POLITECNICO DI BARI

DIPARTIMENTO DI INGEGNERIA ELETTRICA E DELL'INFORMAZIONE

LAUREA MAGISTRALE IN INGEGNERIA DELL'AUTOMAZIONE

Robotics 1st Module : Industrial handling

Prof. Eng. Paolo Lino

Documentation title:

Motion Control with two-layer Neural Network

Students:
Nicola Saltarelli
Francesco Di Chio

ANNO ACCADEMICO 2024-2025

Abstract

This documentation presents the implementation of a two-layer neural network for motion control of a 6-degree-of-freedom (DOF) robotic manipulator, specifically the Puma 560, a manipulator already available in Peter Corke's Robotics Toolbox. The project was developed using Simulink in combination with Peter Corke's Robotics Toolbox.

The primary objective is to demonstrate how a neural network combined with a PD controller can be employed to control the manipulator without explicit knowledge of its dynamics, which are not assumed to be known a priori. Two distinct backpropagation techniques, utilized for online training (i.e., without offline pre-training), will be presented. The performance of both techniques will be compared, highlighting the limitations and advantages of each methodology. This analysis aims to provide insights into the effectiveness of neural network-based control strategies in handling the complexities of robotic manipulators under dynamic conditions.

Contents

1 Neural Network Robot Control

Most commercially available robot controllers implement some variety of PID control algorithms. PID control allows accuracy acceptable for many applications at a set of 'via' points, but it does not allow accurate dynamic trajectory following between the via points. As performance requirements on speed and accuracy of motion increase in today's manufacturing environments, PID controllers lag further behind in providing adequate robot manipulator performance. Since most commercial controllers do not use any adaptive or learning capability, control accuracy is lost when unknown frictions change, for force control in surface finishing applications, and elsewhere. In this section we show how to use biologically inspired control techniques to remedy these problems.

1.1 Background on Neural Networks

A neural network (NN) is a universal approximator whose functioning is inspired by the biological neural networks found in living organisms. A biological neuron is a specialized cell that operates as a basic input-output mechanism: information received from the dendrites (the neuron's inputs) propagates as an electrical impulse, is processed in the nucleus, and is ultimately transmitted to the axon (the neuron's output). The connection between two neurons, known as a synapse, occurs through an electrochemical process, enabling the exchange of information and the formation of highly interconnected networks.
Similarly, a neural network can be represented as a directed graph where each node corresponds to a neuron, modeled as a simple mathematical function. The graph's edges represent artificial synapses and are associated with numerical parameters called synaptic weights, which determine the strength of the connections between neurons and regulate the network's learning process.

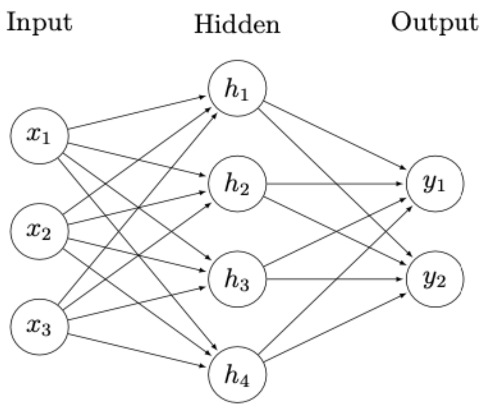

Figure 1: Neural Network base scheme.

1.1.1 Neuron Mathematical Model

The neuron, also known as a Perceptron, is a MISO (Multiple-Input-Single-Output) system, where the inputs $[x_1, x_2, \ldots, x_n]$ are connected to the neuron through synaptic weights $[v_1, v_2, \ldots, v_n]$. The neuron's core N contains a basic mathematical function that processes the weighted inputs and provides the output y, computed as the weighted sum of the inputs.

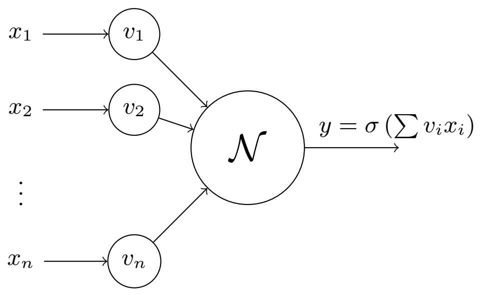

Figure 2: Artificial Neuron scheme.

In addition to the weighted sum, the output of each neuron includes a constant term v_0, known as bias, which acts as an offset. A mathematical model of the neuron can be expressed as:

$$y = \sigma\left(\sum_{i=1}^{n} v_i x_i + v_0\right) \tag{1}$$

Positive weights w_i correspond to excitatory synapses and negative weights to inhibitory synapses. The cell function $\sigma(\cdot)$ is known as the activation function, and is selected differently in different applications. The intent of the activation function is to model the behavior of the cell where there is no ouput below a certain value of the argument of $\sigma(\cdot)$ and the output takes a specified magnitude above that value of the argument. A general class of monotonically nondecreasing functions taking on bounded values at $-\infty$ and $+\infty$ is known as the sigmoid functions. It is noted that, as the threshold or bias w_0 changes, the activation functions shift left or right. For many training algorithms (including backpropagation), the derivative of $\sigma(\cdot)$ is needed so that the activation function selected must be differentiable.
The expression for the neuron output $y(t)$ can be streamlined by defining the column vector of input signals $\bar{x}(t) \in \mathbb{R}^n$ and the column vector of NN weights $\bar{v}(t) \in \mathbb{R}^n$

$$\bar{x}(t) = [x_1\ x_2 \ldots x_n]^T,\ \bar{v}(t) = [v_1\ v_2 \ldots v_n]^T \tag{2}$$

Then, one may write in matrix notation

$$y = \sigma(\bar{v}^T \bar{x} + v_0) \tag{3}$$

A final refinement is achieved by defining the augmented input column vector $x(t) \in \mathbb{R}^{n+1}$ and NN weight column vector $v(t) \in \mathbb{R}^{n+1}$ as

$$x(t) = [1\ \bar{x}^T]^T,\ v(t) = [v_0\ \bar{v}^T]^T \tag{4}$$

Then, one may write

$$y = \sigma(v^T x) \tag{5}$$

In the remainder of this documentation we shall not show the overbar on vectors.

1.1.2 Multilayer Perceptron

A neural network (NN) is thus a system composed of interconnected neurons, arranged in a specific topology. The most common configuration is the Multi-Layer Perceptron (MLP), where neurons are organized into layers. A layer is defined as a set of neurons sharing the same inputs, and the outputs of one layer serve as inputs for the next. The MLP architecture consists of:

- Input Layer: receives the input data and passes it to the network;
- Output Layer: contains as many neurons as the system's output variables;
- One or more Hidden Layers: composed of an arbitrary number of neurons responsible for the nonlinear transformation of input data.

The MLP architecture allows neural networks to learn complex relationships between inputs and outputs, making them powerful tools for modeling and optimizing nonlinear systems.

One of the most relevent and used multilayer perceptron in the control applications is the Two-layers NN. This because this NN meets minimum requirements, in terms of the size of neural networks, for the universal function approximation property. In fact, it is shown that one-layer NN generally does not have universal approximation capability.

The universal approximation of functions is a very powerful property which justifies the use of NNs in control techniques : all problems related to the knowledge of the dynamic model, sometimes difficult to derive, of a system subject to control can be overcome because it is the network which, by exploiting this intrinsic property, generates a function that approximates, in fact, the dynamic model of the system.

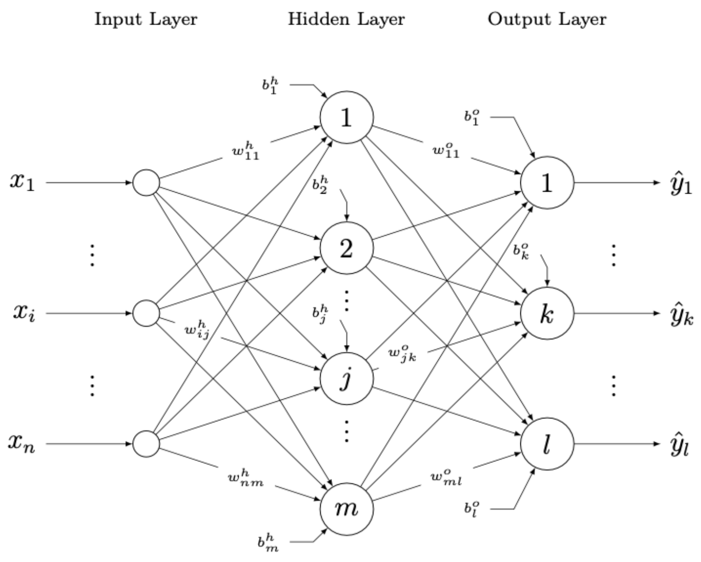

Figure 3: MLP architecture.

