condition for the successful use of the GNN method is that the number of neurons in each layer should be larger than or equal to the number of motors in the system for proper convergence. When this condition is not met, the global NN is forced to express the dynamics of multiple motors in one neuron, which is insufficient to accurately represent the joint dynamics. To our knowledge, the current study is the first one to provide guidelines on the minimal size required for the NN architecture to efficiently learn dynamic compensation.

Furthermore, Table 3 shows that, theoretically, when the INN method is used, each neural network may require only a single neuron. In fact, the RMSE comparison for trajectory tracking has shown that when this condition is met, there is no significant difference in accuracy—less than 10 percent—regardless of the NN size. This aligns with the results from [5], which tested various NN sizes, showing no significant performance differences. The best accuracy was achieved with one of the smallest tested architectures. This result encourages the use of minimal neural networks to limit the number of parameters and improve the convergence process.

Compared to other NN-based control structures in the literature, the smallest NN presented in Figure 4B achieves an accuracy comparable to larger networks while requiring only 15 parameters for the entire 3-DOF robot arm—5 parameters per motor. This is 67 times fewer than the system described in [4] and 6700 times fewer than the architecture in [23] that operates at the torque level. This allows a proportional diminution of the calculation time, allowing for increased frequency of the control structure and performance.

In the same way, when the GNN method is used, the NN should obtain at least as many neurons as there are joints in the system (on each layer). Note that in real systems, it

can be necessary to repeat the training process of the NN to achieve convergence, so it is possible to gradually increase the number of neurons and the number of layers until having a good convergence.

Finally, Figure 7 shows that while not critical, it is good practice to use the highest possible encoder resolution to limit the noise in the data. Although a significant noise—more than 0.5 degrees—does not prevent the control structure from improving the trajectory tracking, with an RMSE ranging from 1.4 to 1.65 degrees (five times lower than the PD controller), lower noise levels yield better results. Even though the NN seems capable of managing this noise during the training, Figure 7 also shows that the best results are obtained when the noise is zero—or almost zero with an RMSE of 0.86, 1.8 times lower than when the noise exceeds 0.5 degrees.

Figure 8 and Table 4 reiterate the interest in using an NN as a feedforward compensation in the control structure, applying both the INN and GNN methods to a real 3D-printed robotic arm. Both methods allow an RMSE of less than 2 degrees for all tested configurations during the trajectory tracking on the real physical robot arms. These results are consistent with the errors observed in the simulated model and the errors of the system in [4], i.e., over a 50% improvement for multi-joint trajectories and 40% for random joint trajectories. In fact, in Table 4, the use of the INN method demonstrates a 76% to 96% reduction in errors across all tested configurations. This result matches the simulation results of Figures 3B and 4 with an 80% to 90% reduction in tracking errors and is better than the 40% improvement for random joint trajectories in [4]. This difference is likely due to the type of robot used—a flexible Baxter robot. In [5], where an industrial ABB robot is used, an 80% to 90% reduction has also been reported compared to the reference PD controller alone. Note that the aim of the INN and the GNN methods is not to outperform the accuracy of control structures that include accurate dynamic models when available but rather to offer a simple and reliable way to accurately track desired trajectories.

Specifically, Figure 8A,B show that the improvement is even more significant for faster trajectories, where the PD controller alone is less accurate than slower trajectories, with a 96% and an 84% reduction for the fastest speed vs. a 76% reduction for the slowest trajectories. Significantly, the INN method achieves an RMSE under 1 degree across all configurations. This result is better than the accuracy of all control structures tested in [5], which had RMSE values between 2.14 and 2.62 degrees, while also using a smaller number of parameters in the NN architectures—1 neuron per motor vs. at least two layers of 50 neurons.

The experimental results in Figure 8 also show that the GNN method does not outperform the accuracy of the INN method. These results are consistent with the decoupled joint dynamics, i.e., the dynamics of the servomotors prevail over 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 from the simulated model on a real system: the INN method requires two to three times less data than the GNN method to effectively learn the dynamics.

A guide summarizing the conclusions of this study for designing a neural network-based control structure for position control is presented in Table 5.

