Intelligent Information Management, 2022, 14, 165-193
https://www.scirp.org/journal/iim
ISSN Online: 2160-5920
ISSN Print: 2160-5912

Implementation on *FPGA* of Neuro-Genetic *PID* Controllers Auto-Tuning

Nícolas Rosa[1], Marcio da Silva Arantes[1], Claudio Fabiano Motta Toledo[1*], João Miguel G. Lima[2]

[1]Institute of Mathematics and Computer Science, University of São Paulo, São Carlos, Brazil
[2]Department of Electronic Engineering and Informatics, University of Algarve, Campus de Gambelas Faro, Portugal
Email: *claudio@icmc.usp.br

How to cite this paper: Rosa, N., da Silva Arantes, M., Toledo, C.F.M. and Lima, J.M.G. (2022) Implementation on *FPGA* of Neuro-Genetic *PID* Controllers Auto-Tuning. *Intelligent Information Management*, **14**, 165-193.
https://doi.org/10.4236/iim.2022.145012

Received: July 23, 2022
Accepted: September 26, 2022
Published: September 29, 2022

Abstract

The present paper studies the use of genetic algorithm to optimize the tuning of the Proportional, Integral and Derivative (PID) controller. Two control criteria were considered, the integral of the time multiplied by the absolute error (ITAE), and the integral of the time multiplied by the absolute output (ITAY). The time variant plant tested is a first-order plant with time delay. We aim at a real time implementation inside a digital board, so, the previous continuous approach was discretized and tested; the corresponding control algorithm is presented in this paper. The genetic algorithms and the PID controller are executed using the soft processor NIOS II in the Field Programmable Gate Array (FPGA). The computational results show the robustness and versatility of this technology.

Keywords

Genetic Algorithms, PID Controller, FPGA Board, Neural Networks

1. Introduction and Related Works

Despite the *PID* controllers being used a long time ago, nowadays, these simple controllers are still one of the most common solutions to practical control problems. The popularity of this controller comes from its simple structure, only three terms to tune, and robust performance over a wide range of operation conditions.

In the beginning of the 20th century, when windmills and steam engines were the dominant technologies, controllers with proportional and integral action (PI) were used. The current form, *PID* controller, emerged in the 30s with the pneumatic controllers [1].

The early 21st century has seen a renewed interest in research in *PID* control.

The book "*PID* Control in the Third Millennium" [2] provides an overview of the advances in this technology field. This publication is of interest both for academics requiring a reference for the current state of *PID* research or industrial practitioners and manufacturers of control systems with application to problems with *PID,* which can find on this book a practical source of appropriate and advanced solutions.

Due to its universal and multipurpose use, several states of the arts have been written; in 2016, a very complete overview and analysis of patents, software, and hardware for *PID* control were published [3].

Over the years, the great popularity of *PID* controllers has led to the development of several tuning methods, which are simultaneously simple to use and fast to run. Due to changes on operation conditions and plant components aging, tuning a controller is a dynamic procedure, this means that any control loop needs, in general, to be returned during its normal running.

Since 1942 with the establishment of empirical tuning rules by Ziegler and Nichols [4] until now, people coming from academia as the industrial field have dedicated a significant effort in developing a huge range of *PID* tuning technics.

In 2008 it was presented by Kocijan an accurate survey of the methods for *PID* auto-tuning was proposed as patents [5]. According to this survey, the main groups for the patents are relay tuning, multiple controllers for nonlinear processes, methods and systems based on model-based tuning with non-parametric and parametric models. We realize that the majority of *PID* control approaches are either characterized by the use of semi-empirical rules or derived from model-based methods employing low-order data-based models [6]. We can conclude that strong guarantees on the real system are difficult to obtain, so only suitable tuning for the application at hand is provided where a generalization is a difficult task. To overcome these problems, the researchers have investigated data-based controller tuning techniques to design suitable feedback controllers directly from data without the need to identify a system model. Based on this new paradigm, several methods have been developed since the early 21st century, such as the Virtual Reference Feedback Tuning *VRFT* method [7] [8] [9] and the Correlation-based Tuning method [10].

