micromachines

Article

Implementation of ANN-Based Auto-Adjustable for a Pneumatic Servo System Embedded on FPGA

Marco-Antonio Cabrera-Rufino *,†, Juan-Manuel Ramos-Arreguín *,†, Juvenal Rodríguez-Reséndiz †, Efren Gorrostieta-Hurtado † and Marco-Antonio Aceves-Fernandez †

Facultad de Ingeniería, Universidad Autónoma de Querétaro, Cerro de las Campanas, Las Campanas, Queretaro 76010, Mexico; juvenal@uaq.edu.mx (J.R.-R.); efrengorrostieta@gmail.com (E.G.-H.); marco.aceves@gmail.com (M.-A.A.-F.)

* Correspondence: mcabrera23@alumnos.uaq.mx (M.-A.C.-R.); jsistdig@yahoo.com.mx (J.-M.R.-A.); Tel.: +52-558-057-8823 (M.-A.C.-R.); +52-442-250-0031 (J.-M.R.-A.)

† These authors contributed equally to this work.

Abstract: Artificial intelligence techniques for pneumatic robot manipulators have become of deep interest in industrial applications, such as non-high voltage environments, clean operations, and high power-to-weight ratio tasks. The principal advantages of this type of actuator are the implementation of clean energies, low cost, and easy maintenance. The disadvantages of working with pneumatic actuators are that they have non-linear characteristics. This paper proposes an intelligent controller embedded in a programmable logic device to minimize the non-linearities of the air behavior into a 3-degrees-of-freedom robot with pneumatic actuators. In this case, the device is suitable due to several electric valves, direct current motors signals, automatic controllers, and several neural networks. For every degree of freedom, three neurons adjust the gains for each controller. The learning process is constantly tuning the gain value to reach the minimum of the mean square error. Results plot a more appropriate behavior for a transitive time when the neurons work with the automatic controllers with a minimum mean error of ±1.2 mm.

Keywords: robot arm; pneumatic actuators; neural network; FPGA; embedded; neuro-PID; control

Citation: Cabrera-Rufino, M.-A.; Ramos-Arreguín, J.-M.; Rodríguez-Reséndiz, J.; Gorrostieta-Hurtado, E.; Aceves-Fernandez, M.-A. Implementation of ANN-Based Auto-Adjustable for a Pneumatic Servo System Embedded on FPGA. *Micromachines* **2022**, *13*, 890. https://doi.org/10.3390/mi13060890

Academic Editor: Aiqun Liu

Received: 16 May 2022
Accepted: 28 May 2022
Published: 31 May 2022

1. Introduction

The employment of pneumatic actuators in the robotic field has great importance in the manufacturing industry and the modern design of controllers [1,2]. Its significant advantages are a low-cost, lightweight, and simple design. The disadvantages are the highly non-linear behavior of the pneumatic actuators due to having the air as a means of generating force, e.g., the flow of air into the valves is not uniform, friction in the joints, and delay in air propagation, among others [3–7].

Artificial neural networks (ANN), fuzzy logic, genetic algorithms, and others are practical when the mathematical model of the dynamic system is highly complex, highly non-linear, or impossible to know due to the lack of knowledge, except for a few known variables [8–13].

Programmable hardware devices, like FPGA, are suitable for hardware implementation of neural networks and PID controllers. They have the advantage of better accuracy, repeatability, and lower noise sensitivity. Also, it is compatible with other types of pre-processors [14–17].

This work aims to position a 3 DoF robot arm with pneumatic actuators and implement an ANN-based auto-adjustable gain tuner for PID controllers. The main contribution is an algorithm proposal to control each DoF of the pneumatic actuators embedded in reconfigurable hardware.

The pneumatic robot arm has six DoF, as presented in Figure 1, where θ_1, θ_2, and θ_3 are the respective angle for each DoF. The moving forces are supplied by one pneumatic motor and two pneumatic pistons.

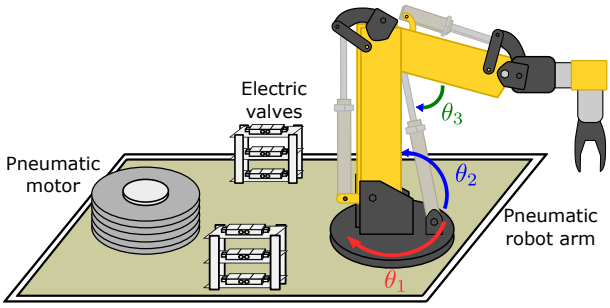

Figure 1. Pneumatic robot prototype with 6 DoF.

To convert a PID design into its VHDL code, we implement Octave software for this work. For example, we set PID gains as integer inputs, and the output is in binary fixed-point format.

Findings

Table 1 shows a brief comparison of the implementations and contributions of state of the art. It mentions the hardware-implemented, and all these works are focused on pneumatic actuators.

Table 1. State of the art.

Ref, Year	Implementation	Contribution
Sánchez, 2017 [18]	A micro-computer with a graphical simulator developed in C++ using OpenGL libraries	A simulated backpropagation artificial neural network controller for a two DoF pneumatic manipulator
Rousbeh, 2018 [6]	PCI-6602 DAQ board and a PC	The design of a controller and its implementation on a position-controlled rotary pneumatic actuator
Humaidi, 2020 [19]	Computer simulation in Matlab and Simulink	The design of a controller based on the Synergetic Control theory concept for a DoF robot arm powered by pneumatic actuators
Lin, 2021 [7]	myRio board and a PC with Labview	A PID controller and a high-order sliding-mode feedback controller for a 3 DoF pneumatic robot manipulator

The prototype architecture proposed has several problems, such as frictions in the joints and the non-linearity inevitably produced by the compressibility of the air. Airflow is not constant and experiences a delay in propagating through the system [3]. The lack of precision could cause catastrophic events that would affect the robot's task and even harm workers [20].

The proposed robot controller remarkably has a slightly more accurate performance, adequately reducing alterations because of the non-linearities of the pneumatic actuators.

The order of this work is as follows: Section 2 deeply describes the materials and methodology, including the hardware description of the PID controller, the configuration of the encoder, and mainly the ANN. Section 3 presents the results obtained for the PID controllers of the motors and the ANN auto-adjustable gains for the pneumatic actuators controller. Section 4 is the discussion of results and their limitations. Finally, Section 5 gives the conclusions of this investigation.

2. Materials and Methods

This section presents the materials and methodology used in this work, such as inverse kinematics, the instrumentation of the encoders, the design of PID controllers and artificial neural networks, and some mathematical tools that helped us develop the before mentioned.