In particular, the basic universal approximation result says that any smooth function $f(x)$ can be approximated arbitrarily closely on a compact set using a two-layer NN with appropriate weights. Specifically, let $f(x) : \mathbb{R}^n \to \mathbb{R}^m$ be a smooth function. Then, given a compact set $S \in \mathbb{R}^n$ and a positive number ε_N, there exists a two-layer NN such that

$$f(x) = W^T \sigma(V^T x) + \varepsilon \tag{6}$$

with $\|\varepsilon\| < \varepsilon_N$ for all $x \in S$, for some (sufficiently large) number L of hidden-layer neurons. The value ε is called the *NN function approximation error*, and it decreases as the hidden-layer size L increases. This is due to the fact that, as the size of the NN increases, its approximate power increases, resulting in a decrease of the approximation error.

2 Two-layers Neural Network Controller

In this section the main idea and the structure of the implemented controller will be presented, focusing on two different weights' tuning techniques : *Unsupervised Backpropagation Tuning* for the ideal case and *Augmented Backpropagation Tuning* for the not ideal case. As will be shown, these techniques differ from each other for the rules of updating weights and for the generation of a robustness term, absent in the controller for the ideal case, that allows to have excellent performance even in case of disturbances on the manipulator.

2.1 Dynamics equations and controller structure

It is well known that the dynamics equation of a manipulator robot is the following one :

$$M(q)\ddot{q} + V_m(q,\dot{q})\dot{q} + F(\dot{q}) + G(q) + \tau_d = \tau \tag{7}$$

where

- $M(q)$: inertia matrix, functions of the manipulator configuration;
- $V_m(q,\dot{q})$: coriolis matrix, functions of the joint variables and their derivative;
- $F(\dot{q})$: friction forces vector, function of joint variables derivative;
- $G(q)$: gravity vector, function of manipulator configuration;
- τ_d : external disturbances vector;
- τ : control forces or torque applied to the joint.

The purpose of the controller is to generate the control input τ that allows the robot to follow the desired trajectory. In other words, the aim of the control strategy is to minimize the tracking and filtered error.
In particular, defining $q_d(t)$ a *prescribed desired trajectory*, the tracking error $e(t)$ and the filtered one $r(t)$ are defined as :

$$e = q_d - q \tag{8}$$

$$r = \dot{e} + \Lambda e \tag{9}$$

with $\Lambda > 0$ a positive definite design parameter matrix.
Expressing the dynamic equation (7) in terms of errors one obtain:

$$M\dot{r} = -V_m r + f(x) + \tau_d - \tau \tag{10}$$

where

$$f(x) = M(q)(\ddot{q}_d + \Lambda\dot{e}) + V_m(q,\dot{q})(\dot{q}_d + \Lambda e) + F(\dot{q}) + G(q). \tag{11}$$

The function $f(x)$ is evidently based on knowledge of the dynamics characteristics of the manipulator. The Neural Network therefore tries to find a function that approximates the dynamic behavior described by $f(x)$, but without relying on knowledge of the manipulator's own descriptive matrices.
By defining $x \equiv [e^T\ \dot{e}^T\ q_d^T\ \dot{q}_d^T\ \ddot{q}_d^T]^T$, according to the universal approximation property of NN, there is a Two-layer NN such that

$$f(x) = W^T\sigma(V^T x) + \varepsilon \tag{12}$$

with $||\varepsilon|| < \varepsilon_n$, where ε_n is a known bound of the approximation error ε.

A very important aspect is that V and W, respectively the weight matrices of the first and second

layer, are the ideal weight that perfectly describe the function $f(x)$. However, the output of the neural network is only an estimate $\hat{f}(x)$ of the function $f(x)$.
Defining the approximate function $\hat{f}(x) = \hat{W}^T\sigma(\hat{V}^T x)$, one can introduce the weight *deviations* or *weight estimation errors* as

$$\tilde{V} = V - \hat{V}, \qquad \tilde{W} = W - \hat{W} \tag{13}$$

The smaller the weight estimation error, the better the performance of the NN. For this reason, as will be shown lately, a candidate Lyapunov function will be function not only of the filtered error $r(t)$, but also of $\tilde{V}$ and $\tilde{W}$. The aim is to zero all these quantities.

The control input is then generated in such a way that:

$$\tau = \hat{f}(x) + K_v r - v = \hat{W}^T\sigma(\hat{V}^T x) + K_v r - v \tag{14}$$

with $v(t)$ a function to be detailed later that provides robustness. Using this equation, the structure of the proposed controller can be derived :

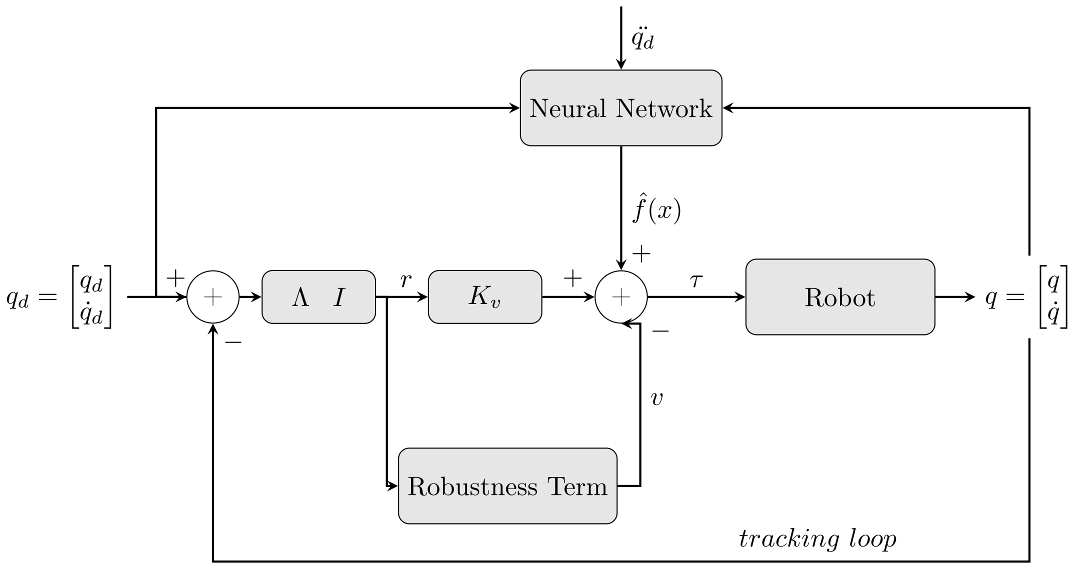

Figure 4: Neural Netowrk-based control scheme.

This control structure is actually the same that can be implemented for an adaptive control strategy, simply replacing the NN with an adaptive control term. However, the similarities end here, since the adaptive control technique is based on very strong assumptions, as the linearity in the unknown parameters (LIP) and the knowledge of the dynamic model of the system to be controlled, that must be fulfilled in order to have good performance in terms of zeroing the tracking error.

2.2 Ideal Case - Unsupervised Backpropagation Tuning of Weights

For the implementation of the first backpropagation technique, let the desired trajectory $q_d(t)$ be bounded by q_B . Assume that the disturbance term $w_1(t)$ in (19) is zero. Let the control input be defined with $v(t) = 0$ and the weight tuning provided by

$$\dot{\hat{W}} = F\hat{\sigma}r^T \tag{15}$$

$$\dot{\hat{V}} = Gx(\hat{\sigma}'^T \hat{W} r)^T \tag{16}$$

with any constant positive definite design matrices F,G. Then the tracking error $r(t)$ goes to zero with t and the weight estimates $\hat{V}$, $\hat{W}$, are bounded.