Table 5. Guide for the design of a neural network-based control structure for position control, followed by an example of its application on the real 3- and 5-DOF systems.

Step 1—Tune the Gains of the PD Controller. Tune the gain values of the PD controller to allow the best accuracy possible for the controller, using either recommendations of motor manufacturers when available or tuning methods from the literature.
Applied to the real systems: the custom parameters were based on the recommendation of the manufacturer.
Step 2—Choose between INN and GNN methods. Estimate the ratio $J_m/(RMR)$ (see a suggested method in Appendix A). Refer to the cases below.

Table 5. *Cont.*

Case 1: the ratio is larger than 100: INN method will have a good accuracy in the trajectory tracking with a minimum number of parameters to train in the NN	**Case 2: the ratio is between 10 and 100, or the ratio is hard to estimate.** INN method should have a good accuracy, but it is advised to compare with the accuracy of GNN method if the accuracy of the system is crucial.	**Case 3: the ratio is smaller than 10:** Using GNN method is necessary to obtain good accuracy.
Applied to the real systems: the ratio is estimated to 194 (see Appendix A) for the 3-DOF system (case 1). Thus, we chose INN method for the 3-DOF system. *For the 5-DOF system, the ratio is estimated to 31 (see Appendix A). Thus, we implemented both methods (case 2); as they both led to the same accuracy, we chose INN method.*		
Step 3—Build the training dataset, following two recommendations: 3.1 The trajectories of the dataset should be randomly generated in the whole workspace of each motor: at least 100 trajectories for INN method or 100 trajectories for each DOF of the system for GNN method. 3.2 If a sufficient encoder resolution is available (the motor should be able to go through at least 20 steps of the encoder between two consecutive time steps), the step time of the PD controller should be equal to the step time of the NN architecture. Otherwise, the choice of the step time should be adapted to the motor encoder resolution.		
Applied to the real systems: To build the training dataset: *1—At least 100 trajectories per motor were randomly generated in the whole workspace of each motor.* *2—The resolution of encoders (4096 steps/rev) and the maximum velocity of the motors (30 rpm) allow a maximum step time of 0.01s (20 steps of the encoder between each time step). Thus, while the PD controller was internal to our motors and with a step time of 0.001s, the step time of the training data was reduced to 0.01s.*		
Step 4—Build and train the NN architecture.		
Case 1: INN method chosen at step 2, build INN, i.e., one NN for each joint of the system with only one neuron. If the training does not converge, it is possible to gradually increase the number of neurons.	**Case 2: GNN method chosen at step 2,** build one GNN, i.e., one NN for the whole robotic arm with at least as many neurons as the number of joints on each layer. If the training does not work, it is possible to gradually increase the number of neurons and the number of layers.	
Then, train the NN architecture with all available data (at least 100 trajectories for INN method or 100 trajectories for each DOF of the system for GNN method)		
Applied to the real systems: the datasets of 100 trajectories per motor were used to train the real 3- and 5-DOF systems. The "MLPRegressor.fit()" function was used to train the NN. *When the NN was trained with only 1 neuron, it did not converge. Thus, we chose to increase the number of neurons to 4 for the real system.*		
Step 5—Add the trained NN in the control structure as a feedforward compensation to predict and correct the trajectory tracking errors.		
Applied to the real systems: The "MLPRegressor.predict()" function was used to generate the commands for new trajectories.		

White background refer to our experimental application of the guidelines while grey background refers to the general guidelines.

The main limitation of this study is that the control structure has been designed for fast trajectory tracking involving large movements and accelerations. For slow and micro-movements, the control structure is not optimized, and the accuracy could be improved by using an internal PID instead of a PD controller to correct static errors. However, a PID would be less accurate for fast trajectory tracking [27]. Future works could also explore more complex NN structures, for example, architectures specialized in time-series prediction as RNN or LSTM.

5. Conclusions

The objective of this paper was to design a minimal neural network (NN)-based model-free control structure for fast and accurate trajectory tracking of robotic arms. Particularly,

the study compared the INN method by using one neural network for each joint of the robot arm and the GNN method by using a single global neural network for the entire system. The results were illustrated by two serial robotic arms with 3 and 5 degrees of freedom in a simulation, then in reality, across several trajectory velocities ranging from 1 to 3 s.