2.1. Materials

In this work, we used 24-VDC and 12-VDC power supplies. An FPGA was the core of this project and also a laptop for data acquisition, Octave script development, and FPGA programming. Eighteen optocouplers isolated the FPGA control signals from the motor drivers to prevent electric noise, and six solid-state relays activated the 12-VDC electro-valves. The motors adjusted the airflow through the pneumatic actuators. The pneumatic electro-valves controlled the direction movement of the links of the robot. Figure 2 illustrates all these components

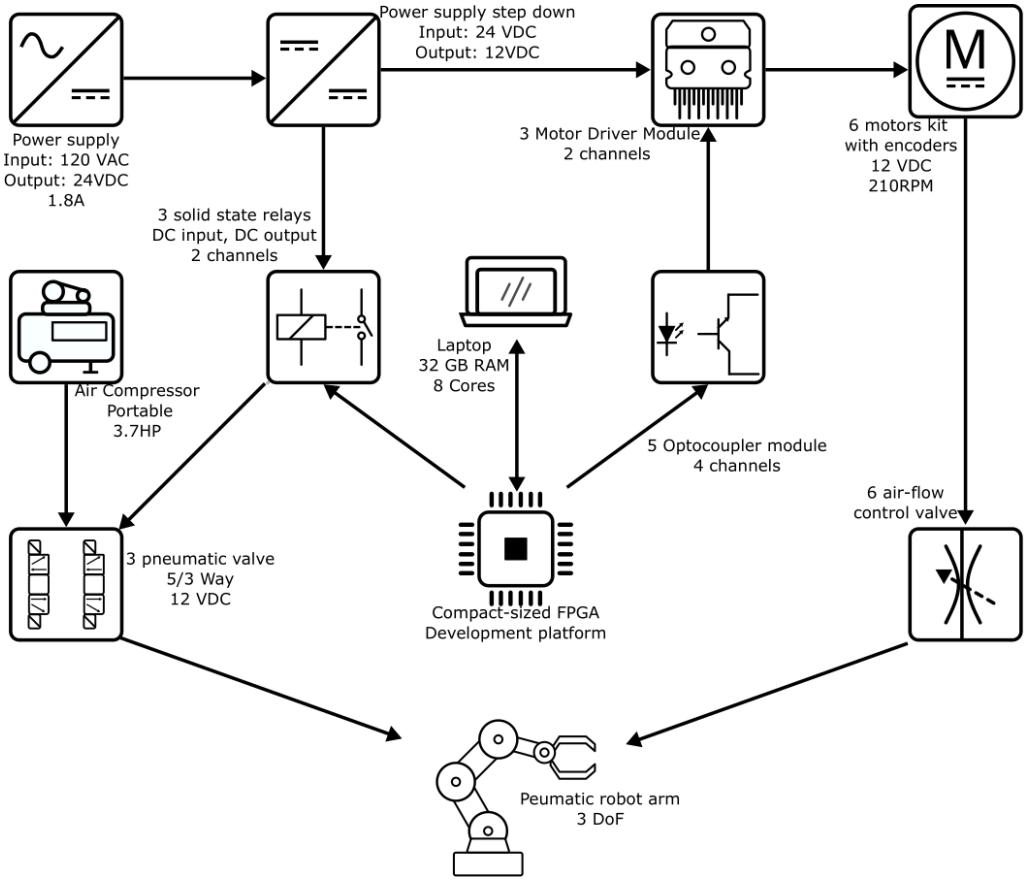

Figure 2. Diagram block of all materials used for this project.

2.2. Inverse Kinematics of a 3 DoF Robot

Figure 3 depicts a geometrical model of a 3-DoF robot, and it provides information to compute the final position $(x_1,\ y_1,\ z_1)$ of the robot [18], where:

- $l_1, l_2, l_3 \rightarrow$ are the links of the robot.
- $\theta_1 \rightarrow$ is the angle of link l_1 with respect to Y axis.
- $\theta_2 \rightarrow$ is the angle of link l_2 with respect to XY axis.
- $\theta_3 \rightarrow$ is the angle of link l_3 with respect to XY axis.

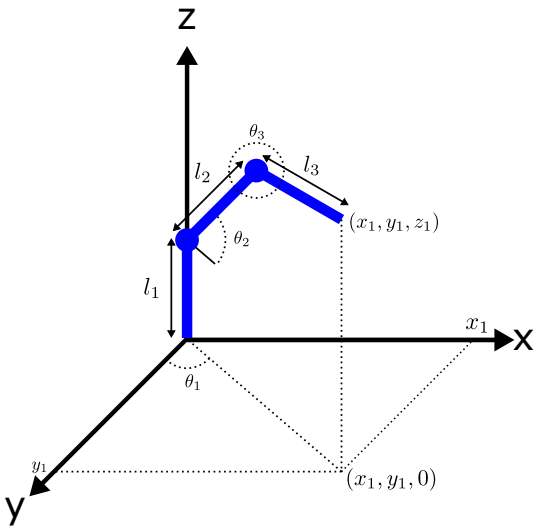

Figure 3. Geometrical model of a 3-DoF robot.

The inverse kinematic calculated the value of the angles θ_1, θ_2, and θ_3 needed to get the final position required $(x_1,\ y_1,\ z_1)$. The Equations (1)–(3) represent the inverse kinematics of the robot [18].

$$\theta_1 = tan^{-1}\left(\frac{y_1}{x_1}\right) \tag{1}$$

$$\theta_2 = cos^{-1}\left(\frac{x^2 + (z - l_1)^2 + l_2^2 - l_3^2}{2l_2\sqrt{x^2 + (z - l_1)^2}}\right) + sin^{-1}\frac{z - l_1}{\sqrt{x^2 + (z - l_1)^2}} \tag{2}$$

$$\theta_3 = \theta_2 + cos^{-1}\left(\frac{l_3^2 + l_2^2 - x^2 - (z - l_1)^2}{2l_3l_2}\right) - 180^\circ \tag{3}$$

2.3. The Pneumatic System

Each DoF consists of one 5/3 electro-pneumatic valve, two airflow valves, two DC motors, and one incremental encoder that allowed us to get the arm position. In Figure 4, the schematic diagram of each DoF is drawn.

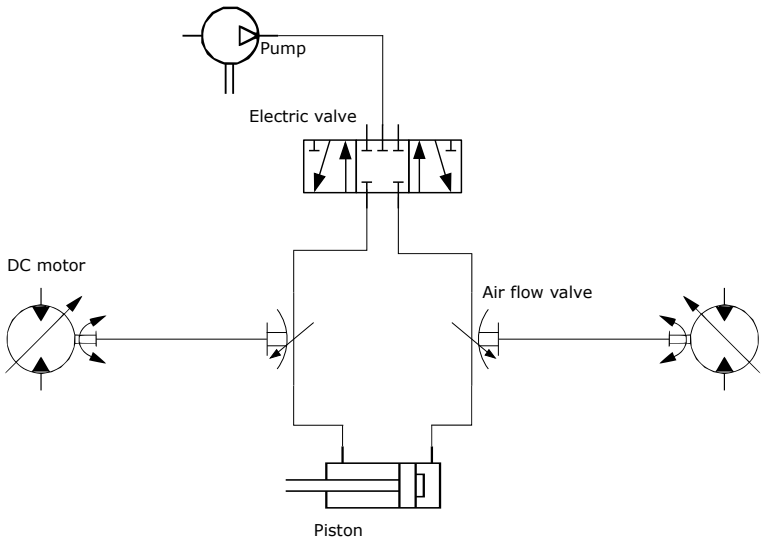

Figure 4. Schematic of one DoF pneumatic system.

2.4. Structure Prototype for Airflow Control

We designed a mechanical base structure to couple the DC Motors to the airflow valves. A bell-type piece was attached to the motor stem. Moreover, it contains an internal hexagonal hole to fit the valve connection. Figure 5 depicts the integration of the mentioned parts.

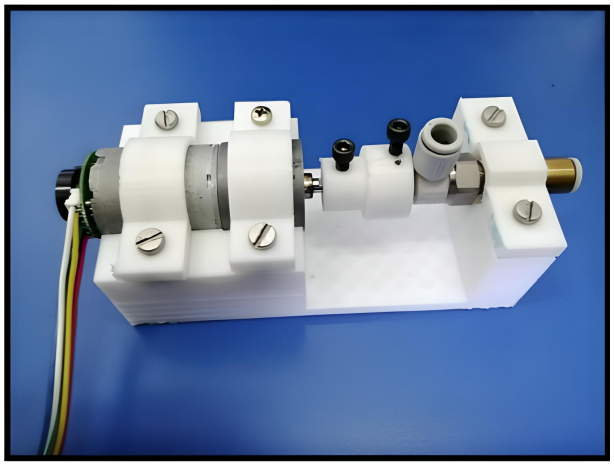

Figure 5. Airflow control system valve with plastic base for DC motors.

