2024 International Joint Conference on Neural Networks (IJCNN) | 979-8-3503-5931-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/IJCNN60899.2024.10651507

Neural fractional order PID controller embedded in an 8-bit microcontroller for neonatal incubator

Igor R. Sousa
Graduate Program on Teleinformatics Engineering
Federal University of Ceara
Fortaleza, Ceará
igor.sousa@alu.ufc.br

Guilherme A. Barreto
Graduate Program on Teleinformatics Engineering
Federal University of Ceara
Fortaleza, Ceará
gbarreto@ufc.br

***Abstract*—In this work, a fractional order PID controller for temperature of a neonatal incubator is implemented through a neural network in an embedded system with an 8-bit microcontroller. Using real data from a retrofitted commercial incubator, the plant transfer function is obtained, achieving RMSE = 0.0598. The fractional order PID controller parameters were obtained through particle swarm optimization metaheuristics using the Grunwald–Letnikov definition to calculate the control system fractional derivatives and integrals. The multilayer perceptron neural network is used in order to mitigate the loss of optimality that occurs in discretization methods of fractional order PID controllers, since they are infinite order filters. The neural fractional order PID controller in a simulated environment achieved RMSE = 0.0660 and was compared to the Merrikh-Bayat discretization technique, which obtained RMSE = 0.6578. The neural controller, with only 4 neurons in the hidden layer, was embedded in an 8-bit microcontroller to control the system, requiring only 6.511 ms to execute the MLP, occupying 1556 bytes of program memory and 241 bytes of RAM.**

***Index Terms*—fractional order controller, multilayer perceptron, nonlinear regression, embedded systems, control system.**

I. Introduction

Fractional order PID controllers (FOPID), also known by the acronym PI$^\lambda$D$^\mu$, are based on fractional calculus. The PI$^\lambda$D$^\mu$ controller transfer function is defined by

$$C(s) = \frac{U(s)}{E(s)} = k_p + \frac{k_i}{s^\lambda} + k_d s^\mu \,, \tag{1}$$

where λ and μ are the fractional orders of integration and derivation, respectively, while $U(s)$ and $E(s)$ are the controller output and the error signal of the control system, respectively. The conventional PID controller is a particular case of the PI$^\lambda$D$^\mu$ controller, where $\lambda = \mu = 1$. Therefore, in addition to parameterizing the gains k_p, k_i and k_d, it is necessary to parameterize the order of integration λ and derivation μ, providing additional degrees of freedom to the controller model, which can improve its performance in controlling complex dynamic systems [1].

This work has financial support from CNPq (No. 313251/2023-1) and the Centro de Referência em Inteligência Artificial - CEREIA, agreements 23067.055633/2021-19 (UFC) / 2020/09706-7 (FAPESP).

According to Jamil *et al.* [2], currently, PI$^\lambda$D$^\mu$ controllers have been used successfully in an increasing number of applications, as they can achieve better performance than classic PID controllers. In temperature control, Feliu-Batlle *et al.* [3] control a reheating oven with a PI$^\lambda$D$^\mu$ controller, presenting better results than conventional PID. Petráš and Vinagre [4] control a heating block using a PI$^\lambda$D$^\mu$ and PID controller, where the PI$^\lambda$D$^\mu$ also achieved better performance.

According to Shah and Agashe [1], PI$^\lambda$D$^\mu$ controller tuning techniques can be divided into three types: rule-based methods, such as Ziegler-Nichols for PI$^\lambda$D$^\mu$ [5], analytical methods, such as the application of piecewise orthogonal functions [6] and numerical optimization methods, such as the application of genetic algorithms [7], differential evolution [8], bee colony [9] and particle swarm optimization (PSO) [10].

One of the biggest problems that arises when implementing PI$^\lambda$D$^\mu$ controllers lies precisely in its mathematical theory, the fractional calculus, which is much more complex than the integer order differential and integral calculus [11]. However, over time, several researchers found different solutions to such problems, exploring geometric and physical interpretations [11]. It is important to mention that fractional-order controllers are infinite-dimensional linear filters, and all existing implementation methods are finite-dimensional approximations [12].

Therefore, the optimality of the control system achieved during the design phase is lost, since, in practice, any discrete system has memory limitations. In fact, an optimal controller designed in continuous time (i.e., the analog controller) will no longer be optimal after the discrete-time approximation [13].

A widely used technique in function regression is the multilayer perceptron (MLP) neural network, a universal function approximator [14]. Several authors use neural network in control systems, such as the development of a gain-scheduling PI$^\lambda$D$^\mu$, where a trained neural network selects the controller parameters for each scenario [15]. Munagala and Jatoth [16] used a MLP to learn the behavior of a PI$^\lambda$D$^\mu$ controller, with the control setpoint and the output of the plant as the MLP input, and the control signal as MLP output.

From the exposed, this work implements a PI$^\lambda$D$^\mu$ controller for temperature of a neonatal incubator by means of a MLP network with the aim of mitigating the loss of optimality that occurs in PI$^\lambda$D$^\mu$ controllers discretization methods.

The purpose of this work is to test the hypothesis that the MLP learns the behavior of a PI$^\lambda$D$^\mu$ controller with the true inputs of a PI$^\lambda$D$^\mu$ controller: the current and past error between the control system setpoint and the plant's output. Furthermore, the MLP output is the control signal variation, providing more smoothness to the system response, since this output condition has a natural integrator capable of obtaining zero error in the steady state [17].

Posteriorly, the neural PI$^\lambda$D$^\mu$ controller is embedded in an 8-bit microcontroller to perform system control. The achieved results are compared with an existing discretization method.

The remainder of the paper is divided as follows: Section II introduces the system identification algorithm. In Section III, the theory of fractional calculus applied to PID controllers is presented. Section IV presents the fundamentals of the PSO algorithm, while the controlled plant is presented in Section V. The methodology used is presented in Section VI. Finally, the results and conclusions are shown in Sections VII and VIII.