The main results were as follows: 1. NN used as a feedforward compensation significantly reduced trajectory tracking errors (RMSE < 2°, see Table 2) without requiring any prior information about the dynamic model. 2. The study showed that the INN method can be used when joint dynamics are decoupled and requires three times less data than the GNN method to learn the dynamics. 3. Table 3 presents the guidelines for designing an optimal minimal NN-based control structure for accurate trajectory tracking in five main steps: 1. Tune the PD gains to optimize the accuracy of the PD controller. 2. Create an NN with an architecture that allows good accuracy with the fastest calculation. 3. Build a dataset representing the entire workspace of each motor. 4. Train the NN system by using all available data (at least 100 trajectories for the INN method or 100 trajectories per DOF of the system for the GNN method). 5. Use the trained NN in the control structure as a feedforward compensation to predict and correct the trajectory tracking errors.

Future perspectives will be to apply these guidelines to the development of 3D-printed low-cost robotic arms and extend them to other systems, such as automated visual inspection robots involving micro- and slow movements.

Author Contributions: Conceptualization: B.T. and M.R.; methodology: B.T. and M.R.; investigation: B.T. and M.R.; visualization: B.T. and M.R.; funding acquisition B.T. and M.R.; project administration: B.T. and M.R.; supervision: M.R.; writing—original draft: B.T. writing—review and editing: B.T. and M.R. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by grants from the Institute for Data Valorisation (IVADO), Montreal, Canada, and from NSERC-Discovery. Number: 796536531.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: All data are available in the main text.

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Appendix A

Appendix A.1. Estimation of the $\boldsymbol{J_m}/(\boldsymbol{RMR})$ Ratio

To quickly obtain an estimation of the magnitude of the impact of motor inertia $\boldsymbol{J_m}$ and the impact of the robot dynamics $\boldsymbol{RMR}$ on joint dynamics for a serial system, the estimation of the biggest value of the $\boldsymbol{RMR}$ matrix is proposed as follows:

$$\max(\boldsymbol{RMR}) = \left(\sum m_i d_i^2 + \sum I_i\right)\boldsymbol{R}^2$$

with m_i the mass of the ith body, d_i being the maximum distance between the basis of the system and the center of mass of the ith body, I_i being the biggest inertia of ith body, and R being the gear reduction ratios of the motors.

Then, it is possible to compare with the value of the motor inertia.

Appendix A.2. Case Example 1—Real 3-DOF Robotic Arm

For 3-DOF real system, the parameters were $m_1 = m_2 = m_3 = 0.25$ kg,$I_1 = I_2 = I_3 = 3.1310^{-4}$ kgm^2, $d_1 = 0.05$ m, $d_2 = 0.15$ m, $d_3 = 0.25$ m, and $R = 1/270$.

Thus, $\max(\boldsymbol{RMR}) = 3.1310^{-7}$ kgm^2.

Moreover, the inertia of the motor around the rotating axis is given by the manufacturer: $J_{mi} = 6.08\ 10^{-5}$ kgm^2. Finally, the ratio $\frac{J_{mi}}{\max(\boldsymbol{RMR})} = 194$.

Appendix A.3. Case Example 2—Real 5-DOF Robotic Arm

For 5-DOF real system, the parameters were $m_1 = m_2 = m_3 = 0.3$ kg, $m_4 = 0.15$ kg, $m_5 = 0.1$ kg $I_1 = I_2 = I_3 = 3.7510^{-4}$ kgm^2, $I_4 = I_5 = 6.7510^{-5}$ kgm^2, $d_1 = 0$ m, $d_2 = 0.05$ m, $d_3 = 0.35$ m, $d_4 = d_5 = 0.65$ m, and $R = 1/270$.

Thus, $\max(\boldsymbol{RMR}) = 1.9810^{-6}$ kgm^2. Finally, the ratio $\frac{J_{mi}}{\max(\boldsymbol{RMR})} = 31$.