2.5. Implemented Algorithm for Binarization of Variables

We developed Octave scripts to write VHDL code. They binarized numeric variables of logic signals, variables and constants, and LUTs faster. More helpful information about this is in [21].

For the binarization process, we used the fixed-point method. A fixed-point number in base-2 format is in (4).

$$(\dots i_2 i_1 i_0 \cdot f_{-1} f_{-2} f_{-3} \dots)_2 \tag{4}$$

which is converted to a decimal number as in Equation (5).

$$\dots i_2 \cdot 2^2 + i_1 \cdot 2^1 + i_0 \cdot 2^0 + f_{-1} \cdot 2^{-1} + f_{-2} \cdot 2^{-2} + f_{-3} \cdot 2^{-3} \dots \tag{5}$$

We wrote the Algorithm 1 on a laptop. The input is an integer or fractional variable. The same variable returned as a fixed-point output. $bf = i + f$, where i is the number of bits for the integer part, f for the fractional part, and bf is the total of bits used for that variable.

Algorithm 1: Binarization of a fixed-point variable.

$Aux = variable \times 2^f$

if $Aux < 0$ **then**
  $Aux = floor(2^{bf} + Aux)$
else
  $Aux = floor(Aux)$
end if

for $i = bf : -1 : 1$ **do**
  $Coeff\_binary(i) = Aux \,\%\, 2$
  $Aux = floor(Auxiliar/2)$
end for
$Aux = variable \times 2^f$

The *Coeff_binary* variable is a matrix with $1 \times bf$ size, bf is the number of zeros and ones stored in the matrix.

2.6. Design of the PID Algorithm

The continuous-time PID controller is given in (6). Its development and analysis are described in [22].

$$u(t) = K_p\left[e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d\frac{de(t)}{dt}\right] \tag{6}$$

where $u(t)$ is the control variable, and $e(t)$ is the error. The error is the difference between the reference $w(t)$ and the output $y(t)$. In Figure 6 a control system with the previous specifications is illustrated. The fundamental parameters of the PID controller are K_p, T_i, and T_d.

The Laplace transform converts the Equations (6) to (7).

$$U(s) = K_p\left[1 + \frac{1}{T_i\, s} + T_d\, s\right]E(s) \tag{7}$$

where s typically represents the Laplace transform operator. From (7), the transfer function $G(s)$ is in (8).

$$G(s) = \frac{U(s)}{E(s)} = K_p\left[1 + \frac{1}{T_i\, s} + T_d\, s\right] \tag{8}$$

The proportional, derivative, and integral components in (6) are discretized to get the PID controller. T_0 is the sample period. For this work, we use $T_0 = 10$ ms, and Equation (9) represents the derivative error.

$$\frac{de}{dt} \approx \frac{e(k) - e(k-1)}{T_0} = \frac{\Delta e(k)}{T_0} \tag{9}$$

where $e(k)$ is the error at the k-th sampling time, i.e., at $t = kT_0$. The most convenient way to do the integral is by summing. Hence we approximate the continuous-time function by sampling using the direct trapezoidal method (see Figure 7). The integration of the error is computed with (10), and the discrete-time PID controller is (11).

$$\int_0^t e(\tau)d\tau \approx T_0\sum_{i=1}^{k} e(i-1) \tag{10}$$

$$u(k) = K_p\left\{e(k) + \frac{T_0}{T_i}\sum_{i=1}^{k} e(i-1) + \frac{T_d}{T_0}[e(k) - e(k-1)]\right\} \tag{11}$$

The Equation (11) reduce the computational time processing on the FPGA, resulting in (12). The constants q_1, q_2, and q_3 in Equation (13) are computed before the PID time process.

$$u(k) = q_1\, e(k) + q_2\, e(k-1) + q_3\, e(k-2) + u(k-1) \tag{12}$$

$$q_1 = K_p\left(1 + \frac{T_0}{2\, T_i} + \frac{T_d}{T_0}\right)$$

$$q_2 = -K_p\left(1 - \frac{T_0}{2\, T_i} + \frac{2\, T_d}{T_i}\right) \tag{13}$$

$$q_3 = K_p\frac{T_d}{T_0}$$

Based on (12), we performed an Octave script to generate the VHDL code for a PID controller. Figure 8 shows the architecture resulting, where the input signals are reset (*RST*),

clock (*CLK*), setpoint (*Sp*), sampling time (*Ts*), actual position (*Xin*), and the constant values (q_1, q_2, q_3). The output signal is *yk* and the sampling time was 10 ms.

Figure 8 is the PID block diagram for hardware configuration; the Register_PP_n module loads the input signal every period to compute the output *yk*. A subtraction module was applied to compute the actual error $ek = sp - xin$; the q_1, q_2 and q_3 signals are the controller constants. An adder module is used to compute $yk_raw = q_1\ e_k + q_2\ e_{k2} + q_3\ e_{k3} + y_{k1}$, and four multipliers modules are implemented. The controller output signal *yk* is set to 18 bits.

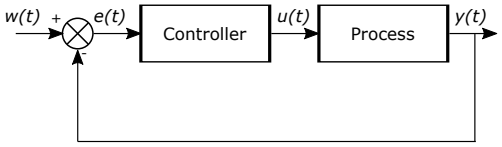

Figure 6. Block diagram of a control system [22].

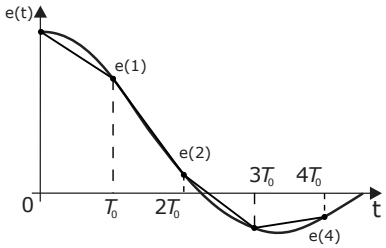

Figure 7. The forward trapezoidal method.

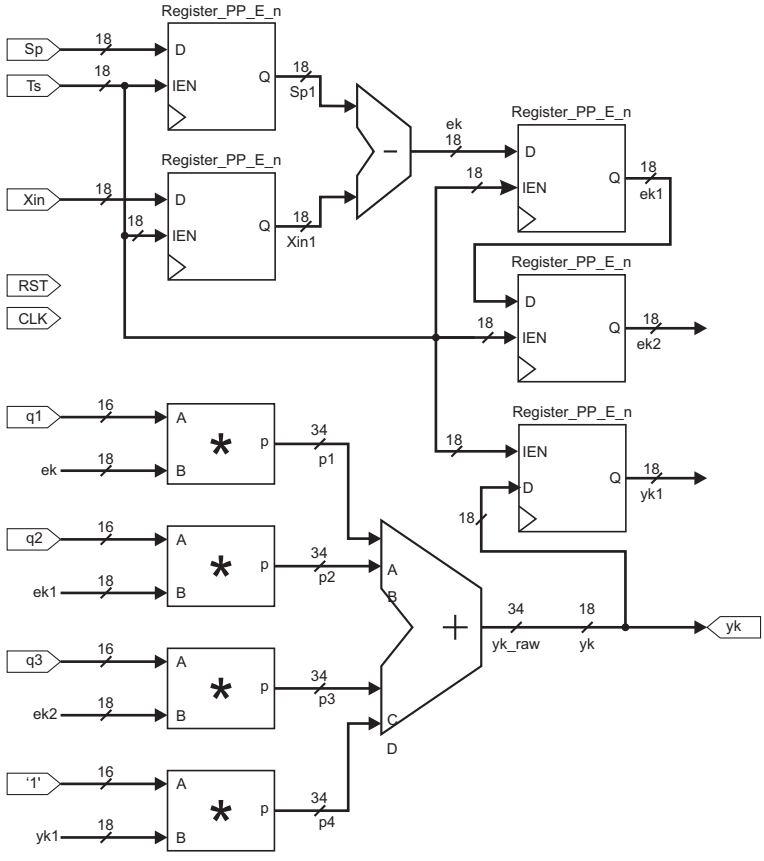

Figure 8. VHDL diagram block for PID controller. The symbol * in this figure represents a multiplier module.