Artificial intelligence has played an increasing role in the control systems leading to intelligent control since the beginning of 21st century. According to this approach, classical control algorithms can be combined with soft computing techniques as artificial neural networks, genetic algorithms [11], or fuzzy logic. However, some drawbacks are founded by using these technologies. The artificial neural networks suffer from the convergence time and the excessive length required for the training set, the fuzzy logic systems depend on the experience of the designer in tuning the membership functions, and genetic algorithms could spend a hard computing effort if we deal with a big population or a complex objective function.

One example of the solution obtained by joint use of different techniques can be found in [11], where neural networks trained off-line were used for supplying on-line *PID* parameters optimized for arbitrary control criteria. The *NN* was used in such an approach for modeling and the classical *GA* for optimization purposes. It was evident that *GA* requires a great computational effort which is incompatible to run the control system in real-time. Classical optimization algorithms, based on gradient and aiming to minimize the objective function, often get trapped in the local minima. This can be overcome by stochastic algorithms like *GA* by combining *GA* with *PID* controller for auto-tuning optimization. This approach leads to an adaptive *PID* control which is widely used in applications for a huge range of practical problems. For instance, a multi-objective genetic algorithm, combined with the Taguchi method, was used for the optimum PID controller design applied to an automatic voltage regulator [12].

Therefore, an improvement was performed [13] by using a multi-population genetic algorithm *MPGA.* According to this approach, three populations evolve separately following the evolutionary model of islands [14]; the individuals are hierarchically structured in trees inside each population. As expected, this approach [13] overcomes the former [11] in terms of optimizing GA efficiency, which is a relevant contribution to real-time implementation. There are several examples where *MPGA* was used successfully, such as [15], where a solution for the asymmetric traveling salesman problem was proposed. A similar *MPGA* approach is used in [16] to solve the Gene Ordering Problem. In [17] and [18], this kind of GA is used to solve planning problems for two independent production scenarios.

The revision of the literature shows us that, no matter the approach applied, most *PID* controller design was implemented in software. This work uses the MPGA with *NN* to support the *PID* autotuning using the soft-processor NIOS II inside an FPGA board as a platform. This approach presents the advantage of the possibility of the designer of embedded systems defining a specific core inside the NIOS II for her/his specific needs [19]. The low power consumption when we use an *FPGA* board [20] compared to the *PC* power is indubitably another advantage. Despite *MPGA* running more efficiently in high-performance processors installed in a *PC* than in an *FPGA* board, the high demanded power required by a *PC* makes some applications using embedded systems unfeasible. Therefore, the current step of our research justifies this paper because previous tests performed in a *MATLAB* environment now are transposed and tested into digital hardware. A former approach of *VRFT* considers a stochastic setup where the involved processes are stationary and evolve in discrete time. This method was reformulated to better fit into the framework of *PID* control design for industrial use [21], specifically, the signals are treated as deterministic in continuous time.

During an adaptive control procedure, the system identification of problems is one of the main issues that should be considered. An adaptive *PID* controller should deal with systems linear or nonlinear, time-invariant or time-variant, integer or fractional-order. This last kind of system is modeled by fractional diffe-

rential equations containing derivatives of non-integer order, which leads to a non-rational transfer function. Fractional-order model often provides a more reliable description for some dynamic processes when compared with integer-order models. The fractional-order models are particularly suitable for systems such as heating furnaces [22], flexible structures [23], and materials with memory and hereditary effects [24]. They are also suitable for some electrical circuit with passive elements named fractance [25] that implements a fractional-order behavior. This kind of electronic device is built into blocks for synthesizing fractional order controllers, and it is characterized by impedance, $Z(j\omega)=(j\omega)^{\alpha}$, $-1\le\alpha\le 1$, which ω is the angular frequency. A Titanium billet heating process was given in [26] to illustrate the identification and *PID* control of a delay fractional-order system. In this example, the optimal *PID* controller was obtained by minimizing the integral of time multiplied by absolute error (*ITAE*).

The adaptive *PID* control to assist mobile robots has been widely used in many scenarios. In the field of agriculture and forestry, the use of adaptive *PID* control to regulate the trajectory of a spray robot inside a greenhouse was investigated in [27]. According to this approach, a parallel *PID* controller is used in a negative feedback control loop. The robot trajectory control is affected by many uncertain factors inside a greenhouse, and using an adaptive algorithm such as adaptive *PID* is compulsory. The experiments using an adaptive PID controller and GA in [27] achieve a slow response which could be a problem for emergency obstacle avoidance. Thus, a specific architecture implemented in hardware would accelerate the whole procedure and could overcome this problem. The work in [28] introduces PID controllers with C-Mantec algorithm, which is validated in cancer detection where satisfactory results are reported. The hardware viability of the method was also evaluated by synthesizing its code.