II. Identification by recursive least squares

In this work, the neonatal incubator is identified by a second order discrete-time ARX (autoregressive with exogenous input) model, defined as

$$y[k] = b_0u[k]+b_1u[k-1]+b_2u[k-2]-a_1y[k-1]-a_2y[k-2]\,, \quad (2)$$

where $u[k]$ and $y[k]$ denote, respectively, the model's input and output at time k. Coefficients $b_0, b_1, b_2, a_1, a_2 \in \mathbb{R}$ are the model parameters to be estimated. Equation (2) can be written in vector form as

$$y[k] = \boldsymbol{\theta}^T \boldsymbol{x}[k]\,, \quad (3)$$

where

$$\boldsymbol{x}[k] = [\ u[k]\ \ u[k-1]\ \ u[k-2]\ \ -y[k-1]\ \ -y[k-2]\ ]^T \quad (4)$$

is the regressor vector and $\boldsymbol{\theta} = \left[b_0\ b_1\ b_2\ a_1\ a_2\right]^T$ is the vector of coefficients of the model. The parameter vector $\boldsymbol{\theta}$ can be estimated by the recursive least squares (RLS) method [18], which is based on the minimization of the following cost function:

$$J_{RLS}[k]=\sum_{k=0}^{n}\phi^{n-k}\epsilon^2[k]=\sum_{k=0}^{n}\phi^{n-k}\Big(y[k]-\hat{\boldsymbol{\theta}}^T[k]\boldsymbol{x}[k]\Big)^2, \quad (5)$$

where $0 \le \phi \le 1$ is the forgetting factor and $\epsilon[k]$ is the error between the actual output value and the estimated one at the time k. This cost function is minimized at each iteration and the number of terms in the summation grows as a function of n. The recursive adjustment rule is written as

$$\hat{\boldsymbol{\theta}}[k+1] = \hat{\boldsymbol{\theta}}[k] + \boldsymbol{K}[k]\epsilon[k]\,, \quad (6)$$

where $\boldsymbol{K}[k] \in \mathbb{R}^5$ is the Kalman gain, defined by

$$\boldsymbol{K}[k] = \frac{\boldsymbol{P}[k]\boldsymbol{x}[k]}{\phi + \boldsymbol{x}[k]^T\boldsymbol{P}[k]\boldsymbol{x}[k]}\,, \quad (7)$$

and $\boldsymbol{P}[k] \in \mathbb{S}^5_{++}$ is the matrix defined by

$$\boldsymbol{P}[k+1] = \frac{1}{\phi}\Big\{\boldsymbol{P}[k] - \boldsymbol{K}[k]\boldsymbol{x}^T[k]\boldsymbol{P}[k]\Big\}\,. \quad (8)$$

III. Fractional Derivatives and Integrals

The Grunwald–Letnikov definition for the fractional integro-differential operation is

$${}_{t_0}\mathscr{D}_t^\alpha f(t) = \lim_{h\to 0}\frac{1}{h^\alpha}\sum_{j=0}^{[t-t_0/h]}(-1)^j\binom{\alpha}{j}f(t-jh)\,, \quad (9)$$

where

$$\binom{\alpha}{j} = \frac{\Gamma(\alpha+1)}{j!\Gamma(\alpha-j+1)}\,. \quad (10)$$

The value of α defines if the operation is a fractional derivative ($\alpha > 0$) or a fractional integral ($\alpha < 0$). The lower and upper limits of the operation are defined respectively by t_0 and t, while $\Gamma(\cdot)$ is the Gamma function.

According to Xue [19], if the calculation step h is small enough, the limit operator in (9) can be removed and can be approximately calculated as

$${}_{t_0}\mathscr{D}_t^\alpha f(t) \approx \frac{1}{h^\alpha}\sum_{j=0}^{[t-t_0/h]}(-1)^j\binom{\alpha}{j}f(t-jh)\,. \quad (11)$$

A. Discrete approximation for PI$^\lambda$D$^\mu$ controllers

Among the existing forms of implementation, the discrete long-memory PI$^\lambda$D$^\mu$ controller (LD-PI$^\lambda$D$^\mu$) stands out. Proposed by Merrikh-Bayat *et al.* [13], it discretizes the continuous variable s in (1) to z using the Tustin Transform and then performing a power series expansion, which is truncated due to memory limitation.

The approximation proposed by Merrikh-Bayat is based on the modified Tustin mapping, which applied to the fractional derivative s^μ, results in

$$s^\mu = \left(\frac{\omega_c}{\tan(\omega_c T/2)}\cdot\frac{1-z^{-1}}{1+z^{-1}}\right)^\mu = \alpha^\mu\left(\frac{1-z^{-1}}{1+z^{-1}}\right)^\mu\,, \quad (12)$$

where

$$\alpha \triangleq \omega_c/\tan(\omega_c T/2)\,, \quad (13)$$

ω_c is the cross-frequency gain for open-loop transfer function, T is the sample rate and $\tan(\cdot)$ is the tangent trigonometric function. The power series expansion of (12) is given by

$$\alpha^\mu\left(\frac{1-z^{-1}}{1+z^{-1}}\right)^\mu = \alpha^\mu\sum_{k=0}^{\infty}f_k(\mu)(z^{-1})^k\,, \quad (14)$$

where

$$f_k(\mu) = \frac{1}{k!}\cdot\frac{d^k}{d(z^{-1})^k}\left(\frac{1-z^{-1}}{1+z^{-1}}\right)\bigg|_{z^{-1}=0}\,. \quad (15)$$

Applying (12) and (14) into (1), with $s^{-\lambda} = (1/s)s^{1-\lambda}$, the approximation expression for the discrete PI$^\lambda$D$^\mu$ controller results in

$$C(z)=K_p+K_i\frac{1+z^{-1}}{1-z^{-1}}\sum_{k=0}^{M}f_k(1-\lambda)z^{-k}+K_d\sum_{k=0}^{M}f_k(\mu)z^{-k}, \quad (16)$$

where

$$K_p = k_p, \quad K_i = k_i\alpha^{-\lambda}, \quad K_d = k_d\alpha^\mu\,, \quad (17)$$

and M is the memory size considered in the approximation. If (16) is developed, it can be rewritten as

$$\frac{U(z)}{E(z)} = \frac{w_0 + w_1 z^{-1} + w_2 z^{-2} + \cdots + w_{M+1} z^{-(M+1)}}{1 - z^{-1}}, \tag{18}$$

where $w_0 \dots w_6$ are constants that depend on the controller gains (k_p, k_i, k_d), the derivation/integration exponents (λ, μ) and M.

IV. PSO Algorithm Fundamentals

The PSO is a population-based stochatic optimization algorithm and the search for optimal solution occurs in multiple directions due to the various particles, instead of a single direction. The algorithm is inspired by the social behavior and self-organization of groups of migratory birds and schools of fish [20]. The PSO algorithm keeps a memory to retain the best solutions found by the particles at each iteration. Furthermore, the algorithm implements cooperation between the particles in the swarm, sharing information [21].