Proof:

Define the Lyapunov function candidate

$$L(r, \tilde{W}, \tilde{V}) = \frac{1}{2} r^T M(q) r + \frac{1}{2} tr\{\tilde{W}^T F^{-1} \tilde{W}\} + \frac{1}{2} tr\{\tilde{V}^T G^{-1} \tilde{V}\} \tag{17}$$

Differentiating yields

$$\dot{L} = r^T M(q) \dot{r} + \frac{1}{2} r^T \dot{M} r + tr\{\tilde{W}^T F^{-1} \dot{\tilde{W}}\} + tr\{\tilde{V}^T G^{-1} \dot{\tilde{V}}\} \tag{18}$$

whence substitution from the closed-loop error system equation (all mathematical steps required to derive (19) are provided in [1], on page 199, in equations 4.3.19-4.3.22):

$$M\dot{r} = -(K_v + V_m) r + \tilde{W}^T \hat{\sigma} + \hat{W}^T \hat{\sigma}' \tilde{V}^T x + w_1 + v \tag{19}$$

(with $w_1 = 0$, $v = 0$) yields

$$\dot{L} = -r^T K_v r + \frac{1}{2} r^T (\dot{M} - 2V_m) r + tr\{\tilde{W}^T (F^{-1} \dot{\tilde{W}} + \hat{\sigma} r^T)\} + tr\{\tilde{V}^T (G^{-1} \dot{\tilde{V}} + x r^T \hat{W}^T \hat{\sigma}')\} \tag{20}$$

The skew symmetry property makes the second term zero, and since $\hat{W} = W - \tilde{W}$ with W constant, so that $d\tilde{W}/dt = -d\hat{W}/dt$ (and similarly for V), the tuning rules yield

$$\dot{L} = -r^T K_v r \tag{21}$$

Since $L > 0$ and $L \leq 0$ this shows stability in the sense of Lyapunov so that r, $\tilde{V}$ and $\tilde{W}$ are bounded. Boundedness of r guarantees the boundedness of e and $\dot{e}$, whence boundedness of the desired trajectory shows q, $\dot{q}$, x are bounded. Now, $\ddot{L} = -2r^T K_v \dot{r}$, and the boundedness of $M^{-1}(q)$ and of all signals on the right-hand side of (19) verify the boundedness of $\dot{r}$ and hence $\ddot{L}$, and thus the uniform continuity of $\dot{L}$. This allows one to invoke Barbalat's Lemma to conclude that $\dot{L}$ goes to zero with t, and hence that $r(t)$ vanishes.

A brief summary of this control strategy is then presented in the following table:

Table 1: Two-Layer NN Controller for Ideal Case

Control Input: $\tau = \hat{W}^T \sigma(\hat{V}^T x) + K_v r - v,$
NN Weight/Threshold Tuning Algorithms: $\dot{\hat{W}} = F\hat{\sigma} r^T,$ $\dot{\hat{V}} = Gx(\hat{\sigma}'^T \hat{W} r)^T,$
Design parameters: F, G positive definite matrices.

2.3 Augmented Backprop Tuning for the General Case

The previous backpropagation technique is sufficient only when the disturbance $w_1(t)$ is equal to zero. This requires no NN estimation errors ε and no robot arm disturbances $\tau_d(t)$. These are serious restrictions that never hold in practical situations.
This subsection contains the main result of this project. We have just seen that backprop tuning can only be guaranteed to work in an unrealistic ideal case. To confront the stability and tracking performance of a NN robot arm controller in the thorny general case that allows NN estimation errors and system disturbances, we shall require: (1) the modification of the weight tuning rules, and (2) the addition of a robustifying term $v(t)$.

Let the robustifying term be :

$$v(t) = -K_z(\|\hat{Z}\|_F + Z_B)r \tag{22}$$

with $K_z > C_2$, where C_2 is related to the bounds on the Disturbance Term (Lemma 4.3.2 of [1]), and Z_B an upper bound of the Frobenius Norm $\|\hat{Z}\|_F$ of the so defined matrix :

$$Z \equiv \begin{bmatrix} W & 0 \\ 0 & V \end{bmatrix}.$$

Defining the weight tuning rules as :

$$\dot{\hat{W}} = F\hat{\sigma}r^T - F\hat{\sigma}'\hat{V}^T x r^T - \kappa F\|r\|\hat{W} \tag{23}$$

$$\dot{\hat{V}} = Gx(\hat{\sigma}'^T\hat{W}r)^T - \kappa G\|r\|\hat{V} \tag{24}$$

with any constant matrices $F = F^T > 0$, $G = G^T > 0$, and $\kappa > 0$ a small scalar design parameter, it can be demonstrated that the filtered tracking error $r(t)$ and NN weight estimates $\hat{V}$, $\hat{W}$ are uniform ultimate boundedness (UUB).

Proof:

Starting from Lyaponov function defined in (17), after differentiation and substitutions with the weight tuning rules (23) and (24) one can obtain :

$$\dot{L} = -r^T K_v r + \kappa\|r\| \operatorname{tr}\left\{\tilde{Z}^T(Z - \tilde{Z})\right\} + r^T(w + v). \tag{25}$$

After some algebraic manipulations presented in equation 4.3.39 on page 204 of [1] it can be demonstrated that :

$$\dot{L} \leq -\|r\| \left\{K_{v_{\min}}\|r\| - \kappa \cdot \|\tilde{Z}\|_F(Z_B - \|\tilde{Z}\|_F) - C_0 - C_1\|\tilde{Z}\|_F\right\} \tag{26}$$

$\dot{L}$ is then negative as long as the term in braces is positive. Defining $C_3 = Z_B + C_1/\kappa$ and substituting in (26), one obtain that the quantity in the braces of (26) is positive as long as

$$\|r\| > \frac{C_0 + \kappa C_3^2/4}{K_{v_{\min}}} \tag{27}$$

or

$$\|\tilde{Z}\|_F > C_3/2 + \sqrt{C_0/\kappa + C_3^2/4} \tag{28}$$

where $K_{v_{\min}}$ is the minimum singular value of K_v , as defined in eq. 4.3.34 on page 203 of [1].
Thus, $\dot{L}$ is negative outside a compact set. Therefore, we can conclude that this demonstrates the UUB of both $\|r\|$ and $\|\tilde{Z}\|_F$ as long as the control remains valid within this set.

A brief summary of this control strategy is then presented in the following table:

Table 2: Two-Layer NN Controller with Augmented Backprop Tuning

Control Input:

$$\tau = \hat{W}^T \sigma(\hat{V}^T x) + K_v r - v,$$

Robustifying Signal:

$$v(t) = -K_z(\|\hat{Z}\|_F + Z_B)r$$

NN Weight/Threshold Tuning Algorithms:

$$\dot{\hat{W}} = F\hat{\sigma}r^T - F\hat{\sigma}'\hat{V}^T x r^T - \kappa F\|r\|\hat{W},$$
$$\dot{\hat{V}} = Gx(\hat{\sigma}'^T \hat{W} r)^T - \kappa G\|r\|\hat{V},$$

Design parameters: F, G positive definite matrices, $\kappa > 0$ a small design parameter.

3 Simulink Implementation

The implementation and simulation of these control strategies is performed using the software MATLAB/Simulink and the Peter Corke's Robotics Toolbox, particularly useful since it provides many functions useful for the study and simulation of classical arm-type robotics.

3.1 Neural Network with Unsupervised Backpropagation Tuning of Weights

The control scheme implementing the Neural Network Motion Control in the ideal case is shown in Figure 5 :

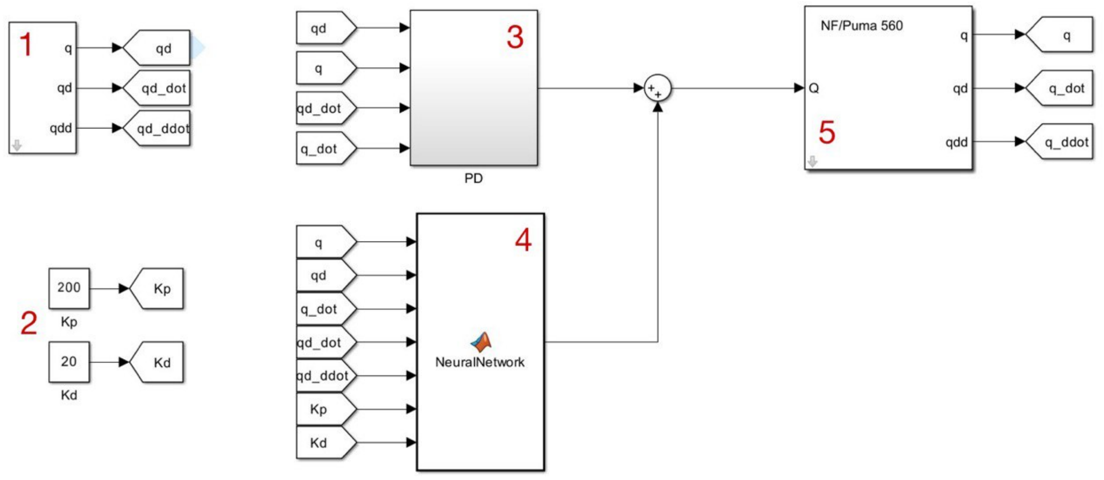

Figure 5: Control Scheme with Neural Network with Unsupervised Backpropagation Tuning of Weights.