As already mentioned, a multi-population genetic algorithm *MPGA* improved results for *PID* controller as reported in [13]. The *MPGA* employs a model of an island for evolving solutions that have been applied in different problems such as the asymmetric traveling salesman in [15], the Gene Ordering Problem in [16], and production planning problems in [17] and [18]. The performance of *MPGA* with individuals hierarchically structured in trees was also compared against other optimization methods in [29]. The authors report superior results using *MPGA* when finding solutions for benchmark multi-modal functions. Besides using an MPGA as an optimization technique, the present paper will advance by proposing algorithms to execute the *PID* control in real time, which means an adaptive control as described in [12] and [27].

This paper has the following outline: the control loop and the tuning module are presented in section 2. Our system is implemented in *FPGA*. Thus, a discrete version is needed, so this section is devoted to the discretization of the plant and the controller, after that, the algorithm for *FPGA* implementation is deducted. Section 3 is devoted to *GA*, where two approaches are pointed out. In the first one, *GA* is combined with *NN* to estimate the fitness function; in the second approach, the *GA* estimates the fitness function without the need for *NN*. As ex-

plained before, *NN* is used for modeling purposes, and the optimization technique adopted can use either these models to evaluate the fitness function (first approach), or directly the signals produced in runtime by the control loop (second approach). For both approaches, *FPGA* implementations are described. Section 4 presents some computational results, first from *MATLAB* environment and after from the *FPGA* implementation. Section 5 points out some conclusions and suggestions for future work.

2. *PID* Controller Tuning Problem

The control architecture was detailed in [29], where only continuous time was considered, for convenience purposes such architecture is repeated (**Figure 1**).

Where

$$PI(s) = k_c\left(1+\frac{1}{st_i}\right) \quad \text{and} \quad D(s) = \frac{1+st_d}{1+st_f}, \quad t_f = \frac{t_d}{10} \tag{1}$$

The *PID* auto-tuning consists of evaluating and delivering online accurate PID parameters optimized for control criteria. As shown in **Figure 1**, it should be considered a simple control loop (the block *control system*), in connection with the *tuning block* composed of the *modeling* and *optimizer* blocks, as explained in [29].

The same two objectives will be considered as in the previous approach:

1) reference tracking,

2) output disturbance rejection.

These objectives are respectively achieved by minimizing:

1) The integral of the time multiplied by the absolute error: $ITAE = \int t\left|e(t)\right|\mathrm{d}t$, $e(t) = y(t) - r(t)$, with a unit step as input ($R(s) = 1/s$),

2) The integral of the time multiplied by the absolute output $y(t)$, $ITAY = \int t\left|y(t)\right|\mathrm{d}t$ with a null reference ($R(s) = 0$) and a unit step added to the $G(s)$ output.

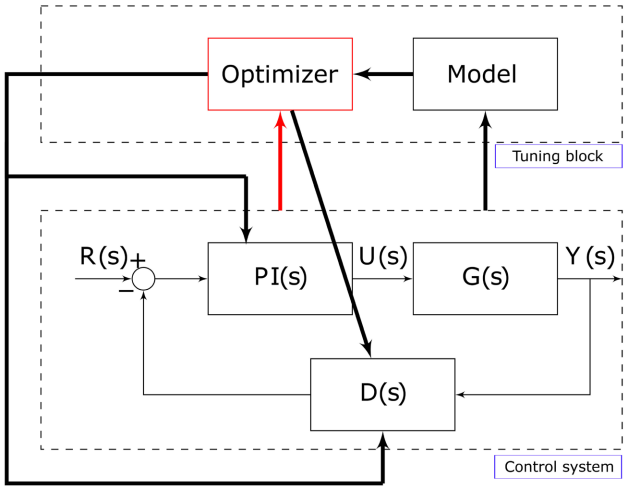

Figure 1. Control system + tuning block.