For optimization in a D-dimensional search space, the position of the j-th particle of the swarm is defined as $\boldsymbol{p}_j = [p_{j1}, p_{j1}, \cdots, p_{jD}]$. In iteration k, the best position found by the particle $\boldsymbol{p}_j$ is defined as $\boldsymbol{p}_j^{best}[k]$, while the best position found by the swarm is defined as $\boldsymbol{g}^{best}[k]$. These positions are used to update the velocities $\boldsymbol{v}_j$ and the positions of each particle as

$$\begin{aligned}
\boldsymbol{\zeta}_1[k] &= c_1 \psi_1[k] \left(\boldsymbol{p}_j^{best}[k] - \boldsymbol{p}_j[k] \right) \\
\boldsymbol{\zeta}_2[k] &= c_2 \psi_2[k] \left(\boldsymbol{g}^{best}[k] - \boldsymbol{p}_j[k] \right) \\
\boldsymbol{v}_j[k+1] &= \varrho \left\{ w[k] \boldsymbol{v}_j[k] + \boldsymbol{\zeta}_1[k] + \boldsymbol{\zeta}_2[k] \right\} && (19) \\
\boldsymbol{p}_j[k+1] &= \boldsymbol{p}_j[k] + \boldsymbol{v}_j[k+1], && (20)
\end{aligned}$$

where $c_1, c_2 \in \mathbb{R}_{>0}$ are the cognitive and social learning coefficients, respectively. The random variables ψ_1 and ψ_2 are uniformly distributed in the interval $(0, 1)$. The coefficient ϱ is a constriction factor. Finally, $w[k]$ is the decreasing inertial weighting factor at time k and aims to avoid problems related to particle speed. In this work, this factor is defined as

$$w[k] = w_{final} + (w_{initial} - w_{final}) \left(\frac{N_{it} - k}{N_{it} - 1} \right), \tag{21}$$

where $w_{initial}$ and w_{final} are, respectively, the initial and final value of w, while N_{it} is the number of iterations of the PSO algorithm.

Finally, the performance of each particle $\boldsymbol{p}_j[k]$ and the positions $\boldsymbol{p}_j^{best}[k]$ and $\boldsymbol{g}^{best}[k]$ are evaluated using an objective function. The optimal solution obtained depends on the established objective function, which is fundamental to the control system design.

An approach for objective function that stands out is presented in Aghababa [22], which uses transient response (TR) criteria of the control system, such as the rise time T_r, settling time T_s, maximum overshoot M_o and steady state error E_{ss},

$$J_{TR} = \frac{1}{1 + e^{-r}} (T_r + T_s) + \frac{e^{-r}}{1 + e^{-r}} (M_o + E_{ss}), \tag{22}$$

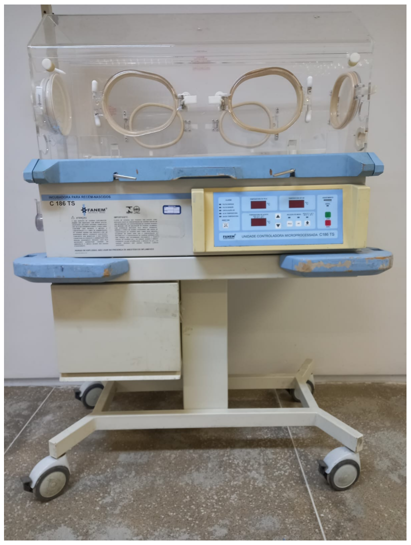
where r is a weighting factor.

V. Controlled Plant Description

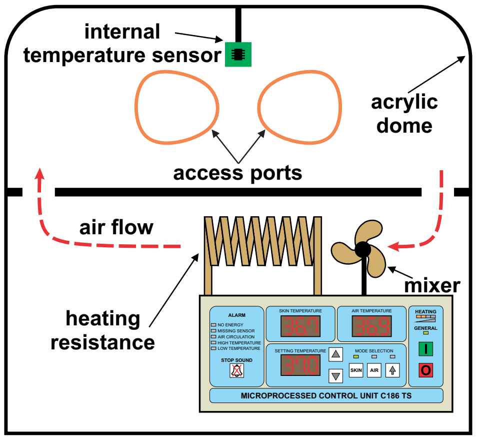
The plant used in this work is a C186 TS neonatal incubator, FANEM®, shown in Fig. 1. The incubator uses an electrical heating resistance, which is triggered by a TRIAC via firing angle. The controlled variable is the internal temperature of the incubator, while the control variable is a numerical value related to the microcontroller's timer register that results in a firing angle for triggering the TRIAC.

A retrofit was carried out on the electronic board of the neonatal incubator processing unit, where the connections of the original microcontroller with the TRIAC's triggering circuit and the safety relays were removed. Then, an external microcontroller was coupled and new connections could be created, such as the zero crossing detection circuit and the TRIAC's triggering circuit.

The microcontroller used in the retrofit is the ATMega328p, Microchip®, which has 2 KB of SRAM and 32 KB of program memory. Thus, it was possible to perform identification and control tests using this external 8-bits microcontroller.



(a) Test bench.



(b) Thermal plant schematic drawing.

Fig. 1: C186 TS neonatal incubator, FANEM®.

VI. Methodology

The process developed for evaluation and implementation of the neural PI$^\lambda$D$^\mu$ controller is presented in Fig. 2. Each of the steps in the process are presented below.

A. System identification

First, plant identification is performed using a second-order ARX model and the RLS estimation algorithm presented in Section II. The input signal used is a PRBS (pseudorandom binary sequence). Using the microcontroller applied in the plant retrofit, the PRBS signal and the internal temperature of the incubator are acquired at a rate of $1000\,s$ due to the large time constants present in the plant.

B. PSO for PI$^\lambda$D$^\mu$ controller

The PSO algorithm is used in this work to find the optimal values of the PI$^\lambda$D$^\mu$ controller parameters. In this work, the j-th particle position is five-dimensional and is defined as $\boldsymbol{p}_j = [\,k_{p_j}\;\; k_{i_j}\;\; \lambda_j\;\; k_{d_j}\;\; \mu_j\,]$, where box constraints for the parameters have been defined, such as $k_{p_j}, k_{i_j}, k_{d_j}, \lambda_j, \mu_j \in [0\,,\,1]$. Ten particles are used in 15 iterations of the algorithm, with $w_{initial} = 0.2$, $w_{final} = 0.001$, $c_1 = 0.8$ and $c_2 = 1$. A total of 10 realizations of the algorithm are executed. In each realization, the particle positions are initialized randomly. The objective function used is the one present in (22) with $r = 0.8$. The control system has a controller output saturator between 0 and 262, necessary to ensure that the system operates within the plant's physically possible ranges.