As illustrated, the scheme is divided into five components:

- Component Number 1 : This is the *jtraj* block implemented in the Peter Corke's Robotics Toolbox. It allows to generate quintic polynomial to move from initial to final joint angles as specified, with zero initial and final velocity;
- Component Number 2 : These two blocks are simply used to set the proportional and derivative gains for the PD controller;
- Component Number 3 : This block implements the standard PD controller, multiplying K_p and K_d respectively for the error and its derivative;
- Component Number 4 : This block implements a MATLAB function, which will be shown and explained later, containing the implemented control algorithm;
- Component Number 5 : This is the *Robot* block implemented in Peter Corke Robotics Toolbox. It allows to simulate the forward rigid-body dynamics of the robot described by the given robot object. From a simulation point of view, for given applied joint torque/force Q it computes the instantaneous joint coordinates, velocities, and accelerations.

Performing a comparison between the implemented control scheme in Figure 5 and the theoretical one shown in Figure 4, two main differences can be observed:

- Absence of the Robustness Term : this arises from dealing with the ideal case, in which there isn't the generation of the robustness term $v(t)$;

- Presence of the PD Controller : this controller is actually the same implemented in the Figure 4, since the proportional gain K_p is equal to $K_v\Lambda$ and the derivative gain K_d is equal to K_v.

In order to clarify the control technique, the MATLAB code of the MATLAB function used in the control scheme is provided.

Code 1: Neural Network with Unsupervised Backpropagation Tuning of Weights.

```matlab
function tau = NeuralNetwork(q, qd, q_dot, qd_dot, qd_ddot,Kp,Kd)
    e = qd - q;
    e_dot = qd_dot - q_dot;
    Lambda = Kp/Kd * eye(6);
    r = Lambda*e + e_dot;

    x = [1; e; e_dot; qd; qd_dot; qd_ddot];

    persistent W V
    if isempty(W) || isempty(V)
        state_dim = length(q);
        hidden_dim = 5 * state_dim;
        W = randn(hidden_dim+1, state_dim) * 0.1;
        V = randn(length(x), hidden_dim) * 0.1;
    end

    F = 0.1;
    G = 0.01;

    activation = @(x) tanh(x);
    activation_derivative = @(x) 1 - tanh(x).^2;

    sigma = activation(V' * x);
    f_hat = W' * [1; sigma];

    tau = f_hat;

    sigma_prime = diag(activation_derivative(V' * x));
    V = V + G * (x * (sigma_prime * (W(2:end, :)* r))');
    W = W + F * ([1; sigma] * r');
end
```

The function generates the NN control torque *tau* according to the equation (14). After defining the tracking error e and its derivative, the diagonal matrix Λ, the filtered error r and the states variables are declared. It is important to note that the state vector includes an additional row with a unitary value, which has been introduced to account for the offset, as explained in Section 1.1.1. Then, the weight matrices V and W are initialized with small values and the learning rates F and G are set. Note that W and V are declared as *persistent* : this is useful for declaring variables that keep their value between subsequent calls to a function, as in this case in which the function is continuously recalled during the simulation. Additionally, the number of hidden neurons is set to five times the degrees of freedom of the manipulator, as also suggested in [1], to ensure a good trade-off between computational cost and performance.
After this part, the activation function is declared and the control action *tau* is generated. The chosen activation function is the hyperbolic tangent, this choice is motivated by the fact that the hyperbolic tangent provides a nonlinear and bounded output, with a more pronounced response to small signals compared to the sigmoid function. Additionally, since it is centered at zero, it facilitates the convergence of the learning algorithm by avoiding unbalanced weight updates and improving gradient propagation.
The function ends with the tuning rules of the weight matrices W and V, as described in the equations (15) and (16).

3.1.1 Tracking trajectory performances evaluation

The aim is for the manipulator robot to follow a reference trajectory. We focused on a particular trajectories for each joint, generated in the joint space, that bring from an upward elbow initial configuration to a fully extended arm downward configuration, as shown in Figure 6 and Figure 7. In order to go from the initial to the final position, as will be shown lately, is sufficient to control only three joints.

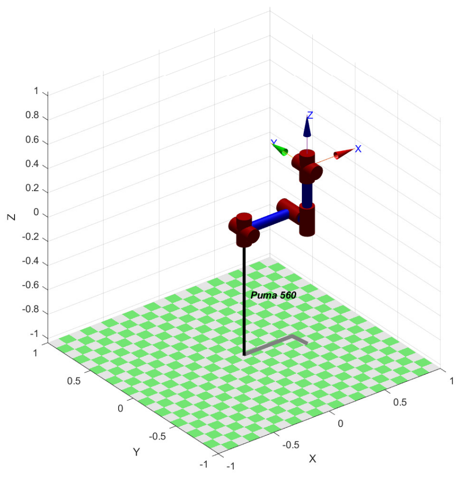

Figure 6: Initial configuration: $q_0 = [0\ 0\ 0\ 0\ 0\ 0]$.

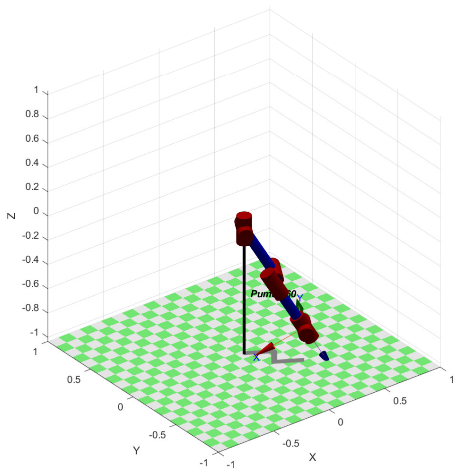

Figure 7: Final configuration: $q_f = [-\frac{\pi}{6}\ -\frac{\pi}{3}\ -\frac{\pi}{2}\ 0\ 0\ 0]$.

Referring to Figure 8, one can observe the dashed desired joint trajectories and the actual ones performed by the manipulator.
As mentioned before, controlling three joints is sufficient to bring the manipulator from the initial

to final configuration : for this reason one can focus only on the implemented trajectory for the three controlled joints, since for the other three the reference trajectories are always zero in time.

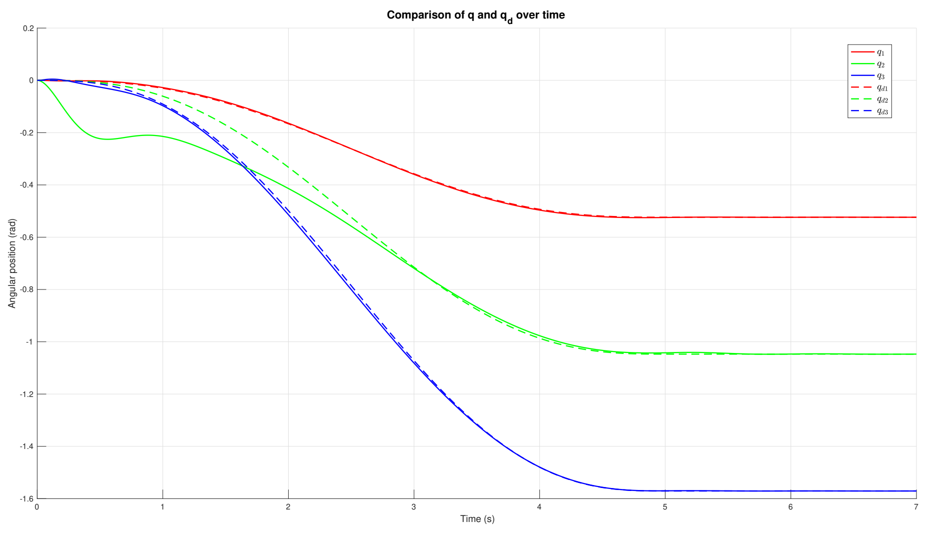

Figure 8: Desired and actual joints trajectories evolution.

Observing the dashed desired and actual trajectories for each joint, one can firstly notice that, in the steady-state condition, there is an excellent tracking of all trajectories, as seen in the theoretical part.

Focusing on the first part of the simulation, one can notice that there is not a proper fit of the desired and actual trajectory, especially referring to the joint 2. However, this is an expected behavior of the control system, related to the implementation of the neural network, which takes a certain amount of time to set correctly the weights V and W weights.
A control performance improvement, from the time needed for the overlapping of trajectories point of view, can be achieved by increasing the learning rate G of the equation (16). However, this increase would lead to an increase in the oscillating behavior of the control action, resulting in additional stresses on the motors of the joints.
By acting on proportional gain, as will be explained in the section 3.1.2, it is also possible to obtain better performance.

Additionally, as will be also clear in the section 3.1.2, in the first part of the simulation is the PD controller that generates most of the control action, thus avoiding the loss of reference.