2.7. PID Controller for the Airflow Valve

The arm position and speed are essential to perform the proper airflow control. The DC motors are linear systems that are practical to open or close the valve at the desired opening level to get the airflow required.

Based on Equations (12) and (13), Figure 9 shows the arm position and speed control proposal, where W_p is the desired position, e_p is the error position, u_p is the control signal. We used a saturation block where W_v is the saturation value and the maximum velocity desired. e_v is the speed error, and u_v is the control signal for the DC motor. The motor encoder sends position and speed feedback to the controller.

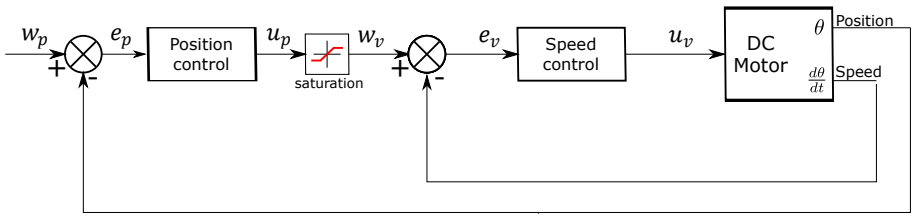

Figure 9. Diagram block for the motors PID controller.

2.8. Encoder Instrumentation

The quadrature encoder module has two inputs, Channel A and B. DIR output signal gives the direction of the motor. ENA is active when a state changes on channels A and B. We implemented six flip-flops type D to catch the signals, one XOR Gate with two inputs for DIR output and another XOR gate with four inputs for ENA output. Figure 10 displays the architecture and the corresponding truth table of the encoder module.

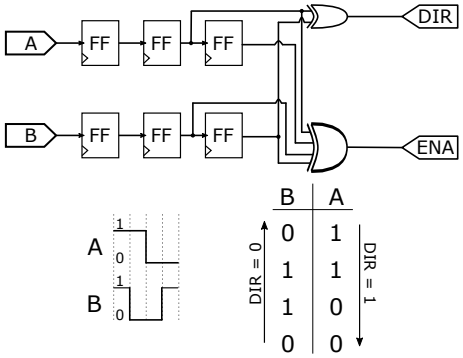

Figure 10. Encoder module functionality.

Figure 11 presents the quadrature encoder instrumentation, written in Octave software. The motor channel inputs are A and B. Ts is the period signal. We set a counter for the pulses from the encoders for the speed. When the T_s signal activates the rising edge module, the counter value is saved, and it is reset to start over. In Figure 11, outputs are drawn. The vel_nom and pos_nom are the raw values of the velocity and position. These have 16 bits.

The third output signal represents the normalized velocity. We obtained the value of the constant res_vel_norm in (14). Where $v_{MAX} = 108\ RPM$ and $N_v = 3432$ ppr, converted to 18 bits size. The vel_norm is the output of the normalized velocity given in the Equation (15).

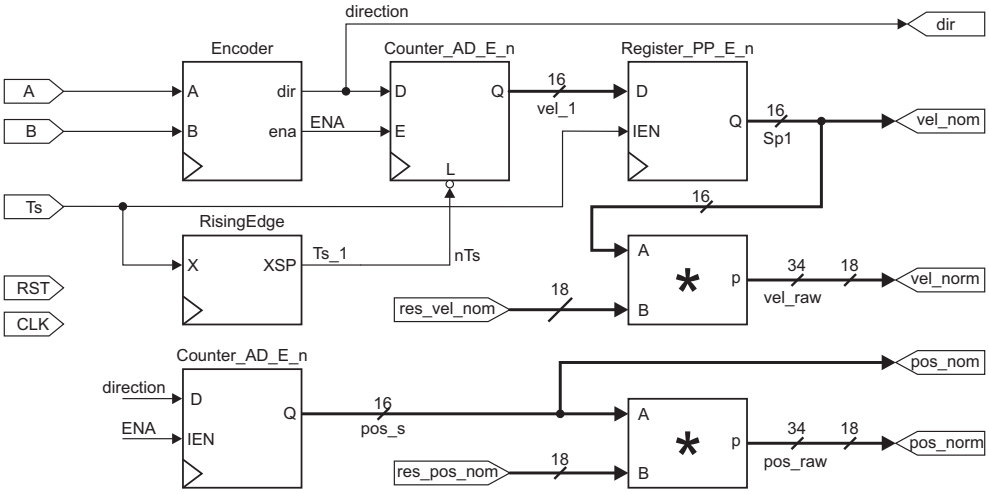

Figure 11. VHDL diagram block for the encoder configurations. The symbol * in this figure represents a multiplier module.

$$res\_vel\_norm = \frac{60}{V_{MAX} \times N_v \times T_s} \tag{14}$$

$$vel\_norm = res\_vel\_norm \times vel\_nom \tag{15}$$

For the normalization of the position in 18 bits, we considered the maximum of 10 rotations of the flow control valve for its constant given in Equation (16). The pos_norm is the output of the normalized position in Equation (17).

$$res\_norm = \frac{1}{P_{MAX} \times N_v} \times pos\_nom \tag{16}$$

$$pos\_norm = res\_pos\_nom \times res\_pos \tag{17}$$

2.9. Neural Network Design

Artificial neurons are suitable for non-linear systems, providing continuous outputs, gathering signals available on their inputs, and assembling them according to their operational and intuitive activation. Figure 12 illustrates each neuron of a network. The multiple input signals coming from the external environment (specific application) are represented by the set $\{x_1, x_2, x_3, \ldots, x_n\}$ [23].

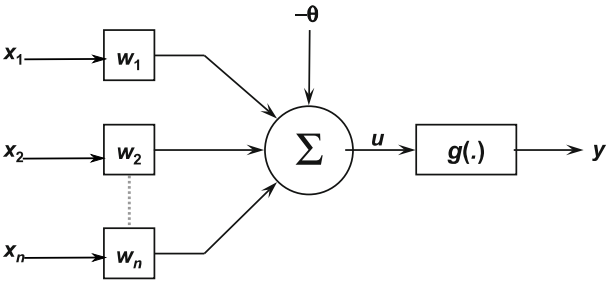

Figure 12. Artificial neural network.