2.1. The Plant

The plant under test will be linear with time delay, represented by the transfer function $\boldsymbol{G}(s)$ modeling a continuous system. The exact expression for this transfer function will be defined in section 3.3 where the control algorithm will be outlined.

This approach is valid because most of the dynamic systems to be controlled are continuous, as described by the continuous transfer function of Laplace variable $\boldsymbol{s}$ [30].

However, since the control will be performed by a digital processor, the interface between the continuous and discrete domains should be taken into account. Thus, the plant $\boldsymbol{G}(s)$ together with both converters (the analog to digital (*A/D*), and the digital to analog (*D/A*)), is depicted in **Figure 2**.

The current approach aims at the real time discrete control assisted by a processor, and we should compute the discrete transfer function between the samples coming from the processor, $u[kT]$, and the samples picked up from the plant output by the A/D converter, $y[kT]$; we will represent this transfer function by $G_d(z)$. The sampling time T was used for sampling procedure.

The *D/A* converter is an electronic device called zero-order hold (*ZOH*) because it accepts a sample at a given instant, $t = kT$, let call it $u[kT]$, and holds its output constant until the next sample is sent at $t = kT + T$. According to this procedure, the *D/A* converter generates a continuous signal $x(t)$ with a shape like a stair with steps wide equals to sampling time *T*. Therefore, the transfer function that we want evaluate, $G_d(z)$, is the z-transform of the signal $y[kT]$ when the input $u[kT] = \delta[kT]$ (the discrete delta Dirac impulse), so, we can compute $G(z)$ following the next steps:

1) Knowing the shape of $u[kT] = \delta[kT]$ and the role of *D/A* we can define $x(t)$.

2) Evaluation $X(s)$ (the Laplace transform of $x(t)$).

3) Evaluation $Y(s) = X(s)G(s)$.

4) $y(t)$ is obtained by inverting the Laplace transform $Y(s)$: $y(t) = \boldsymbol{L}^{-1}\{Y(s)\}$.

5) $y[kT]$ is obtained from the continuous signal $y(t)$ for $t = kT$: $y[kT] = y(t)\big|_{t=kT}$

6) $G_d(z)$ is the z-transform of $y[kT]$: $G_d(z) = \boldsymbol{Z}\{y[kT]\}$.

Moreover, (see **Figure 2**) following the above steps, we can obtain [29] the discrete transfer function from the input $u[kT]$ to output $y[kT]$ by (2).

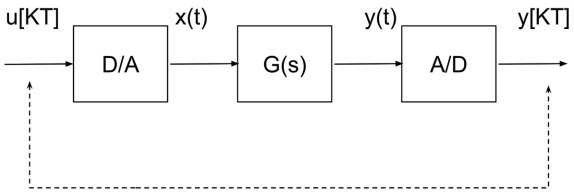

Figure 2. The prototype sampled-data system.

$$G_d(z) = \left(1 - z^{-1}\right) \mathbf{Z}\left\{\frac{G(s)}{s}\right\} \tag{2}$$

The symbol $\mathbf{Z}\{.\}$ means that the Laplace transform (expression in variable *s*) should be inverted to the continuous time domain, after, it will be taken for $t = kT$ and finally evaluated its z-transform.

As mentioned before, the plant under test should model a linear system with time delay; we rewrite *G*(*s*) as $G(s) = \mathrm{e}^{-\lambda s} H(s)$ for practical reasons.

Let us assume that time delay λ is greater than the sampling time *T*: $\lambda > T$. We define λ in terms of *T*:

$$\lambda = lT - mT \tag{3}$$

where *l* is the minimum number of *T* needed to transcend the time delay λ, so: $l = Ceil(\lambda/T)$, (*Ceil* round towards plus infinity). Thus, *m* is a fractional part of *T*, it can be evaluated after knowing *l*, λ and *T*. Finally, we can deduct [29] from (2):

$$G_d(z) = \left(1 - z^{-1}\right) z^{-l} \mathbf{Z}\left\{\mathrm{e}^{mTs} \frac{H(s)}{s}\right\} \tag{4}$$

In spite of both expressions (2) and (4) be the same, the use of (4) is preferable when we deal with time delay systems, so, it will be adopted further.

2.2. The *PID* Control

The adopted control technique is the discrete equivalent of continuous controllers [31]. This is an indirect method:

- We started with a continuous time design [29] (in this case a continuous *PID* controller) and then (actual work) we make a discretization because it should be implemented in a processor; this method of design is called emulation.