References

1. Taylor, C. Robots Could Take over 20 Million Jobs by 2030, Study Claims. Available online: https://www.cnbc.com/2019/06/26/robots-could-take-over-20-million-jobs-by-2030-study-claims.html (accessed on 25 June 2024).
2. Zhihong, M.; Palaniswami, M. Robust Tracking Control for Rigid Robotic Manipulators. *IEEE Trans. Autom. Control* **1994**, *39*, 154–159. [CrossRef]
3. Koç, O.; Maeda, G.; Peters, J. Optimizing the Execution of Dynamic Robot Movements with Learning Control. *IEEE Trans. Robot.* **2019**, *35*, 909–924. [CrossRef]
4. Chen, S.; Wen, J.T. Neural-Learning Trajectory Tracking Control of Flexible-Joint Robot Manipulators with Unknown Dynamics. In Proceedings of the 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), the Venetian Macao, Macau, 3–8 November 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 128–135.
5. Chen, S.; Wen, J.T. Industrial Robot Trajectory Tracking Control Using Multi-Layer Neural Networks Trained by Iterative Learning Control. *Robotics* **2021**, *10*, 50. [CrossRef]
6. Spong, M.; Khorasani, K.; Kokotovic, P. An Integral Manifold Approach to the Feedback Control of Flexible Joint Robots. *IEEE J. Robot. Autom.* **1987**, *3*, 291–300. [CrossRef]
7. Arimoto, S. Learning Control Theory for Robotic Motion. *Int. J. Adapt. Control Signal Process* **1990**, *4*, 543–564. [CrossRef]
8. Norrlof, M. An Adaptive Iterative Learning Control Algorithm with Experiments on an Industrial Robot. *IEEE Trans. Robot. Autom.* **2002**, *18*, 245–251. [CrossRef]
9. Chen, S.; Wen, J.T. Adaptive Neural Trajectory Tracking Control for Flexible-Joint Robots with Online Learning. In Proceedings of the 2020 IEEE International Conference on Robotics and Automation (ICRA), Paris, France, 31 May–31 August 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 2358–2364.
10. He, W.; Yan, Z.; Sun, Y.; Ou, Y.; Sun, C. Neural-Learning-Based Control for a Constrained Robotic Manipulator with Flexible Joints. *IEEE Trans. Neural Netw. Learn. Syst.* **2018**, *29*, 5993–6003. [CrossRef] [PubMed]
11. Toussaint, B.; Raison, M. High-Speed Trajectory Tracking on Robotic Arm by Learning the Dynamic Response of the Pid Controller with a Neural Network. In Proceedings of the 10th ECCOMAS Multibody Dynamics Conference; Budapest University of Technology and Economics, Budapest, Hungary, 12–15 December 2021.
12. Wang, M.; Ye, H.; Chen, Z. Neural Learning Control of Flexible Joint Manipulator with Predefined Tracking Performance and Application to Baxter Robot. *Complexity* **2017**, *2017*, 7683785. [CrossRef]
13. Lou, W.; Guo, X. Adaptive Trajectory Tracking Control Using Reinforcement Learning for Quadrotor. *Int. J. Adv. Robot. Syst.* **2016**, *13*, 38. [CrossRef]
14. Ouyang, Y.; Dong, L.; Wei, Y.; Sun, C. Neural Network Based Tracking Control for an Elastic Joint Robot with Input Constraint via Actor-Critic Design. *Neurocomputing* **2020**, *409*, 286–295. [CrossRef]
15. Jiang, Y.; Wang, Y.; Miao, Z.; Na, J.; Zhao, Z.; Yang, C. Composite-Learning-Based Adaptive Neural Control for Dual-Arm Robots with Relative Motion. *IEEE Trans. Neural Netw. Learn. Syst.* **2020**, *33*, 1010–1021. [CrossRef] [PubMed]
16. Kong, L.; He, W.; Yang, C.; Sun, C. Robust Neurooptimal Control for a Robot via Adaptive Dynamic Programming. *IEEE Trans. Neural Netw. Learn. Syst.* **2020**, *32*, 2584–2594. [CrossRef] [PubMed]
17. He, W.; Dong, Y.; Sun, C. Adaptive Neural Impedance Control of a Robotic Manipulator with Input Saturation. *IEEE Trans. Syst. Man Cybern. Syst.* **2015**, *46*, 334–344. [CrossRef]
18. Pérez-Cruz, J.H.; Chairez, I.; Rubio, J.; Pacheco, J. Identification and Control of Class of Non-linear Systems with Non-symmetric Deadzone Using Recurrent Neural Networks. *IET Control Theory Appl.* **2014**, *8*, 183–192. [CrossRef]
19. Yoo, S.J.; Park, J.B.; Choi, Y.H. Adaptive Output Feedback Control of Flexible-Joint Robots Using Neural Networks: Dynamic Surface Design Approach. *IEEE Trans. Neural Netw.* **2008**, *19*, 1712–1726. [PubMed]
20. Talebi, H.A.; Patel, R.V.; Khorasani, K. Inverse Dynamics Control of Flexible-Link Manipulators Using Neural Networks. In Proceedings of the 1998 IEEE International Conference on Robotics and Automation (Cat. No. 98CH36146), Leuven, Belgium, 20 May 1998; IEEE: Piscataway, NJ, USA, 1998; Volume 1, pp. 806–811.
21. Calandra, R.; Ivaldi, S.; Deisenroth, M.P.; Rueckert, E.; Peters, J. Learning Inverse Dynamics Models with Contacts. In Proceedings of the 2015 IEEE International Conference on Robotics and Automation (ICRA), Seattle, WA, USA, 26–30 May 2015; IEEE: Piscataway, NJ, USA, 2015; pp. 3186–3191.
22. Rueckert, E.; Nakatenus, M.; Tosatto, S.; Peters, J. Learning Inverse Dynamics Models in o (n) Time with Lstm Networks. In Proceedings of the 2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids), Birmingham, UK, 15–17 November 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 811–816.
23. Yang, C.; Wang, X.; Cheng, L.; Ma, H. Neural-Learning-Based Telerobot Control with Guaranteed Performance. *IEEE Trans. Cybern.* **2016**, *47*, 3148–3159. [CrossRef] [PubMed]