The Grunwald-Letnikov definition, see (11), was used to compute the fractional derivatives and integrals of the control system simulation for the iterations of each PSO particle. The calculation step used is 1 s, small enough given the large time constants of the plant. For a given time instant k, all previous samples are used to calculate the derivatives and fractional integrals, given zero initial conditions. The simulated time of the control system is 20,000 s, i.e., 20,000 samples per run for each iteration of each particle and each realization. Given the presented PSO settings, the algorithm's runtime to find the best particles is 97 hours for a computer with a 4.6 GHz i7 processor and 16 GB RAM.

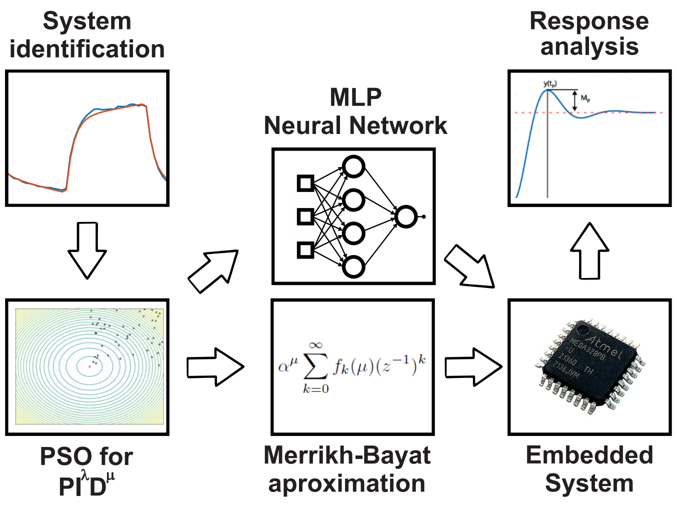

System identification
MLP Neural Network
Response analysis
$y(t_p)$
M_p
$\alpha^\mu \sum_{k=0}^{\infty} f_k(\mu)(z^{-1})^k$
Atmel
PSO for PI$^\lambda$D$^\mu$
Merrikh-Bayat aproximation
Embedded System

Fig. 2: Steps for neural PI$^\lambda$D$^\mu$ controller development.

C. Merrikh-Bayat aproximation for PI$^\lambda$D$^\mu$ controller

The Merrikh-Bayat approximation is used in this work for comparison purposes with the proposed idea. The sampling rate used is 1 s. For the Merrikh-Bayat approximation of the designed PI$^\lambda$D$^\mu$ controller, the expression (16) is developed with $M = 5$, a value used by [13] and [23]. Thus, Eq. (18) is obtained with

$$
\begin{aligned}
w_0 &= K_p + K_i + K_d \\
w_1 &= -\left\{K_p + K_d(2\mu+1) + K_i[2(1-\lambda)-1]\right\} \\
w_2 &= 2\left\{K_d(\mu^2+\mu) + K_i[(1-\lambda)^2-(1-\lambda)]\right\} \\
w_3 &= -2\Bigg\{K_d\left(\frac{2}{3}\mu^3+\mu^2+\frac{1}{3}\mu\right)+ \\
&\quad + K_i\left[\frac{2}{3}(1-\lambda)^3-(1-\lambda)^2+\frac{1}{3}(1-\lambda)\right]\Bigg\} \\
w_4 &= 2\Bigg\{K_d\left(\frac{1}{3}\mu^4+\frac{2}{3}\mu^3+\frac{2}{3}\mu^2+\frac{1}{3}\mu\right)+ \\
&\quad + K_i\left[\frac{1}{3}(1-\lambda)^4-\frac{2}{3}(1-\lambda)^3+\frac{2}{3}(1-\lambda)^2-\frac{1}{3}(1-\lambda)\right]\Bigg\} \\
w_5 &= -2\Bigg\{K_d\left(\frac{2}{15}\mu^5+\frac{1}{3}\mu^4+\frac{2}{3}\mu^3+\frac{2}{3}\mu^2+\frac{1}{5}\mu\right)+ \\
&\quad + K_i\Bigg[\frac{2}{15}(1-\lambda)^5-\frac{1}{3}(1-\lambda)^4+\frac{2}{3}(1-\lambda)^3+ \\
&\quad -\frac{2}{3}(1-\lambda)^2+\frac{1}{5}(1-\lambda)\Bigg]\Bigg\} \\
w_6 &= 2\Bigg\{K_d\left(\frac{2}{15}\mu^5+\frac{2}{3}\mu^3+\frac{1}{5}\mu\right)+ \\
&\quad - K_i\left[\frac{2}{15}(1-\lambda)^5+\frac{2}{3}(1-\lambda)^3+\frac{1}{5}(1-\lambda)\right]\Bigg\}.
\end{aligned}
\tag{23}
$$

Therefore, Eq. (18) for $M = 5$ can be rewritten as

$$
u[k] = u[k-1] + \boldsymbol{w}^T\boldsymbol{E}[k]\,, \tag{24}
$$

where $\boldsymbol{w} = [w_0\;\ldots\;w_6]^T$ and $\boldsymbol{E}[k] = [\,E[k]\;\ldots\;E[k-6]\,]^T$.

Remark 1. The parameters k_p, k_i and k_d are the controller gains in the continuous time domain, i.e., the analog controller presented in (1). On the other hand, K_p, K_i and K_d are the versions of these gains in the discrete approximation proposed by Merrikh-Bayat *et al.* [13].

D. MLP neural network application

After obtaining the PI$^\lambda$D$^\mu$ controller parameters through PSO, the neural network is trained. The architecture used to train the neural network is shown in Fig. 3, where the

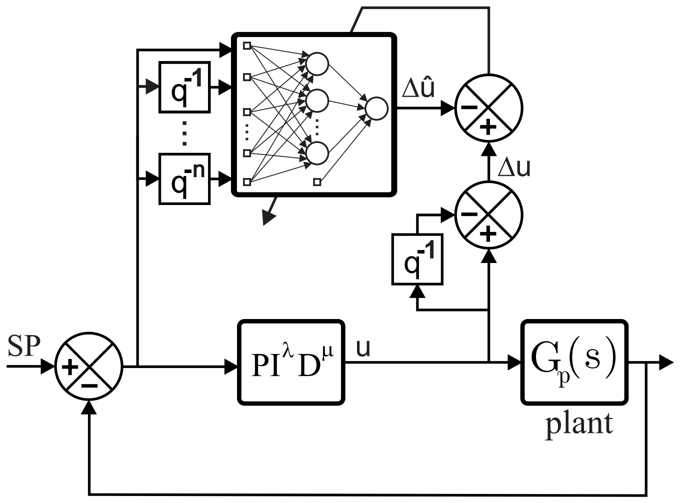


Fig. 3: Architecture used for training the neural network.

temperature setpoint (SP) used is 36.1 °C. To obtain a fair comparison, the neural network has the current error and six past values from the control system with the PI$^\lambda$D$^\mu$ controller as inputs, i.e., the same number of inputs as the Merrikh-Bayat approximation (see (24)).