Among the ways that we have to discretize the analog controller, we choose the numerical integration method. According to this technique, the integrals obtained from the differential equations are approximated by differences, leading to differences equations, which are models of discrete systems.

The approximation chosen was the bilinear transformation (also called trapezoidal integration or Tustin transformation) [31] within the numerical integrations method. Therefore, it is proved [30] that, a continuous transfer function leads to a discrete equivalent transfer function by replacing the Laplace variable ***s*** according to (5).

$$s \leftarrow \frac{2}{T} \frac{1 - z^{-1}}{1 + z^{-1}} \tag{5}$$

Thus, using the emulation design explained before, the discrete *PID* controller is obtained from (1) using (5):

$$PI_d(z) = k_c \frac{(2t_i + T)z - 2t_i + T}{2t_i z - 2t_i} \tag{6}$$

$$D_d(z)=\frac{(2t_d+T)z-2t_d+T}{(2t_f+T)z-2t_f+T},\quad t_f=\frac{t_d}{10} \tag{7}$$

Now, we have all the blocks needed to establish the discrete version (see **Figure 3**) of the continuous *control system* presented in the bottom of **Figure 1**.

In **Figure 3** we locate the signals defined in the discrete time domain (k variable), which will be useful for the establishment of the algorithm. The discrete negative feedback control loop together with the z transform (6), (7), and $G(z)$, will be used to define the algorithm which was implemented on the processor. This procedure will be detailed in the next section. Note that for simplicity purposes the d sub-index for the discrete transfer function (4) (6) and (7) will be suppressed in the following.

2.3. The Discrete Control Loop Algorithm

The algorithm to be implemented in the processor should take into account the kind of plant that we will use. In the sequence of [29], the continuous plant will be a first-order plant with a time delay, FOPDT (8).

$$G(s)=\frac{\mathrm{e}^{-\lambda s}}{s+a} \tag{8}$$

In the previous approach [29], we considered a time-invariant plant ($a=1,\lambda=1$), now we extend the work to accommodate small changes for the polo location a, simulating a time-variant plant. Thus, taking into account the time delay λ decomposed according to (3) and the sampled-data system (**Figure 2**); the continuous plant (8) leads to a discrete transfer function (9).

$$G_d(z)=\frac{(1-\alpha)z-\mathrm{e}^{-aT}+\alpha}{az^{l+1}-a\mathrm{e}^{-aT}z^{l}},\quad \alpha=\mathrm{e}^{-amT} \tag{9}$$

Now, we have all the transfer functions (6), (7), and (9) for the discrete version of the continuous control system (**Figure 3**), so, we can establish different equations for the signals $u[k]$, $y[k]$ and $m[k]$. Thus, using (6), (9) and (7) we have (10), (11) and (12) respectively.

$$u[k]=u[k-1]+k_c\frac{2t_i+T}{2t_i}e[k]+k_c\frac{T-2t_i}{2t_i}e[k-1] \tag{10}$$

$$y[k]=\mathrm{e}^{-aT}y[k-1]+\frac{1-\alpha}{a}u[k-l]+\frac{\alpha-\mathrm{e}^{-aT}}{a}u[k-(l+1)] \tag{11}$$

$$m[k]=\frac{2t_f-T}{2t_f+T}m[k-1]+\frac{2t_d+T}{2t_f+T}y[k]+\frac{T-2t_d}{2t_f+T}y[k-1] \tag{12}$$

Based on these deference equations we can deduct the discrete close loop algorithm.

In spite of this algorithm is implemented into a processor, it was before validated with a *Simulink* model which is the discrete counterpart of the continuous system presented in [29].

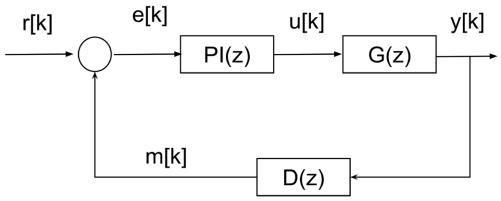

Figure 3. The discrete version of the continuous control system.

Algorithm 1. Discrete close loop algorithm.