The performances of the controller are also interpretable by observing the Figure (9), in which the tracking error for each joint is plotted. The plot is easy to understand if compared with the Figure (8), since the information are more or less the same, but from different point of view : in the first part of the simulation there is a bigger tracking error, whose maximum value is about 0.21 rad, due to the PD's control action alone, when in steady-state condition the error is zero.
Acting on the proportional gain, one could have smaller initial error, but this would led to an important increase of the required performance of the joints motors.
Actually, very large values of proportional gains would make the PD controller sufficient to carry out the control. However, these values would be totally unrealistic and not implementable in reality.

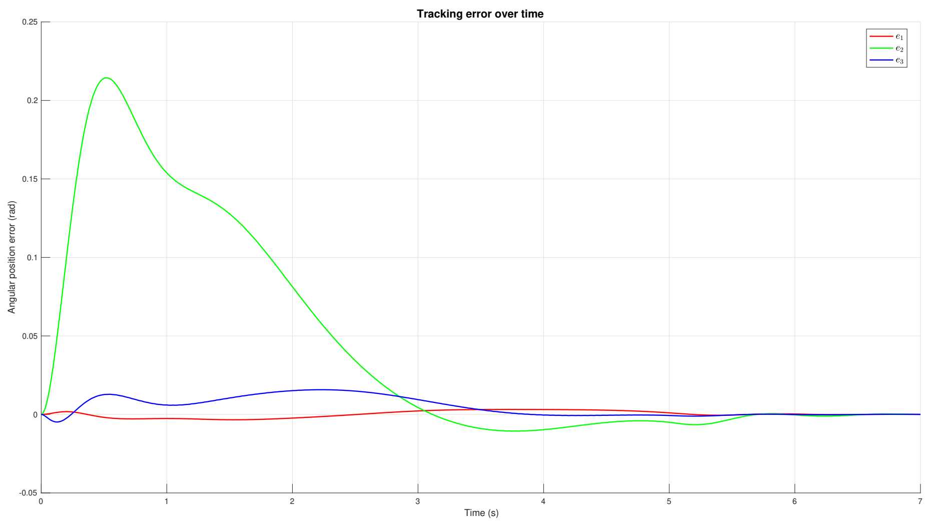

Figure 9: Tracking error evolution.

3.1.2 Control action evaluation

For a deeper and complete understanding of the control technique, an evaluation of the evolution of the control actions is necessary. Referring to the Figure (10), one can focus on the PD controller and Neural Network contributions on the total torque computed, that is simply their sum.

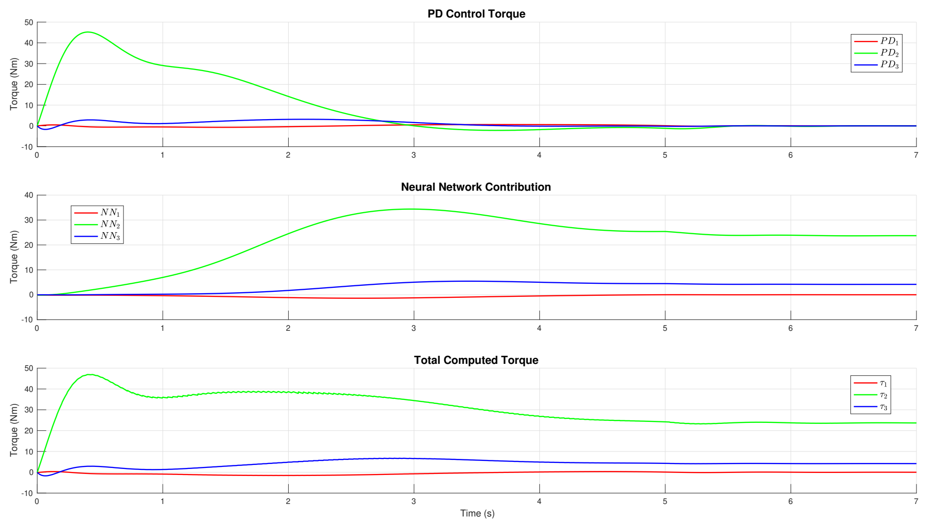

Figure 10: Control actions evolution.

By observing the first two plots, you can divide each one in two parts : a first one, covering about the first second, and the second one from that time instant. It is evident that in the first part of the simulation the PD control almost entirely generates the total control action, while the contribution of the NN to the total control torque is almost zero. This regards the learning strategy of the NN:

the NN weights are initialized, as in Code 1, at very small value in order to avoid going into the saturation zone of sigmoid activation functions. This means that there is no off-line learning phase, but NN learning occurs in real-time. For this reason, in the first part of the simulation, the tracking performance are provided only by the PD controller control action, while the NN starts the learning phase. Once the NN starts to learning, one can notice that Neural Network allows you to track the desired trajectories, without knowing the model of the manipulator and its dynamics, but based only on that generated online.

An important consideration about the PD control action is required. The initial evolution, in terms of slope, is closely linked to the value of the chosen proportional gain and has important consequences passing from a simulation phase to a real implementation.
By increasing the proportional gain value, one can obtain smaller maximum values of the tracking error and a faster control on terms of hooking up the desired trajectories. However, this would led to an increase of the initial slope of the PD control action. It is therefore necessary, in a real implementation, to have joints motors that really allow to generate torques having a that slope.
In case of available joints motors not able to generate that torque slope, it is necessary to decrease the value of proportional gain. This would of course lead to an increase in the tracking error in the first phase of the control and to an increase in the time needed to hook up the reference.
In conclusion, it is therefore necessary to choose the values of the proportional and derivative gains according to the specific joints motors available.

3.2 Neural Network with Augmented Backprop Tuning for the General Case

The Simulink model used to implement the control based on Neural Network with Augmented Backprop Tuning for the General Case is as follows:

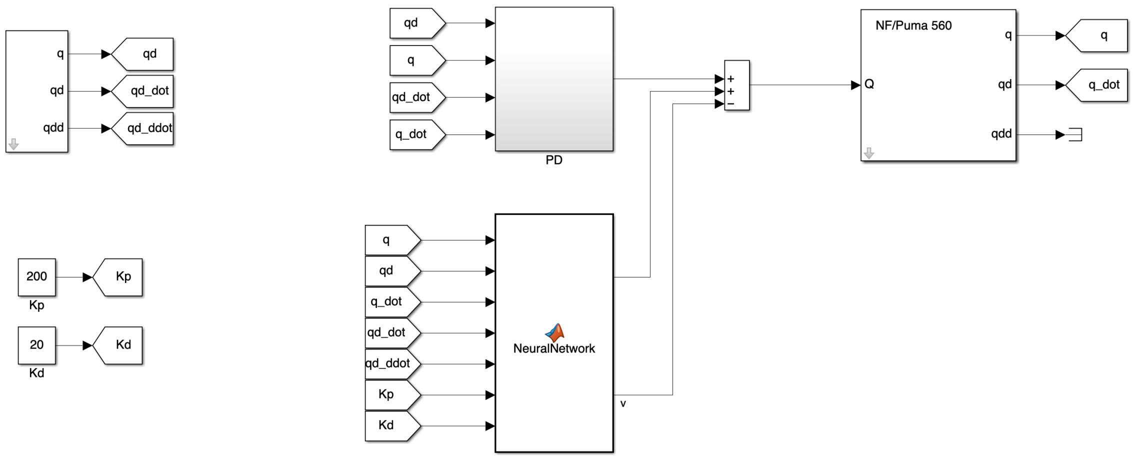

Figure 11: Control scheme with Neural Network with Augmented Backprop Tuning.

As observed in the control scheme in Figure 11, the structure is very similar to that in Figure 5. The only modified block is the Matlab Function, which implements the Neural Network with Augmented Backprop Tuning and computes the robustifying term $v(t)$.

The MATLAB code that implements the neural network and the calculation of the robustifying term is shown in Code 2:

Code 2: Neural Network with Augmented Backprop Tuning for the General Case.

```
function [tau,v] = NeuralNetwork(q, qd, q_dot, qd_dot, qd_ddot,Kp,Kd)
    e = qd - q;
    e_dot = qd_dot - q_dot;
    Lambda = Kp/Kd * eye(6);
    r = Lambda*e + e_dot;

    x = [1; e; e_dot; qd; qd_dot; qd_ddot];
    state_dim = length(q);
    hidden_dim = 5 * state_dim;

    ki = 0.10;

    persistent W V
    if isempty(W) || isempty(V)
        W = randn(hidden_dim+1, state_dim) * 0.1;
        V = randn(length(x), hidden_dim) * 0.1;
    end

    F = 0.1;
    G = 0.4;

    activation = @(x) tanh(x);
    activation_derivative = @(x) 1 - tanh(x).^2;

    sigma = activation(V' * x);
    f_hat = W' * [1;sigma];

    tau = f_hat;

    sigma_prime = diag(activation_derivative(V' * x));
    V = V + G * (x * (sigma_prime * (W(2:end, :) * r))') -ki*G*norm(r)*V;
    W = W + F * ([1;sigma] * r') - F*[zeros(1, state_dim); sigma_prime * ...
        (V'*x*r')] - ki*F*norm(r)*W ;

    Z = [W zeros(hidden_dim+1,hidden_dim); zeros(length(x),state_dim) V];

    norm_Zf = norm(Z, 'fro');
    Z_b = 1.1*norm_Zf;

    C2 = 0.07*Z_b;

    Kz = 1.2*C2;
    v = -Kz*(norm_Zf + Z_b)*r;

end
```

The code implementing the neural network using the second backpropagation methodology introduces additional variables compared to the first approach described in Code 1. Among these, the variable "chi" represents a small design parameter that, together with the variables "F" and "G" (which act as learning rates), allows for the regulation of the network's learning speed. This parameter directly influences the rate at which the weights of the matrices "W" and "V" are updated. High values of "chi" can accelerate the learning process but tend to introduce more pronounced oscillations, especially in the final stages of execution. This phenomenon occurs because, although learning becomes faster, small fluctuations in the error can cause significant variations in the weight matrices, leading to larger oscillations in the torque generated by the neural network.