The neural network's output is an estimate of control signal variation $\Delta\hat{u}[k]$, not the estimate of control signal $\hat{u}[k]$ itself. This choice is due to the fact that digital controllers with control signal variation as their output have a natural integrator, capable of obtaining zero error in the steady state [17]. The signals used for training the neural network are the signals generated by control system simulation without the saturation. Since the PI$^\lambda$D$^\mu$ controller is a filter, it is important that the data used to train the MLP are not interfered with by a saturator. The sampling rate used 1 s, the same used in the Merrikh-Bayat approximation.

The MLP has one hidden layer, with neurons using hyperbolic tangent as activation functions. The output layer consists of just one linear neuron. The data is split into 80% for training and 20% for the test. The network was trained using gradient descent algorithm. Heuristics and successive tests reveal that the learning rate with linear decay, 0.2 initial value and 0.001 final value, along with 400 epochs, is the best option to be used. Input and output data are rescaled to the range $[-1, +1]$. The parameters of the normalizations are stored for later use in the embedded system implementation step.

To find the optimal number of hidden layer neurons, 20 rounds of training were performed for different numbers of neurons, where five indexes are analyzed: the correlation coefficient R, its standard deviation σ_R, the coefficient of determination R^2, the Mean Bias Error - MBE and the Root Mean Squared Error - RMSE, whose expressions are given by

$$R = \frac{n(\sum xy)-(\sum x)(\sum y)}{\sqrt{\left[n(\sum x^2)-(\sum x)^2\right]\left[n(\sum y^2)-(\sum y)^2\right]}}, \tag{25}$$

$$\sigma_R = \sqrt{\tfrac{1}{n}\textstyle\sum_{i=1}^{n}(R_i-\bar{R})^2}, \tag{26}$$

$$\text{MBE} = \tfrac{1}{n}\textstyle\sum_{i=1}^{n}(y_i-x_i), \tag{27}$$

$$\text{RMSE} = \sqrt{\tfrac{1}{n}\textstyle\sum_{i=1}^{n}(y_i-x_i)^2}, \tag{28}$$

where x_i and y_i represent $\Delta\hat{u}$ and Δu, respectively.

The control system implementation with the neural PI$^\lambda$D$^\mu$ controller is shown in Fig. 4. Since the neural network's output is the estimate of control signal variation $\Delta\hat{u}$, the estimate of the control signal $\hat{u}$ that is sent to the plant is obtained as

$$\hat{u}[k] = \hat{u}[k-1] + \Delta\hat{u}[k]\,. \tag{29}$$

This control system implementation has the same saturator present during the parameter search step with the PSO algorithm (see Section VI-B), necessary to ensure that the control signal remains within physically possible values of real implementation. The same is applied to the control system with Merrikh-Bayat approximate controller.

E. Evaluation of the system in an embedded platform

The best neural network configuration is embedded in an 8-bit microcontroller to evaluate the MLP performance running in an hardware system. The microcontroller used is the same one used in the retrofit and data acquisition, described in the Section V, the ATMega328p. In this work, the ATMega328p clock frequency is 16 MHz.

The embedded control system uses the following sequence of operations: the microcontroller requests the current temperature information from the sensor, calculates the current error, performs MLP calculations generating the PI$^\lambda$D$^\mu$ control signal variation $\Delta\hat{u}[k]$, and finally updates the control signal. All the calculation variables are floating point type.

Finally, the runtime, the program memory footprint and RAM usage are measured and analyzed in each of the stages involved in the process: sensor response, inputs update & data normalization, hidden layer's summations, hidden layer's activation functions, output layer's summation, data denormalization, and controller output update. The runtime of each stage is measured with the aid of an oscilloscope.

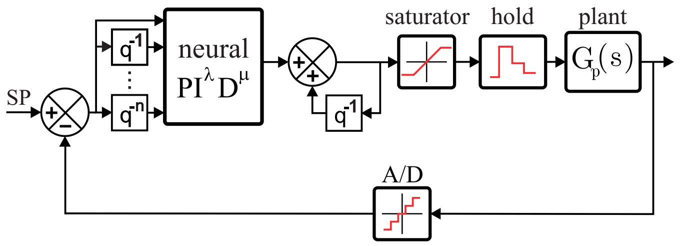


Fig. 4: Control system with neural PI$^\lambda$D$^\mu$ controller.

VII. Results

The second-order discrete-time model resulting from the RLS estimation presented in Section II is

$$G_p(z) = \frac{Y(z)}{U(z)} = \frac{0.0044 + 0.0455z^{-1} - 0.0476z^{-2}}{1 - 1.5531z^{-1} + 0.5621z^{-2}}\,, \quad (30)$$

while the continuous model, obtained from (30) by applying the zero-order hold method, is given by

$$G_p(s) = \frac{Y(s)}{U(s)} = \frac{0.0044s^2 + 6.989{\cdot}10^{-5}s + 3.03{\cdot}10^{-9}}{s^2 + 0.0005761 + 1.185 \cdot 10^{-8}}. \quad (31)$$

After preliminary tests, it was decided to use the RLS algorithm's forgetting factor as $\phi = 0.95$. The comparison between acquisition data and those estimated by the model are shown in Fig. 5. As expected, the model and the incubator have similar dynamics. The RMSE between acquisition data and that estimated by the model is 0.0598.

The result of PSO algorithm, used to find the best parameters of the PI$^\lambda$D$^\mu$ controller for 10 realizations, is shown in Fig. 6. The control system responses for the 10 $\boldsymbol{g}^{best}$ particles of the 10 realizations are shown in Fig. 6a. The respective convergence curves are shown in Fig. 6b. The curve producing the lowest objective function is highlighted. The PI$^\lambda$D$^\mu$ controller parameters resulting from the optimization process are presented in Table I.