1. Initialize:
2. Time Delay
3. Sampling Time
4. number Of Points
5. $l \leftarrow$ Time Delay/Sampling Time
6. frac_m % is m according expression (**3**)
7. frac_m $\leftarrow l$ – Time Delay/Sampling Time
8. $\alpha \leftarrow \mathrm{e}^{-a \times \text{frac\_m} \times \text{SamplingTime}}$
9. $m \leftarrow [0 \quad 0]$
10. For cont $\leftarrow$ 1 to number Of Points – 1
11. $k \leftarrow$ cont+1
12. $e(k) \leftarrow 1 - m(k)$
13. If $k = 2$ then
14. $u[k] = k_c \dfrac{2t_i + T}{2t_i} e[k]$ % expression (10)
15. else
16. $u[k] = u[k-1] + k_c \dfrac{2t_i + T}{2t_i} e[k] + k_c \dfrac{T - 2t_i}{2t_i} e[k-1]$ % expression (10)
17. End if
18. % expression (11)
19. If $k = 2$ then
20. $y[k] = 0$
21. Else if $k \geq 2$ and $k \leq l$
22. $y[k] = \mathrm{e}^{-aT} y[k-1]$
23. Else
24. $y[k] = \mathrm{e}^{-aT} y[k-1] + \dfrac{1-\alpha}{a} u[k-l-1] + \dfrac{\alpha - \mathrm{e}^{-aT}}{a} u[k-l]$
25. End if
26. % expression (12)
27. If $k = 2$ then
28. $m[k] = \dfrac{2t_d + T}{2t_f + T} y[k]$
29. Else
30. $m[k] = \dfrac{2t_f - T}{2t_f + T} m[k-1] + \dfrac{2t_d + T}{2t_f + T} y[k] + \dfrac{T - 2t_d}{2t_f + T} y[k-1]$
31. End If
32. $m[k+1] = m[k]$
33. End For

3. Proposed Methods

A Multi-Population Genetic Algorithm (MPGA) is applied to solve the tuning problem approached. Algorithm 2 has the pseudocode of the *MPGA*.

Algorithm 2. Multi-Population genetic algorithm.

```
1.  Repeat
2.      for i ← 1 to nPop do
3.          initialize(pop(i));
4.          evaluate (pop(i));
5.          structure(pop(i));
6.  repeat
7.      for j ← 1 to crossRate*popSize do
8.          (ind1,ind2) ← selectedParents(pop(i));
9.          newInd ← crossover(ind1,ind2);
10.     if λ • mutatioRate then
11.         newInd ← mutation(newInd);
12.         evaluate(newInd);
13.         insert(newInd, pop(i));
14.     end for
15.         structure (pop(i));
16.     until convergence(pop(i));
17.         executeMigration(pop(i),pop(i+1));
18.     until stopCriterion;
```

The individuals used by *MPGA* are *PID* parameters in the context presented on section 3. Thus, each individual *i* is encoded as $ind_i = \left[k_c \quad t_i \quad t_d\right]$ and, the full amount of individuals will be distributed by a total of nPop populations which be used on *MPGA* running. The populations are initialized through lines 2-5 in Algorithm 1. First, the procedure *initialize*() in line 3 generates the individuals of each population from random values uniformly distributed such that $k_c, t_i, t_d \in \left[Min, Max\right]$. Next, the fitness of each individual is calculated by *evaluate*() function in line 4. The fitness function follows Equation (13):

$$Fitness\left(k_c \quad t_i \quad t_d\right) = w_1 F\_ITAE\left(k_c, t_i, t_d\right) + w_2 F\_ITAY\left(k_c, t_i, t_d\right), \tag{13}$$

where w_1 and w_2 are weights and functions *F_ITAE*() and *F_ITAY*() return the *ITAE* and *ITAY* values, respectively. Finally, the individuals on each population are hierarchically structured in trees from *structure*() by line 5. Figure 4 illustrates a population where the individuals are hierarchically structured in trees.