The weighting carried out by the synaptic junctions of the network is implemented on the artificial neuron as a set of weights $\{w_1, w_2, \ldots, w_n\}$. Analogously, the relevance of each of the $\{x_i\}$ inputs is measured by multiplying them by their corresponding weight $\{w_i\}$, then weighting all the external information arriving at the neuron. Therefore, the neurons

output is denoted by y, representing the weighted sum of its inputs. Equations (18) and (19) synthesize the result produced by the artificial neuron [24].

$$u = \sum_{i=1}^{n} w_i \, x_i - \theta \tag{18}$$

$$y = g(u) \tag{19}$$

We trained the neurons by the method known as the backpropagation algorithm. By this method, we got the mean square error of the Equation (20).

$$E(t) = \frac{1}{2} \sum e^2(t) \tag{20}$$

2.10. Neuron Learning Algorithm

The specific purpose of this design is to properly use perceptron neurons to tune the variables k_p, T_i, and T_d of a digital PID controller in Equation (13). The objective is the mean square error, with a ± 0.5 tuning on each variable. These neurons are constantly learning. And the steps for the algorithm are the following [23].

1. The error signals e and Δe in (21) and (22) are computed from the desired position for each of the robot joints $q = (\theta_1, \ \theta_2, \ \theta_3)$, and the actual position being measured of the system $r = (x_1, \ x_2, \ x_3)$.

$$e(t) = y(t) - w(t) \tag{21}$$

$$\Delta e(t) = e(t-1) + e(t) \tag{22}$$

The error and derivative error are the inputs for each neuron. The description of this is in the Equation (23).

$$X(t) = \begin{bmatrix} e(t) & \Delta e(t) \end{bmatrix} \tag{23}$$

2. Initialize w with small random values. For example $w = 0.1$. We defined them in (24), and it represents the sum of the neuron weights in the first layer.
3. We set the learning rate to $\eta = 0.99$. And the following steps are repeated permanently, and the neuron is constantly learning.
4. The s variable is computed in (24).

$$s = w_1 \, e(t) + w_2 \, \Delta e(t) \tag{24}$$

5. The activation function, a sigmoid, of the intermediate neurons in (25) is computed, and Figure 13 is its representation.

$$h = \frac{0.5}{1 + e^{-s}} \tag{25}$$

6. The adaptive Equations (26) and (27) allowed the proportional coefficient K_p values to be adjusted. We used a similar development to find the adjustment equations for T_d and T_i. The value η is the learning coefficient of the neural network and was 0.9.

$$v(t+1) = v(t) + \eta e(t)^2 h \tag{26}$$

$$w_j(t+1) = w_j(t) + \eta e(t)^2 v h (1-h) x_j \tag{27}$$

7. The proportional gain is denoted by the Equations (28) and (29), where v is the weight of the last neuron and h is the activation function.

$$\Delta K_p = vh \tag{28}$$

$$K_p = K_p + \Delta K_p \tag{29}$$

8. Return to step 4.

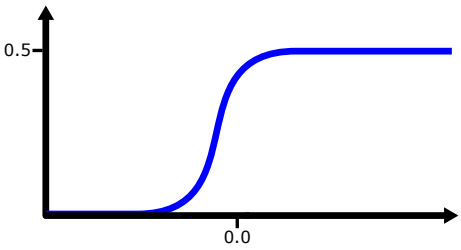

Figure 13. The sigmoid function.

Figure 14 shows the design of the neurons for k_p, and identical block diagrams are implemented for T_d and T_i. And they need to be initialized in this structural design. K_{pc}, T_{ic}, and T_{dc} are the constant inputs. The Rising_edge and Latch modules detect the first pulse of the period signal T_0, adding these variables to a summand that stores the initial value and updates the value of the kp, ti, and td variables. For Δe_k, a subtraction with a delay module is implemented to achieve $\Delta e(k) = e(k-1) - e(k)$, for s signal two multipliers with an adder module were implemented for (24). To achieve (25) a LUT was implemented, and it contains 2^{10} values between -1 to 1.

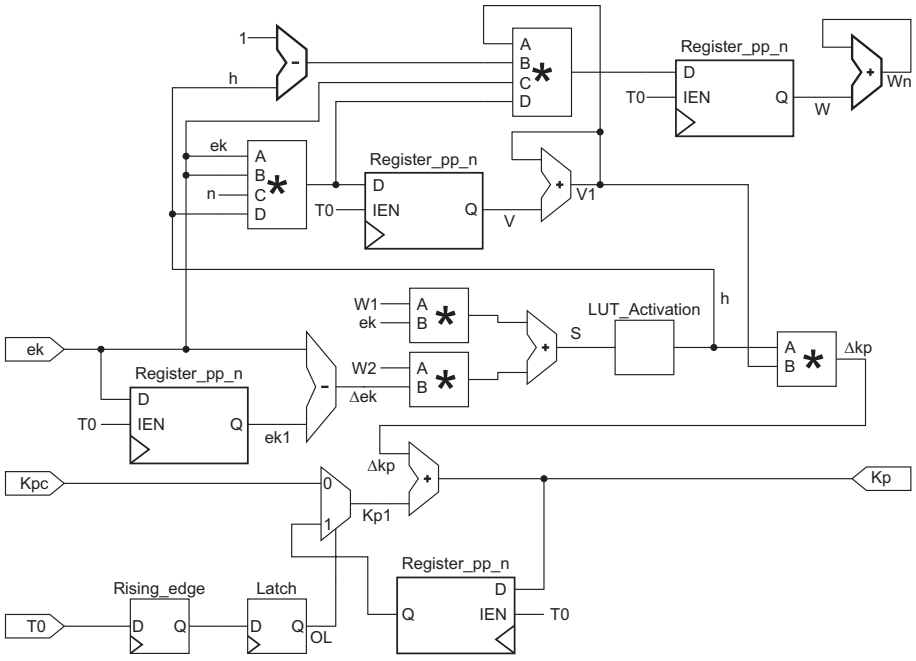

Figure 14. NNET module, neuro tuning variable for k_p. The symbol * in this figure represents a multiplier module.

For v in Equation (25), a multiplication module with four inputs was implemented, and an adder with a period delay module to compute $v(k+1)$. For the values of the weights w_1 and w_2 another four variables multiplier was needed, the $\eta\ e(t)^2\ h$ factor in (25) was repeated for (26). A single multiplier is needed to calculate (27). The value of k_p is constantly tuning, and k_{pc} is the initial value of k_p. To get k_p an adder is implemented, summing Δkp every period.

Finally, K_p, T_i, and T_d are computed to obtain q_0, q_1, and q_2 in Equation (13), Figure 15 is this conversion.

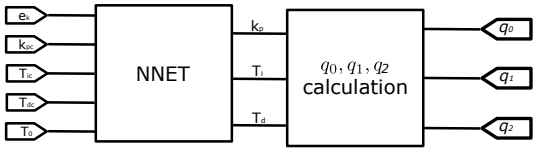

Figure 15. The calculation for q_0, q_1, and q_2.

Figure 16 presents the general diagram of a DoF, as it is observed inverse kinematics compute the desire angle values for a particular position of the end effector. Though only the second DoF is drawn, identical architecture was designed for the first and last DoF. According to e_k signed value, the direction of the valve was chosen.