A further distinctive element compared to Code 1 is the introduction of the robustness term $v(t)$, which is added to the control torque to ensure system stability. This term is calculated based on the formula provided in Equation 22.

The values of the variables "Z_b", "C_2", and "K_z" have been determined empirically with the aim of avoiding instability while adhering to the theoretical constraints outlined in Section 2.3 of this documentation and Section 4.3 of the cited text [1]. Specifically, these constraints require that:

- Z_b be greater than or equal to $\|Z_f\|_F$,
- $C2$ be defined as $C_2 = c_7 * Z_b$, where c_7 is a positive constant,
- K_z be greater than C_2.

These criteria ensure that the system maintains stable behavior and remains consistent with the mathematical foundations underlying the neural network implementation.

3.2.1 Tracking trajectory performances evaluation

Even with the neural network implemented using the second backpropagation methodology, the performance in terms of tracking capability proves to be very good. Although modest errors are observed in the initial phase of execution, primarily attributable to the action of the PD controller, Figure 12 shows how, after approximately 3.5 seconds, the error tends to settle at zero. This behavior is due to the neural network's ability to progressively compensate for the effects introduced by the robot's dynamics.

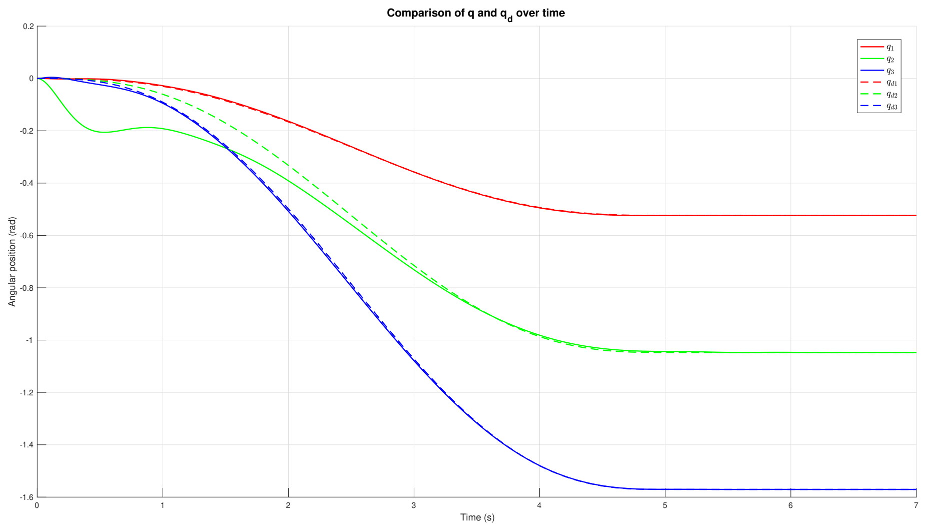

Figure 12: Desired and actual joint trajectories evolution.

For a clear and direct understanding of how the error decreases over time, a graphical representation is provided in Figure 13. This figure illustrates the trend of the error, highlighting its reduction and eventual convergence, thereby offering an intuitive visualization of the system's performance improvement during execution.

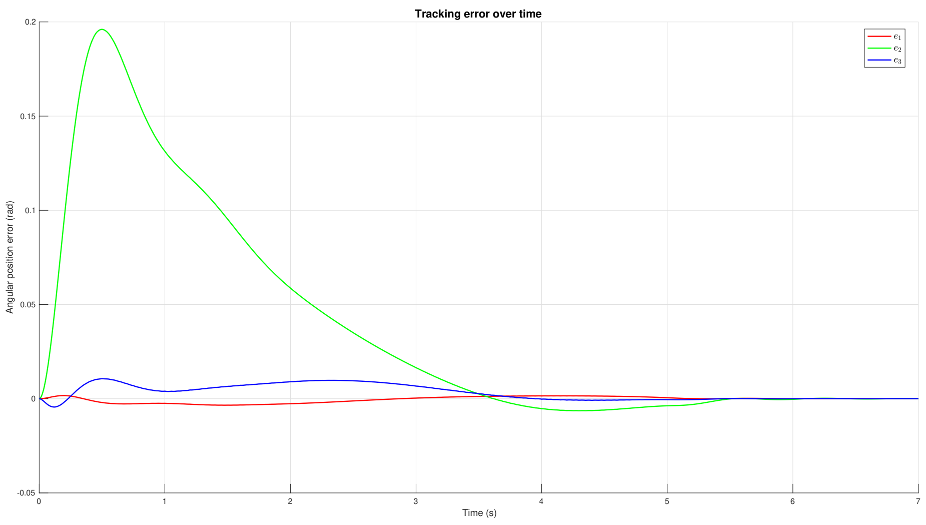

Figure 13: Tracking error evolution.

3.2.2 Control action evaluation

As previously mentioned in Section 3.1.2, even with the second methodology, the graph related to the control action in Figure 14, calculated by the PD controller and the neural network, shows that for the first 2 seconds, the overall control action is almost entirely derived from the PD controller, while for the subsequent 2 seconds, it is predominantly driven by the neural network. However, this case differs due to the presence of the robustness term, which in this instance has a minimal contribution, as we considered a disturbance τ_d equal to zero. Indeed, although the second methodology demonstrates greater speed in driving the error to zero, the performance of the two controllers remains very similar.

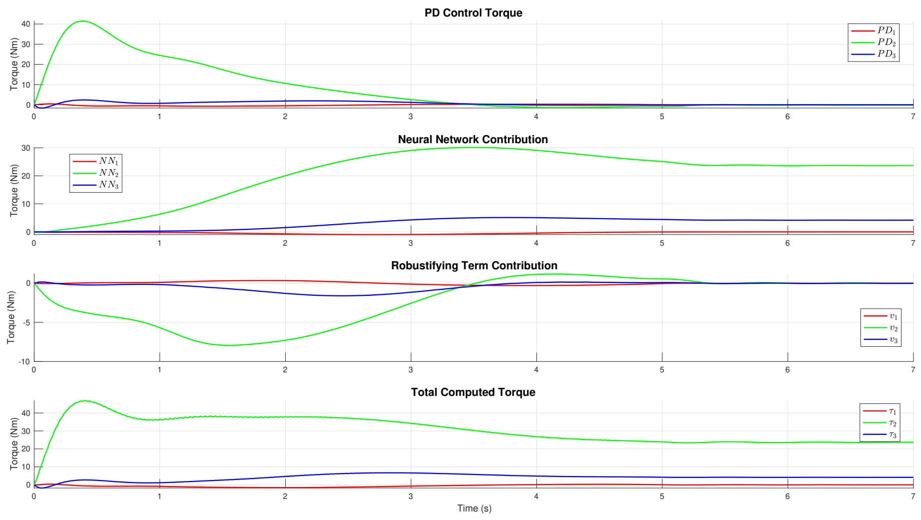

Figure 14: Control actions evolution.

3.3 Comparison of Performance in the Presence of Disturbance

In this subsection, the graphs related to the tracking errors obtained using the two backpropagation techniques for tuning the weights of the neural network matrices in the presence of disturbances will be presented, along with the graphs depicting the control actions computed at each time instant. The aim is to provide a comprehensive comparison of the performance of the two methodologies, highlighting both their tracking accuracy and the dynamic behavior of the control actions under perturbed conditions.

Before proceeding with the presentation of the graphs related to the obtained results, it is important to illustrate the applied disturbance. As shown in the graph in Figure 15, the disturbance is sinusoidal in nature, with a non-arbitrarily chosen amplitude and a frequency of 1 rad/s. This disturbance is applied for a duration of 3 seconds, starting from the 2nd second of the simulation and continuing until the 5th second. This specific choice of disturbance allows for a clear evaluation of the system's robustness and the controllers' ability to mitigate its effects under well-defined conditions.