Table II presents the performance indices for the best results for each MLP configuration. All tested configurations have low standard deviation and high correlation coefficient values, bringing reliability to training. Despite not having the best MBE and RMSE index, the configuration with 4 hidden neurons has the highest average value of R (0.981) and highest R^2 (0.9999). Therefore, this configuration is chosen as the best. The coefficient of determination for the configuration with 4 hidden neurons is shown in Fig. 7, while the MLP output, i.e., the estimated control signal variation $\Delta\hat{u}$ is shown in Fig. 8.

The achieved results corroborate the hypothesis that a MLP network can lear the PI$^\lambda$D$^\mu$ controller behavior, an infinite order filter, with high accuracy and only few inputs.

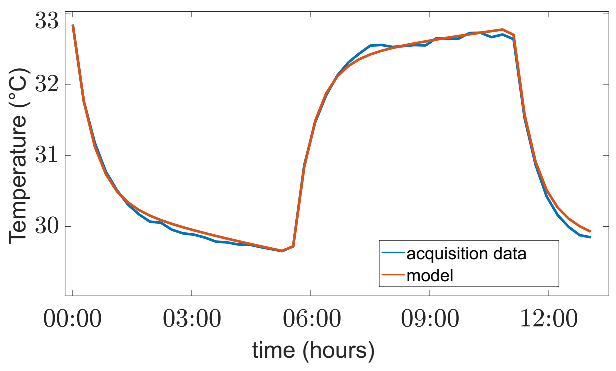
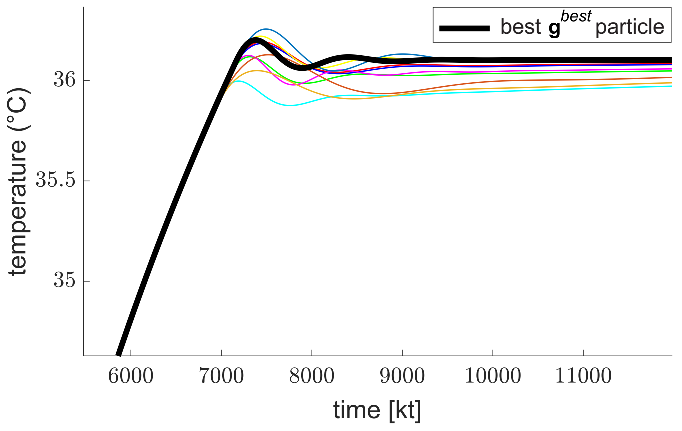
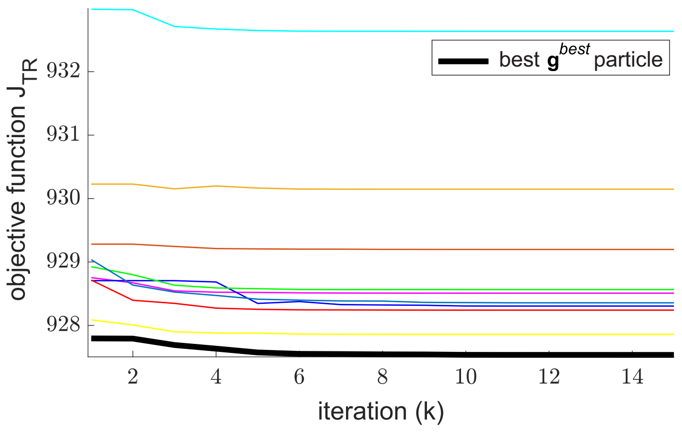


Fig. 5: Comparison between acquisition data and ARX model.



(a) Control system temporal responses.



(b) Objective function convergence.

Fig. 6: Performance of the 10 $\boldsymbol{g}^{best}$ particles.

TABLE I: Resulting optimal parameters.

k_p	k_i	k_d	λ	μ
0.409792	0.754561	0.637085	0.953790	0.689195

TABLE II: Performance indices for each configuration.

hidden neurons	best			average
	R^2	MBE	RMSE	$R \pm \sigma_R$
3	0.9998	9.38·10^{-4}	4.03·10^{-2}	0.9705 ± 0.0215
4	**0.9999**	**-2.68·10^{-3}**	**3.53·10^{-2}**	**0.9810 ± 0.0436**
5	0.9998	1.10·10^{-3}	4.78·10^{-2}	0.9749 ± 0.0373
6	0.9999	-2.19·10^{-3}	3.46·10^{-2}	0.9696 ± 0.0391
7	0.9999	-1.92·10^{-3}	3.51·10^{-2}	0.9691 ± 0.0382

To evaluate the neural PI$^\lambda$D$^\mu$ controller performance, the proposed controller is compared to the Merrikh-Bayat approximation PI$^\lambda$D$^\mu$ controller. The cross-frequency gain, see (13), for the controller parameters obtained with the PSO and presented in Table I is $\omega_c = 0.006727$. Applying the PI$^\lambda$D$^\mu$ controller parameters values in (17) and (23),

$$w_0 = 1.8266 \qquad w_1 = -2.4994 \qquad w_2 = 2.3575$$
$$w_3 = -1.9065 \qquad w_4 = 1.7144 \qquad w_5 = -1.5853$$
$$w_6 = 0.76690\,.$$

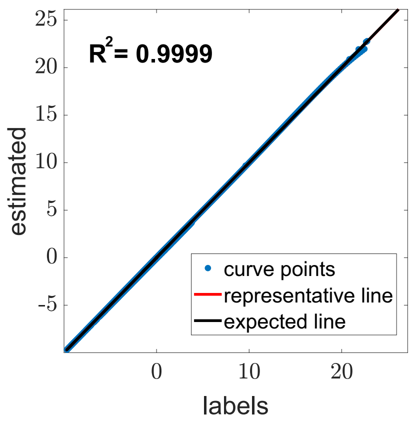


Fig. 7: Comparison between $\Delta\hat{u}$ and Δu.

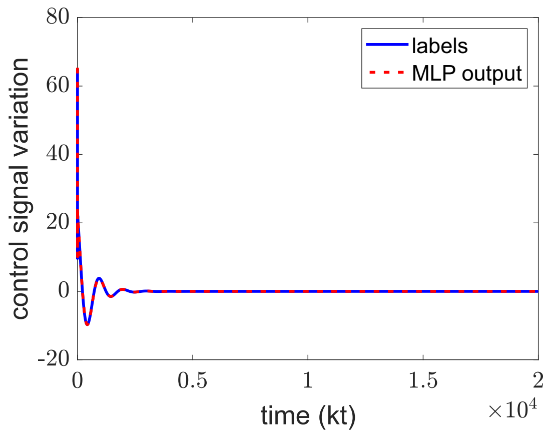


Fig. 8: Time evolution of the MLP output.