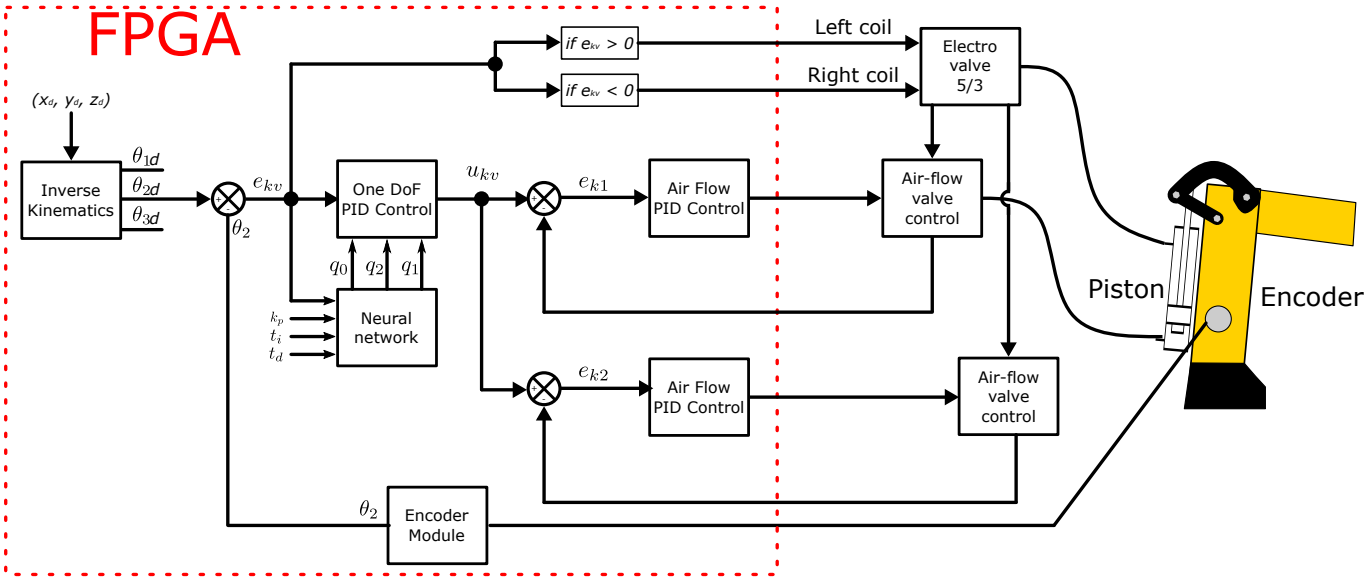

Figure 16. General diagram for a DoF.

The FPGA resources implemented in the project are observed in Table 2. We spent almost half of the resources, but the multipliers, that our project used 100%.

Table 2. Total resources of the hardware implementation and its quantities.

Total Resources	Quantity
Logic elements	10,134/22,320 (45%)
Registers	1671
Pins	50/154 (32%)
Multipliers 9 bits	132/132 (100%)

3. Results

This section, first shows the flow air control signals, position, and speed, including simulations and experimental results. Then, a graph with a PID and Neuro-PID experimental results for comparison. After that, we plotted the results of the 3 DoF, and we included error and derivative error signals graphs.

3.1. DC Motors Control for Airflow

As a first step, controlling the airflow valves is necessary. The VDC motors are the appropriate actuators due to their linear behavior and easy tuning. Figure 17 plots the simulations of the position and speed. When the system starts, the motor reaches the maximum allowed speed; it tends to be zero once it gets to the desired position. Figure 18 contains the experimental results, which are similar to those simulated. The sample

timeairflows. As the air flow valves have ten spins maximum, the top position is 63 rads. The full speed of the motors is 210 RPM, which means 22 rad/s.

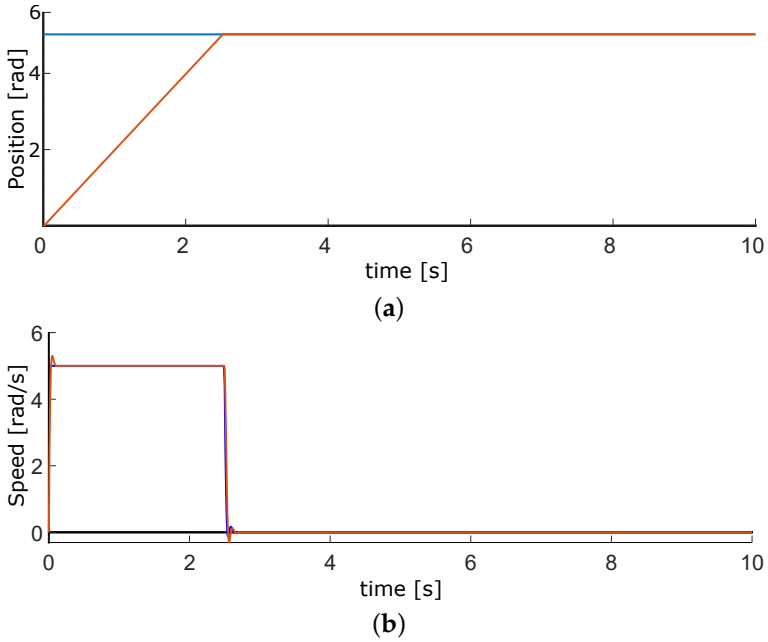

Figure 17. Simulated results for position and velocity control of DC motors where the blue lines are the setpoint and red ones the control signals. (**a**) Position plot. (**b**) Velocity plot.

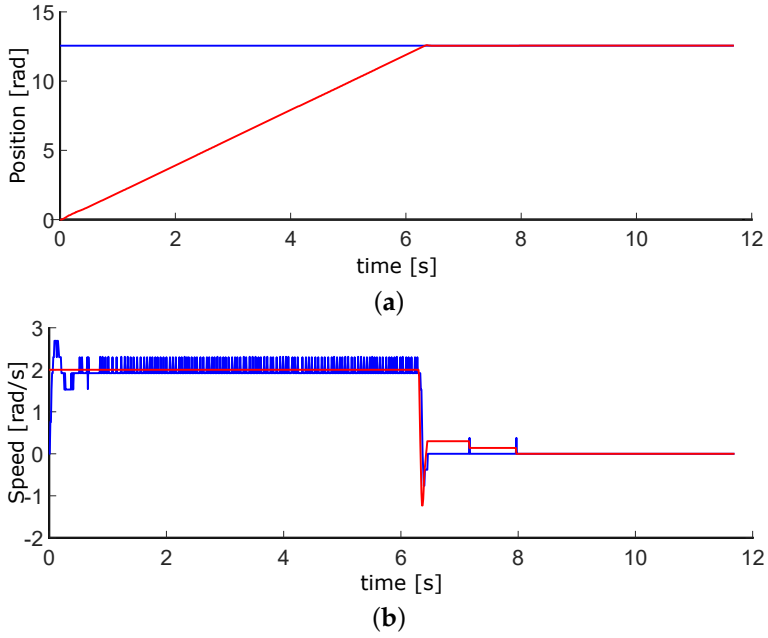

Figure 18. Experimental results for position and velocity control of DC motorswhere the blue lines are the setpoint and red ones the control signals. (**a**) Position plot. (**b**) Velocity plot.

We first considered a Texas Instruments DSP for the experimental results, which gave excellent results. However, PID controllers and neural networks needed a fast response due to the number of encoders. We decided to switch to the FPGA, a device that can perform all these processes in parallel. For these results, the position PID parameters are $q_0 = 0.1030$, $q_1 = -0.0949$, and $q_2 = 0.003$, for speed $q_0 = 0.0770$, $q_1 = -0.0140$, and $q_2 = 0.007$.

3.2. PID vs. Neuro-PID Controller

Once the robotic arm was configured, we performed tests with the designed controllers. For the PID and Neuro-PID gains, we are using the heuristic method, and the gain adjustment was performed by experimental tests. Figure 19 and the blue line is the classic PID controller. The red line is the Neuro-PID controller, which has a particular behavior with a faster response time, a little less overshoot, and reaches the steady-state before the typical PID. Table 3 displays the transient response specifications of these controllers.