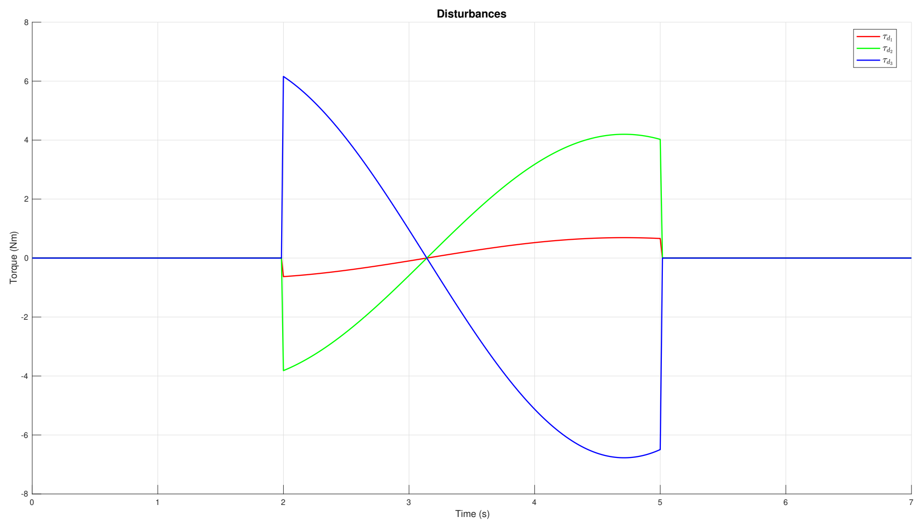

Figure 15: Applied Disturbances.

3.3.1 Tracking trajectory performances evaluation

From the analysis of the graphs presented in Figure 16 and Figure 17, it can be observed that, in the presence of disturbance, the neural network-based controller employing the first backpropagation technique, namely Unsupervised Backpropagation Tuning of Weights, exhibits significantly higher errors compared to the controller using the second methodology, Augmented Backprop Tuning of Weights.

In particular, as highlighted in the graph in Figure 16, during the application of the disturbance, the error in the third joint reaches a peak of approximately 0.04 rad (2.3 degrees). In contrast, the controller utilizing the second backpropagation methodology (Figure 17) demonstrates greater robustness, with a maximum error recorded in joint 3 of about 0.02 rad (1.15 degrees), which is roughly half that of the first case.

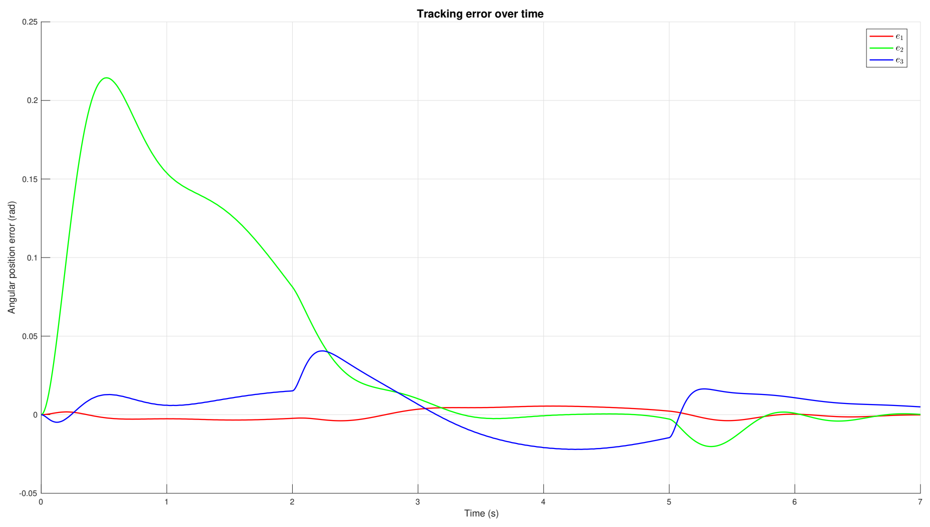

Figure 16: Tracking error with the first NN in the presence of disturbance.

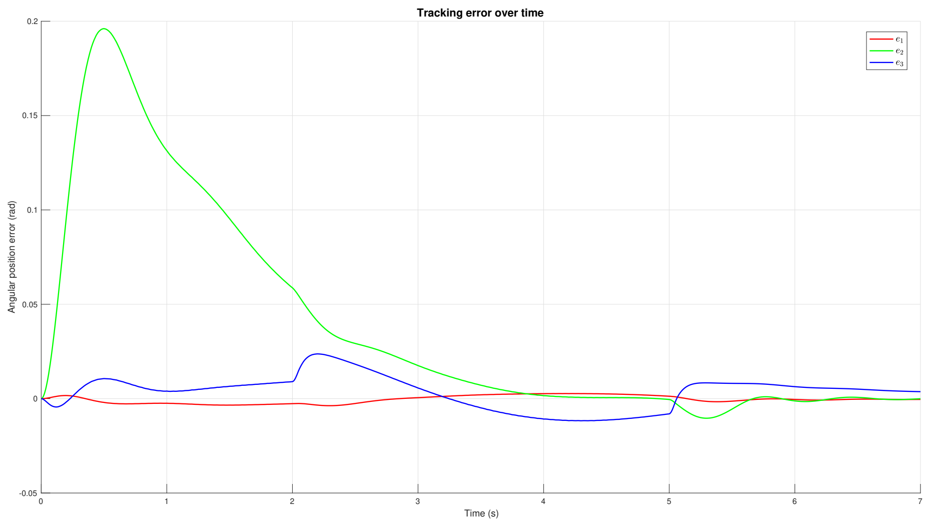

Figure 17: Tracking error with the second NN in the presence of disturbance.

Once the disturbance is removed, both controllers show the ability to return the error to values close to zero, indicating stable system convergence.

3.3.2 Control action evaluation

By comparing the control actions, it is evident that the neural network-based controller employing the first backpropagation technique is able to reject disturbances primarily due to the action of the PD controller. This implies that if the PD gains are chosen to be lower, the errors would increase significantly. This demonstrates that the neural network using the first backpropagation technique lacks robustness, as it is not inherently capable of rejecting disturbances effectively. In contrast, the second neural network, which utilizes the second backpropagation technique, incorporates a dedicated robustness term $v(t)$, specifically designed to reject disturbances. As a result, the disturbance rejection capability of this controller does not strictly depend on the chosen PD gains, which, as mentioned in Section 3.1, can vary depending on the specific case.

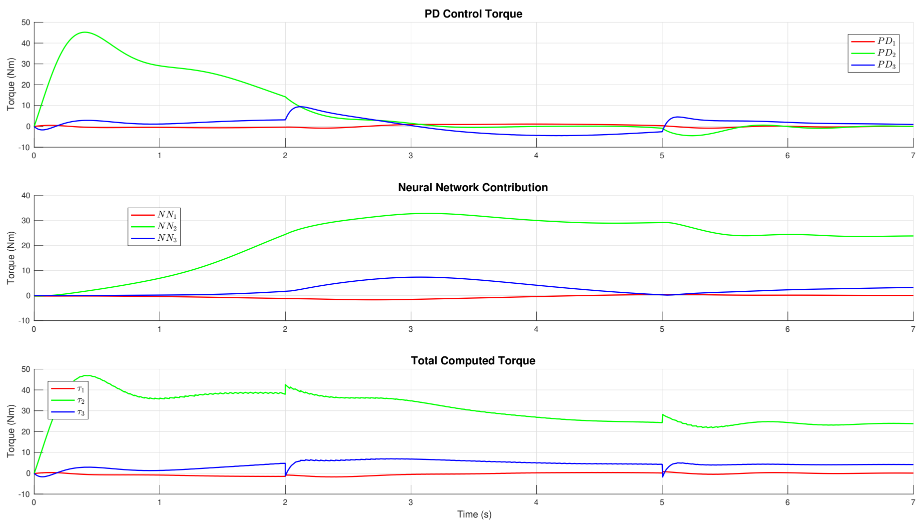

Figure 18: Control actions with the first NN in the presence of disturbance.