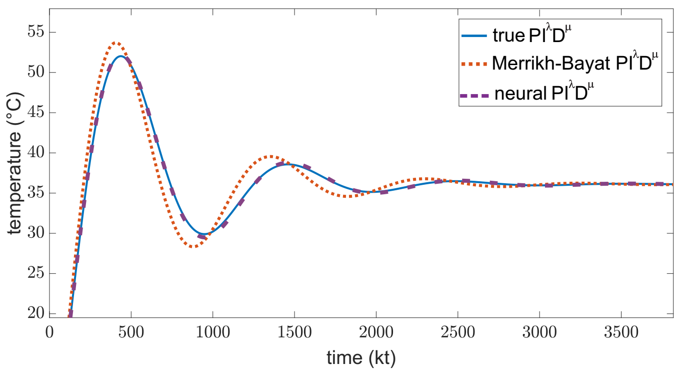


Fig. 9: Controllers performance without saturation.

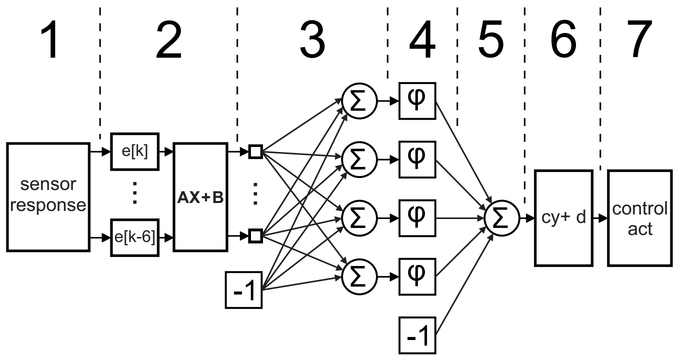


Fig. 10: Stages of the embedded system in ATMega328p.

TABLE III: Embedded system performance metrics.

	Stage	Runtime (**μs**)	Program memory (**bytes**)	RAM (**bytes**)
1	sensor response	834	998 (3.25%)	32 (1.56%)
2	inputs update & data normalization	3496	28 (0.09%)	8 (0.39%)
3	hidden layer's summations	874.5	858 (2.79%)	41 (2.0%)
4	hidden layer's activation functions	2000	122 (0.40%)	16 (0.78%)
5	output layer's summation	123.5	258 (0.84%)	144 (7.03%)
6	data denormalization	16.94	290 (0.94%)	32 (1.56%)
7	control output update	39.6	2932 (9.54%)	62 (3.03%)
	total	7384.54	5486 (17.86%)	335 (16.36%)

The comparison between the control systems simulated responses with the true[1] $PI^\lambda D^\mu$, neural $PI^\lambda D^\mu$ and Merrikh-Bayat approximate $PI^\lambda D^\mu$ controllers without saturator is shown in Fig. 9. The system output resulting from the neural $PI^\lambda D^\mu$ controller is the closest to the system output with true $PI^\lambda D^\mu$ controller, with smaller overshoot. In fact, the neural $PI^\lambda D^\mu$ controller RMSE is 0.0660 while the Merrikh-Bayat approximation RMSE is 0.6578.

It should be noted that the control systems, whose responses are shown in Fig. 9, do not have a saturator, since it is essential to compare the $PI^\lambda D^\mu$ controller output approximations without interference in the control signal.

To evaluate the practical performance of the neural $PI^\lambda D^\mu$ controller, the MLP network was embedded in the ATMega328p, the same microcontroller used in the retrofit and data acquisition, described in Section V.

Spatial and temporal measurements performed for each embedded stage are shown in Fig. 10 and Table III. The total runtime on the embedded system is $7384.54\,\mu s$. The stages with the longest runtime are the input update & normalization stage, and hidden layer's activation functions stage.

[1]obtained by applying the Grunwald-Letnikov definition to compute the fractional derivatives and integrals at each instant (see Section III and VI-B).

The actual temporal responses of embedded systems with neural and Merrikh-Bayat $PI^\lambda D^\mu$ controllers are shown in Fig. 11. The controllers update time used is 1000 s, the same as used in data acquisition, described in Section VI-A. In practice, unwanted oscillations occur in system response with Merrikh-Bayat $PI^\lambda D^\mu$ controller, while these oscillations are strongly damped in the system with a neural $PI^\lambda D^\mu$ controller.

The actual control system response achieved 0.77% of overshoot and a settling time of 11787 s for the neural $PI^\lambda D^\mu$ controller. For the Merrikh-Bayat approximation, it achieved an overshoot of 1.80% and a settling time of 43086 s. It was observed that in steady state, both systems present a small-amplitude oscillations around the desired setpoint.

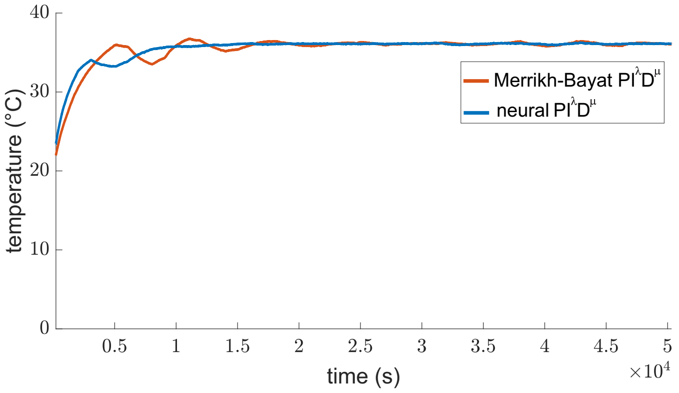


Fig. 11: Actual temporal responses of the evaluated embedded control systems.

VIII. Conclusion

This work presented a promising approach by implementing a $PI^{\lambda}D^{\mu}$ controller through an MLP network, which has shown to be effective in the regression task, offering an alternative to $PI^{\lambda}D^{\mu}$ controller discretization methods.