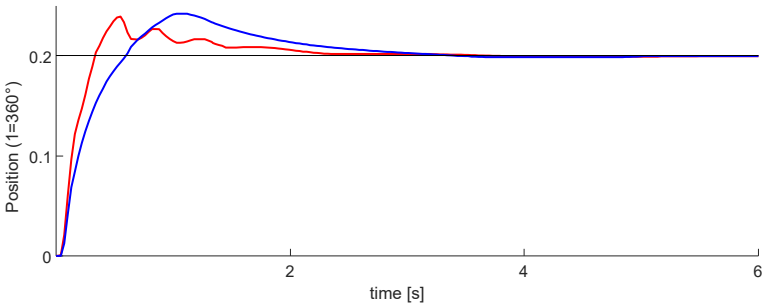

Figure 19. Experimental results for PID (blue) vs. Neuro-PID (red) controller for a DoF.

Table 3. Transient response specifications of PID and Neuro-PID controllers.

	PID	**Neuro-PID**
	$K_p = 2.5, T_d = 0$, **and** $T_i = 0.98$	
Rise time [s]	0.61	0.33
Peak time [s]	1.07	0.52
Overshoot [mm]	39.1	36.3
Settling time [s]	4.38	2.37
steady-state-error [mm]	2.22	2.2

We observed that the intelligent controller has some disturbances, possibly due to the adjustment of the variable K_p. For this test, we set $K_p = 2.5$, $T_i = 0.98$, and $T_d = 0$. In addition, for this and the following tests, the position and velocity variables were normalized. The maximum value of the signals is equal to one; this helps to use the computational resources better.

3.3. Positioning for the 3 DoF Links

The graph of the first DoF is in Figure 20. This is the one controlled by the pneumatic motor. Within these tests, this was the easiest to tune. It performs well, even when moving in conjunction with the other two DoFs.

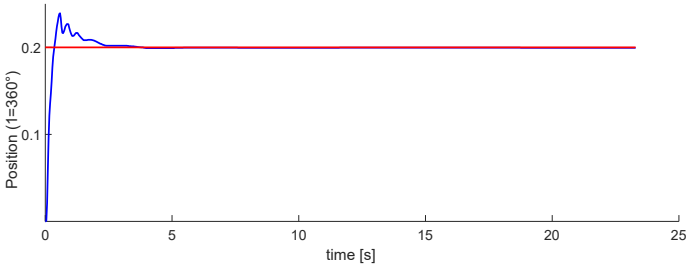

Figure 20. Experimental results for the first DoF. The Setpoint signal is colored in red, control signal in blue.

Figures 21 and 22 plot the second and third DoFs. These are controlled by the pistons located on the robot. Their tuning was more complex than the first one. In the second DoF, there are disturbances, but it is possible to tune to the marked reference. The third one was the most difficult to implement.

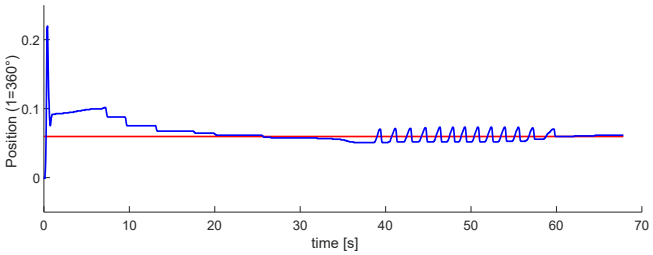

Figure 21. Experimental results for the second DoF. The Setpoint signal is colored in red, control signal in blue.

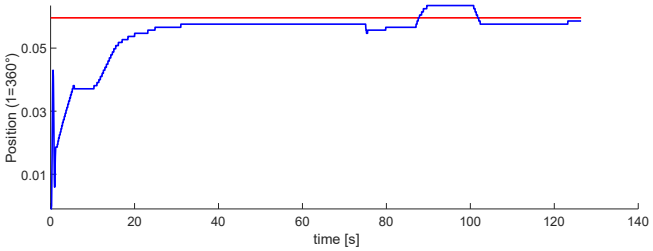

Figure 22. Experimental results for the third DoF. The Setpoint signal is colored in red, control signal in blue.

Figure 23 plots the error signal in blue color, the derivative error in red color. These signals are important for the PID and Neural network modules, and the error at steady-state is 1.2 mm.

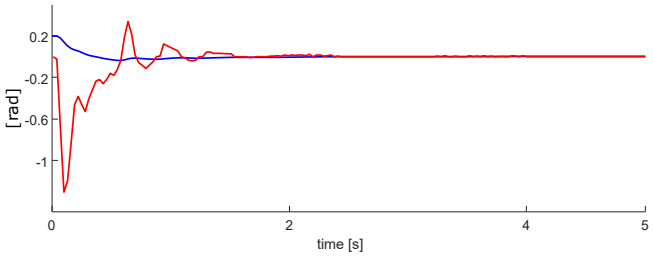

Figure 23. Experimental results for the error signal (blue) and derivative error (red) of a DoF.

4. Discussion

The proposed model has been developed to be applied to a 3 DoF pneumatic robot. The model uses three optical encoders with 1000 pulses by turn. Therefore, the resolution of the movement arm is 6.28 mrad. A limitation of the model is that the arm only has 120° to turn. That is due to the mechanical limitations.

The flow control is implemented with electrical motors coupled with flow control valves. Each motor for the air valve uses an encoder to know the state of the valve (open, close, and intermediate level). The DC motors use an electromagnetic encoder with 3432 pulses by turn. A limitation of the model is that we need two DC motors with encoders and their hardware to control them. In consequence, the model requires 6 DC motors with an encoder.

The implemented model must consider the hardware implementation for nine encoders, 3 PID control algorithms, 3 ANN algorithms, and the external logical signals used were 50 pins. We are using the following hardware of the FPGA D0-Nano board: 45% elements, 1671 registers, 32% pins, and 100% of the 9 bits multipliers.

In Section 3, according to Table 3 and Figure 19, we can observe that the Neuro-PID algorithm requires less rise time, overshoot, and settling time. The error is similar in both cases, but we consider that the ANN-PID algorithm has a better behavior than a simple PID. The limitation of the algorithm is the disturbances observed in Figure 19 on the Neuro-PID graph.

The disturbances in ANN-PID are due to the self-tuning process of the ANN algorithm to obtain the appropriate gains for the corresponding arm. One of the advantages of this process is that in the event of any disturbance that occurs in the arm positioning, the system will automatically respond to adjust itself. Moreover, the system has the advantage that if you change the weight being moved or add weight to one of the arms, the system responds by adjusting gains accordingly.

5. Conclusions

This work had three phases. The first one was the airflow control through the pneumatic system; although we did not control the air pressure directly, the positioning of the motors is adequate for its regulation. We performed the first tests on development boards based on microcontrollers. The motor speed control is considered for this design to avoid abrupt airflows. Figure 18a, shows the position of DC motor control, with an error of less than 0.01 rad. Moreover, Figure 18b represents the speed control, with an error of ± 0.5 rad/s. In this phase, the goal was achieved, With accurate position control and the speed controlled at slow or medium values.

The second phase was the control design for 1 DoF. This step was performed entirely in VHDL. Moreover, in this phase, the results in Figure 19 show a better performance when the neural network was active, improving the overshoot, response time, and reaching the steady state. We got a minimum error of 1.2 mm in the steady-state. The results in this phase were satisfactory.