24. Sun, T.; Pei, H.; Pan, Y.; Zhou, H.; Zhang, C. Neural Network-Based Sliding Mode Adaptive Control for Robot Manipulators. *Neurocomputing* **2011**, *74*, 2377–2384. [CrossRef]
25. Trierweiler, J. A Systematic Approach to Control Structure Design. Ph.D. Thesis, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil, 1997.
26. Li, Q.; Qian, J.; Zhu, Z.; Bao, X.; Helwa, M.K.; Schoellig, A.P. Deep Neural Networks for Improved, Impromptu Trajectory Tracking of Quadrotors. In Proceedings of the 2017 IEEE International Conference on Robotics and Automation (ICRA), Singapore, 29 May–3 June 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 5183–5189.
27. Kawamura, S.; Miyazaki, F.; Arimoto, S. Is a Local Linear PD Feedback Control Law Effective for Trajectory Tracking of Robot Motion? In Proceedings of the 1988 IEEE International Conference on Robotics and Automation, Leuven, Belgium, 20 May 1998; IEEE: Piscataway, NJ, USA, 1998; pp. 1335–1340.
28. Docquier, N.; Poncelet, A.; Fisette, P. ROBOTRAN: A Powerful Symbolic Gnerator of Multibody Models. *Mech. Sci.* **2013**, *4*, 199–219. [CrossRef]
29. Rocco, P. Stability of PID Control for Industrial Robot Arms. *IEEE Trans. Robot. Autom.* **1996**, *12*, 606–614. [CrossRef]
30. Scikit-Learn: Machine Learning in Python—Scikit-Learn 1.5.0 Documentation. Available online: https://scikit-learn.org/stable/ (accessed on 25 June 2024).
31. Toussaint, B. Public Code of Paper: Design of a Minimal Model-Free Controller for Fast Robotic Arms Trajectory. Available online: https://github.com/LiBRTy-Lab/minimal_neural_controller (accessed on 16 July 2024).

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.