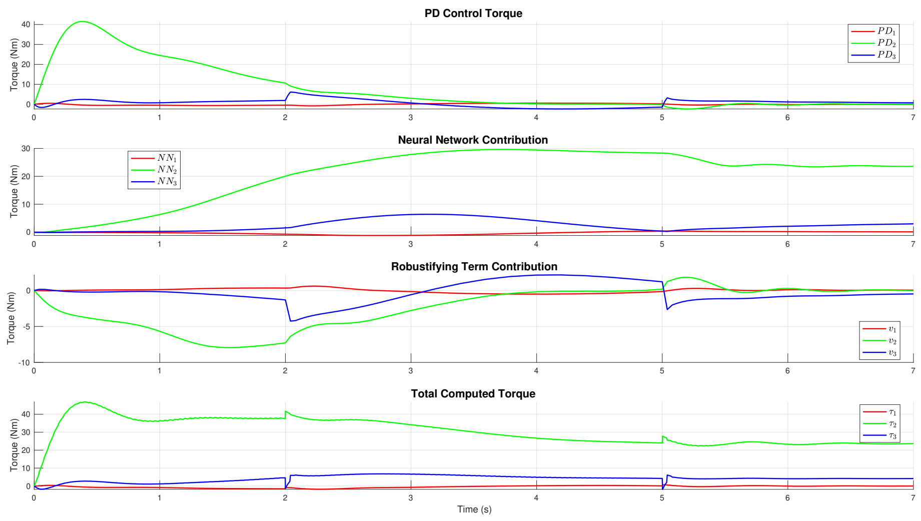

Figure 19: Control actions with the second NN in the presence of disturbance.

In conclusion, the difference in performance during the application of the disturbance highlights the greater effectiveness of the Augmented Backprop Tuning of Weights technique in mitigating the effects of external perturbations, ensuring improved precision and control stability. This approach provides a more robust and reliable control solution, independent of the PD gain selection, making it better suited for handling uncertainties and disturbances in dynamic systems.

3.4 Performance Analysis in the Absence of Neural Network

In this subsection, the results related to the tracking capability will be presented, comparing the performance of a controller based solely on a PD regulator without a neural network and the motion control technique based on inverse dynamics.

3.4.1 PD control without Neural Network

In Figure 20, the results related to the tracking capability of a PD controller without a neural network for centralized control of the Puma 560 manipulator are presented. It can be observed that the PD controller is unable to achieve efficient control, as it exhibits a steady-state position error of approximately 0.1 rad (5.73 degrees) in the second joint.

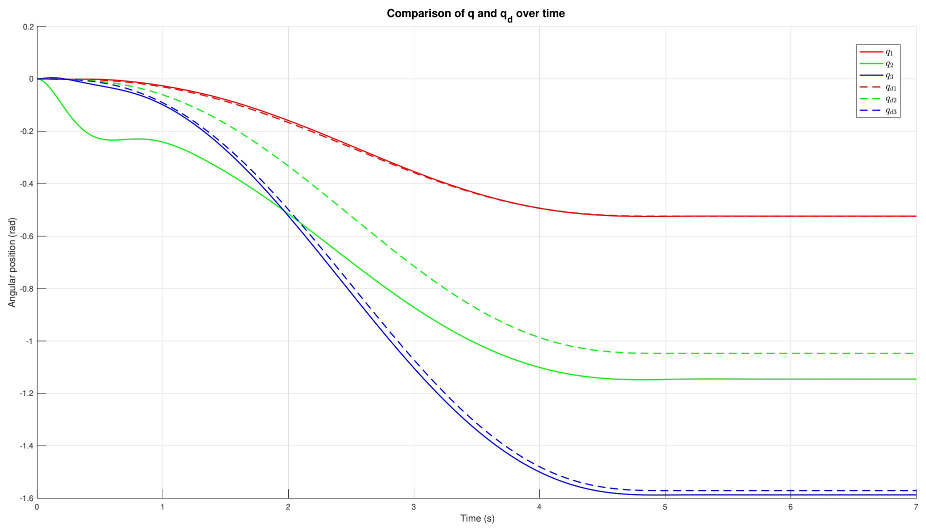

Figure 20: Desired and actual joint trajectories with PD.

This highlights the crucial role of the neural network in the manipulator's control. Without it, the PD controller alone is insufficient, likely due to its inability to fully compensate for the dynamic effects introduced by the manipulator, such as gravity, Coriolis forces, and inertial variations. These effects introduce nonlinearities and uncertainties that a simple PD controller cannot adequately address, leading to persistent errors in the system's response.

3.4.2 Inverse dynamics control

The final test conducted concerns motion control using inverse dynamics. This technique is based on feedback linearization, allowing perfect compensation of the dynamic effects introduced by the system, under the assumption that the inertia matrix, Coriolis forces, and other dynamic parameters are perfectly known.

Since inverse dynamics control is primarily a theoretical approach, as it requires exact knowledge of all the matrices that define the manipulator's dynamics, a simulation was performed in which the masses of the manipulator's links were slightly modified.

The masses of the manipulator's links were modified as follows: the mass of link 1 was decreased by 0.2 kg, the mass of link 2 was decreased by 0.1 kg, the mass of link 3 was increased by 0.5 kg, the mass of link 4 was increased by 0.1 kg, the mass of link 5 was increased by 0.6 kg, and the mass of link 6 was increased by 0.4 kg. These modifications represent small variations that can be interpreted as measurement uncertainties in the model parameters.

The results (Figure 21) indicate that even slight changes in the mass values prevent the manipulator from accurately tracking the reference trajectory, leading to a nonzero steady-state error. This further confirms that inverse dynamics control lacks robustness, as even minor inaccuracies in the system model introduce persistent errors in the control performance.

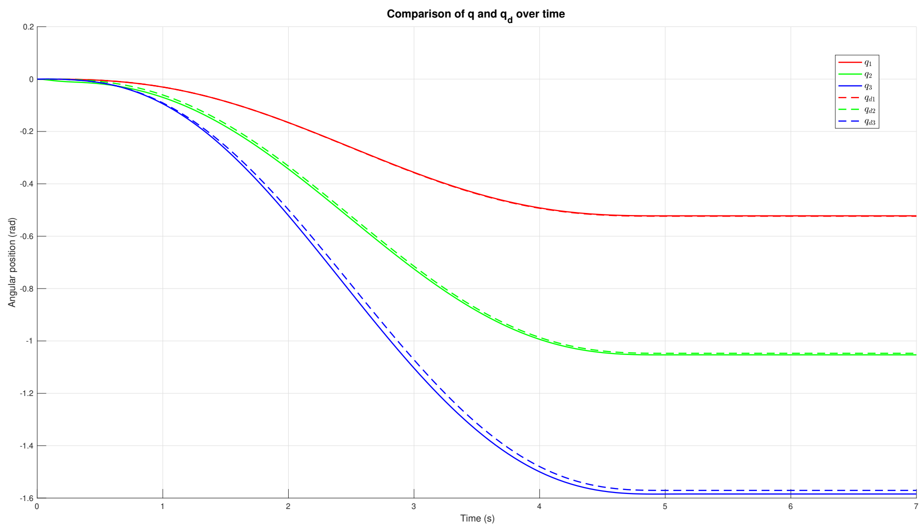

Figure 21: Desired and actual joint trajectories with inverse dynamics control.

4 Conclusions

In this study, a Neural Network-based control system for a six degrees of freedom robot manipulator was designed. The implementation and simulation of the control system were made using the software MATLAB and Simulink, as well as the performances' evaluation.

This control strategy offers several advantages over classical control techniques, such as inverse dynamics control or adaptive control. In particular, the NN-based controller overcomes the limits that characterize the other types of control, namely the need to know exactly the dynamic model of the system and/or its characteristic parameters. Without knowing the dynamic model of the system you want to control, the neural network-based controller learns in real time through precise learning strategies by building a model very close to the real model of the system you want to control.

However, the NN-based control approach also presents certain challenges. One key drawback is the necessity of fine-tuning the network's parameters on a case-by-case basis. This implies that the same set of parameters cannot be universally applied to all manipulators; rather, they must be carefully selected while adhering to the mathematical constraints imposed by control theory. Additionally, in a PD-based control system with an online learning neural network, an initial tracking error potentially significant must be tolerated. This occurs because the neural network updates its weights in real-time and requires a certain adaptation period to approximate the system's dynamic behavior. This adaptation time can be further extended when computational resources are limited, as resource constraints may necessitate a reduction in the number of neurons used in the network.

A detailed analysis of the NN-based controller's performance, implemented with two different weight updating strategies, showed that this control strategy allows to follow faithfully the desired trajectory, even in case of presence of disturbances.
Comparative analysis with other control techniques have shown that NN-based control is much more efficient and less subject to various constraints.

These analysis provide solid basis for future developments, concerning the optimization of the control algorithm, the implementation of new tuning weights strategies and, of course, the transition from a simulation environment to a real one, adapting the controller's performance to the real available resources in the application context.

References

[1] F.L. LEWIS, S. JAGANNATHAN, and A. YES ¸ILDIREK. *Neural Network Control of Robot Manipulators and Nonlinear Systems*. Taylor&Francis, 1999.