The last phase is the implementation of the 3 DoFs. Results in Figures 20–22 show that the last two DoF were positioned to the desired position and had an error of 2 mm. We concluded that these results were acceptable. In future work, We consider implementing algorithms with fuzzy logic, neuro-fuzzy, and ANFIS.

Author Contributions: Conceptualization, M.-A.C.-R. and J.-M.R.-A.; methodology, M.-A.C.-R., J.R.-R. and M.-A.A.-F.; formal analysis E.G.-H. and M.-A.A.-F.; writing—original draft preparation, M.-A.C.-R. and J.-M.R.-A.; writing—review and editing, M.-A.C.-R., J.R.-R. and J.-M.R.-A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received funding from CONACYT.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

DoF	Degree of freedom
LUT	Look-Up Table
PID	Proportional, integral, and differential control
ANN	Artificial Neural Network
FPGA	Field Programming Gate Array
VHDL	Very High-Speed Integrated Circuit Hardware Description Language
DSP	Digital Signal Processing

DC	Direct Current
VDC	Volts of Direct Current
VAC	Volts of alternating Current
ANFIS	Adaptive neuro-fuzzy inference system

References

1. Ali, H.; Noor, S.; Bashi, S.; Marhaban, M. A review of pneumatic actuators (modeling and control). *Aust. J. Basic Appl. Sci.* **2009**, *3*, 440–454.
2. Singh, S.; Kishore, K.; Akbar, S.A. Neuro-evolutionary based controller design for linear and non-linear systems. In Proceedings of the 2021 21st International Conference on Control, Automation and Systems (ICCAS), Jeju, Korea, 12–15 October 2021; pp. 211–215. doi: 10.23919/ICCAS52745.2021.9649985. [CrossRef]
3. Miramontes, F.; Soto, J. Modelación de la Operación Neumática de un Brazo Manipulador. *Congr. Nac. Robót.* **2003**, *2*, 2.
4. Sorli, M.; Gastaldi, L.; Codina, E.; Heras, S. Dynamic analysis of pneumatic actuators. *Simul. Pract. Theory* **1999**, *7*, 589–602. [CrossRef]
5. Wang, X.; Peng, G. Modeling and control for pneumatic manipulator based on dynamic neural network. SMC'03 Conference Proceedings. In Proceedings of the 2003 IEEE International Conference On Systems, Man And Cybernetics. Conference Theme-System Security And Assurance (Cat. No. 03CH37483), Washington, DC, USA, 5–8 October 2003; Volume 3, pp. 2231–2236.
6. Rouzbeh, B.; Bone, G.; Ashby, G. High-accuracy position control of a rotary pneumatic actuator. *IEEE/ASME Trans. Mechatronics* **2018**, *23*, 2774–2781. [CrossRef]
7. Lin, C.; Sie, T.; Chu, W.; Yau, H.; Ding, C. Tracking control of pneumatic artificial muscle-activated robot arm based on sliding-mode control. *Actuators* **2021**, *10*, 66. [CrossRef]
8. Khanafer, M.; Shirmohammadi, S. Applied AI in instrumentation and measurement: The deep learning revolution. *IEEE Instrum. Meas. Mag.* **2020**, *23*, 10–17. [CrossRef]
9. Rodriguez-Abreo, O.; Rodriguez-Resendiz, J.; Fuentes-Silva, C.; Hernández-Alvarado, R.; Falcon, M. Self-Tuning Neural Network PID with Dynamic Response Control. *IEEE Access* **2021**, *9*, 65206–65215. [CrossRef]
10. Rodríguez-Abreo, O.; Garcia-Guendulain, J.; Hernández-Alvarado, R.; Flores-Rangel, A.; Fuentes-Silva, C. Genetic algorithm-based tuning of backstepping controller for a quadrotor-type unmanned aerial vehicle. *Electronics* **2020**, *9*, 1735. [CrossRef]
11. Rodríguez-Abreo, O.; Ornelas-Rodíguez, F.; Ramírez-Pedraza, A.; Hurtado-Ramos, J.; González-Barbosa, J. Backstepping control for a UAV-manipulator tuned by Cuckoo Search algorithm. *Robot. Auton. Syst.* **2022**, *147*, 103910. [CrossRef]
12. Hesselroth, T.; Sarkar, K.; Van Der Smagt, P.; Schulten, K. Neural network control of a pneumatic robot arm. *IEEE Trans. Syst. Man Cybern.* **1994**, *24*, 28–38. [CrossRef]
13. Alshareefi, H.; Lupu, C.; Olteanu, S.; Ismail, L. Design and Simulation of Adaptive Neuro-Fuzzy Inference System Inverse Controller for a Coupled Tank System. In Proceedings of the 2021 10th International Conference on Energy and Environment (CIEM), Bucharest, Romania, 14–15 October 2021; pp. 1–5.
14. Gillespie, M.; Best, C.; Townsend, E.; Wingate, D.; Killpack, M. Learning non-linear dynamic models of soft robots for model predictive control with neural networks. In Proceedings of the 2018 IEEE International Conference On Soft Robotics (RoboSoft), Livorno, Italy, 24–28 April 2018; pp. 39–45.
15. Mourad, A.; Zennir, Y. Fuzzy-PI Controller Tuned with HBBO for 2 DOF Robot Trajectory Control. *Eng. Proc.* **2022**, *14*, 10. [CrossRef]
16. Subramanian, S.; Mohan, R.; Shanmugam, S.; Bacanin, N.; Zivkovic, M.; Strumberger, I. Speed control and quantum vibration reduction of Brushless DC Motor using FPGA based Dynamic Power Containment Technique. *J. Ambient. Intell. Humaniz. Comput.* **2021**, *12*, 1–15. [CrossRef]
17. Muthuramalingam, A.; Himavathi, S.; Srinivasan, E. Neural network implementation using FPGA: Issues and application. *Int. J. Inf. Technol.* **2008**, *4*, 86–92.
18. Sánchez-Solar, S.; Rivas-Araiza, E.; Gorrostieta-Hurado, E.; Ramos-Arreguín, J. Simulation of a two DOF pneumatic manipulator robot using control based on back propagation neural network. In Proceedings of the 2017 International Conference On Electronics, Communications And Computers (CONIELECOMP), Cholula, Mexico, 22–24 February 2017; pp. 1–8.
19. Humaidi, A.; Ibraheem, I.; Azar, A.; Sadiq, M. A new adaptive synergetic control design for single link robot arm actuated by pneumatic muscles. *Entropy* **2020**, *22*, 723. [CrossRef] [PubMed]
20. Sünderhauf, N.; Brock, O.; Scheirer, W.; Hadsell, R.; Fox, D.; Leitner, J.; Upcroft, B.; Abbeel, P.; Burgard, W.; Milford, M.; et al. The limits and potentials of deep learning for robotics. *Int. J. Robot. Res.* **2018**, *37*, 405–420. [CrossRef]
21. Gazi, O. *A Tutorial Introduction to VHDL Programming*; Springer: Berlin/Heidelberg, Germany, 2019.
22. Bobál, V.; Böhm, J.; Fessl, J.; Macháček, J. *Self-Tuning PID Controllers*; Springer: Berlin/Heidelberg, Germany, 2005.
23. Silva, I.; Spatti, D.; Flauzino, R.; Liboni, L.; Alves, S. *Artificial Neural Networks: A Practical Course*; Springer International Publishing: Cham, Switzerland, 2017.
24. Gorrostieta, E.; Vargas, E.; Aguado, A. A neuro PD control applied for free gait on a six legged robot. *WSEAS Trans. Comput.* **2004**, *3*, 612–618.