The incubator transfer function obtained with the RLS estimation algorithm using real data achieved RMSE = 0.0598. The optimization process of the parameters of the $PI^{\lambda}D^{\mu}$ controller by the PSO algorithm took 97 hours to achieve the result, given the large computational effort involved in simulating the $PI^{\lambda}D^{\mu}$ controller derivatives and fractional integrals, which are infinite order filters, for a 20000 s running time for each particle.

In a simulation environment, the neural $PI^{\lambda}D^{\mu}$ controller achieved a better approximation (RMSE = 0.0660) than the Merrikh-Bayat discretization method (RMSE = 0.6578) for the same input dimension. In a real environment, the control system with the neural $PI^{\lambda}D^{\mu}$ controller achieved an overshoot of 0.77% and a settling time of 11787 s, while the Merrikh-Bayat $PI^{\lambda}D^{\mu}$ controller achieved an overshoot of 1.80% and a settling time of 43086 s.

The embedded neural network has 7 inputs, 4 neurons in the hidden layer, 2 parameters for normalization and 2 for denormalization, occupying 1556 bytes of program memory and 241 of RAM during its execution. Additionally, the 8-bit microcontroller requires only 6511 μs to run the MLP, a much smaller time than the control system update time.

In future work, it is intended to apply the same methodology to other real plants, such as non-linear plants and short settling time plants, in order to evaluate the MLP network's capability to learn the behavior of the $PI^{\lambda}D^{\mu}$ controller for these cases.

Therefore, the use of an MLP neural network to implement a neural $PI^{\lambda}D^{\mu}$ controller does not require large computational efforts or large memory spaces on the embedded system. Thus, this approach is very promising when it is desired to embed the $PI^{\lambda}D^{\mu}$ controller and maintain the optimality achieved for the controller in continuous time.

References

[1] P. Shah and S. Agashe, "Review of fractional PID controller," *Mechatronics*, vol. 38, pp. 29–41, 2016.

[2] A. A. Jamil, W. F. Tu, S. W. Ali, Y. Terriche, and J. M. Guerrero, "Fractional-order PID controllers for temperature control: A review," *Energies*, vol. 15, no. 10, p. 3800, 2022.

[3] V. Feliu-Batlle, R. Rivas-Perez, and F. J. Castillo-Garcia, "Robust fractional-order temperature control of a steel slab reheating furnace with large time delay uncertainty," in *ICFDA'14 International Conference on Fractional Differentiation and Its Applications 2014*. IEEE, 2014.

[4] I. Petráš and B. Vinagre, "Practical application of digital fractional-order controller to temperature control," *Acta Montanistica Slovaca*, vol. 7, no. 2, pp. 131–137, 2002.

[5] R. S. Barbosa, J. T. Machado, and I. S. Jesus, "Effect of fractional orders in the velocity control of a servo system," *Computers & Mathematics with Applications*, vol. 59, no. 5, pp. 1679–1686, 2010.

[6] M. K. Bouafoura and N. B. Braiek, "$PI^{\lambda}D^{\mu}$ controller design for integer and fractional plants using piecewise orthogonal functions," *Communications in Nonlinear Science and Numerical Simulation*, vol. 15, no. 5, pp. 1267–1278, 2010.

[7] J.-Y. Cao, J. Liang, and B.-G. Cao, "Optimization of fractional order PID controllers based on genetic algorithms," in *2005 international conference on machine learning and cybernetics*, vol. 9. IEEE, 2005, pp. 5686–5689.

[8] A. Abraham, A. Biswas, S. Das, and S. Dasgupta, "Design of fractional order $PI^{\lambda}D^{\mu}$ controllers with an improved differential evolution," in *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, 2008, pp. 1445–1452.

[9] K. Vanchinathan and N. Selvaganesan, "Adaptive fractional order PID controller tuning for brushless DC motor using artificial bee colony algorithm," *Results in Control and Optimization*, vol. 4, p. 100032, 2021.

[10] P. W. Oliveira, G. A. Barreto, and G. A. Thé, "A general framework for optimal tuning of PID-like controllers for minimum jerk robotic trajectories," *Journal of Intelligent & Robotic Systems*, vol. 99, pp. 467–486, 2020.

[11] I. Podlubny, "Geometric and physical interpretation of fractional integration and fractional differentiation," *arXiv preprint math/0110241*, 2001.

[12] A. J. Calderón, B. M. Vinagre, and V. Feliu, "Fractional order control strategies for power electronic buck converters," *Signal Processing*, vol. 86, no. 10, pp. 2803–2819, 2006.

[13] F. Merrikh-Bayat, N. Mirebrahimi, and M. R. Khalili, "Discrete-time fractional-order PID controller: Definition, tuning, digital realization and some applications," *International journal of control, automation and systems*, vol. 13, pp. 81–90, 2015.

[14] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[15] A. Asgharnia, A. Jamali, R. Shahnazi, and A. Maheri, "Load mitigation of a class of 5-MW wind turbine with RBF neural network based fractional-order PID controller," *ISA transactions*, vol. 96, pp. 272–286, 2020.

[16] V. K. Munagala and R. K. Jatoth, "A novel approach for controlling DC motor speed using NARXnet based FOPID controller," *Evolving Systems*, vol. 14, no. 1, pp. 101–116, 2023.

[17] A. G. Perry, G. Feng, Y.-F. Liu, and P. C. Sen, "A design method for pi-like fuzzy logic controllers for DC–DC converter," *IEEE transactions on industrial electronics*, vol. 54, no. 5, pp. 2688–2696, 2007.

[18] S. S. Haykin, *Adaptive filter theory*. Pearson Education India, 2002.

[19] D. Xue, *Fractional-order control systems*. de Gruyter, 2017.

[20] Y. Shi and R. C. Eberhart, "Empirical study of particle swarm optimization," in *Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406)*, vol. 3. IEEE, 1999, pp. 1945–1950.

[21] H.-L. Hung, Y.-F. Huang, C.-M. Yeh, and T.-H. Tan, "Performance of particle swarm optimization techniques on PAPR reduction for OFDM systems," in *2008 IEEE International Conference on Systems, Man and Cybernetics*. IEEE, 2008, pp. 2390–2395.

[22] M. P. Aghababa, "Optimal design of fractional-order PID controller for five bar linkage robot using a new particle swarm optimization algorithm," *Soft Computing*, vol. 20, no. 10, pp. 4055–4067, 2016.

[23] M. A. El-Shafei, M. I. El-Hawwary, and H. M. Emara, "Implementation of fractional-order PID controller in an industrial distributed control system," in *2017 14th International Multi-Conference on Systems, Signals & Devices (SSD)*. IEEE, 2017, pp. 713–718.