The individuals are represented as nodes with their fitness values as shown by Figure 4 for a minimization problem. The ternary tree structure is employed since each node presents degree three, except by the leaves nodes. The three degree is

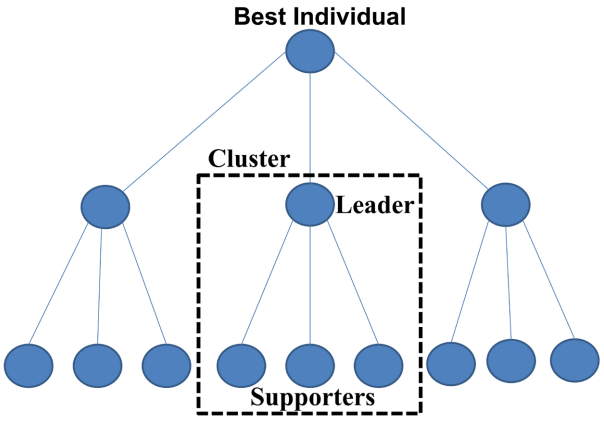

Figure 4. Population hierarchically structured in ternary tree.

defined based on empirical tests and previous results applying the MPGA in other problems as reported in [15] [16] [17] [18]. The procedure *structure*() will dispose of individuals hierarchically based on their fitness values. In this case, the better individuals will be placed as a parent node (cluster leader) with three supporter nodes within each cluster. Thus, the *structure*() arranges individuals keeping cluster leaders with better fitness value than their followers. In the whole population, the best individual found so far will be at the root node of such a hierarchical tree.

The evolving process happens from lines 6 to 16 in **Algorithm 1**. A total of *crossRate*popSize* new individuals are generated at each iteration (line 7). In this process, *crossRate* is the rate of individuals to be generated from *popSize* by applying crossover and mutation operators. The *selectedParents*() procedure selects two parents by choosing randomly a cluster leader and taking one of its followers in line 8. Next, three crossover operators are employed: *arithmetic* [4], *geometrical* [4] e *blend [2]*. For each pair of parents, one of these three crossover operators is randomly selected to be applied over the two parents in line 9, generating only one new individual. If the mutation rate is satisfied, the new individual is changed by *mutate*() procedure, which also selects randomly three possible mutation types (lines 10 and 11). All the mutation operators begin by choosing randomly one parameter of the individual. The first mutation replaces this parameter with another one randomly taken from [*Min*, *Max*]. The second mutation replaces by *Min* or *Max* value allowed for this parameter. The third mutation changes the parameter value following equation (14).

$$\text{new Value} = \max \text{Value} + \min \text{Value} - \text{old Value} \tag{14}$$

The new individual can be inserted into the population based on procedure *insert*() in line 13. If the new individual is better than one of their parents, it replaces the worst parent in the hierarchical tree. After the generation of *crossRate*popSize* individuals, the inserted individuals are hierarchically disposed in the population by *structure*() in line 15. If none of the *crossRate*popSize* new individuals is inserted through lines 7 - 14, a population convergence is assumed

and a *migration*() operator takes place in line 17. This operator sends the best individual from the current population pop(i) to the next one pop(i+1). The current population is reinitialized before the steps on lines 6-16, but the re-initialization procedure keeps its best individual, and the migrated one. All steps described are repeated until the stop criterion has been reached in line 18.

The *MPGA* previously described was coded using as the main tool the *NIOS II* processor embedded in *FPGA*. The hardware employed was the development kit DE2-70 from Altera-Cyclone II-EP2C70F896C6N-100 MHz, [31] whose features supported two core architectures. The schematic of the hardware is shown in Figure 5.

There is a control algorithm that updates the control parameters of *PID* and it is executed in the first processor, while a second algorithm triggers the *MPGA* to run in the second processor of Figure 5. In order to have the interaction of the *MPGA* with the *PID* controller algorithm, a shared region is demanded from which the *MPGA* sends $\begin{bmatrix} k_c & t_i & t_d \end{bmatrix}$ parameters for the controller tuning algorithm. Two solution approaches are introduced to manage the interactions between *MPGA* and the controller algorithm as explained in the next subsections.

3.1. Approach 1

The approach 1 combines *MPGA* with an artificial neural network (*ANN*) that is applied to estimate fitness values. This is done by taking the *PID* parameters encoded in each individual of *MPGA* and propagating them through the *ANN*. The authors in [11] evaluated several types of ANN, where a Multi-Layer Perceptrons (*MLP*) with a sigmoid activation function was chosen. Two *MLP* with the same topology (Figure 6) were trained off-line for mapping *PID* parameters into *F_ITAE*() and *F_ITAY*(), respectively.

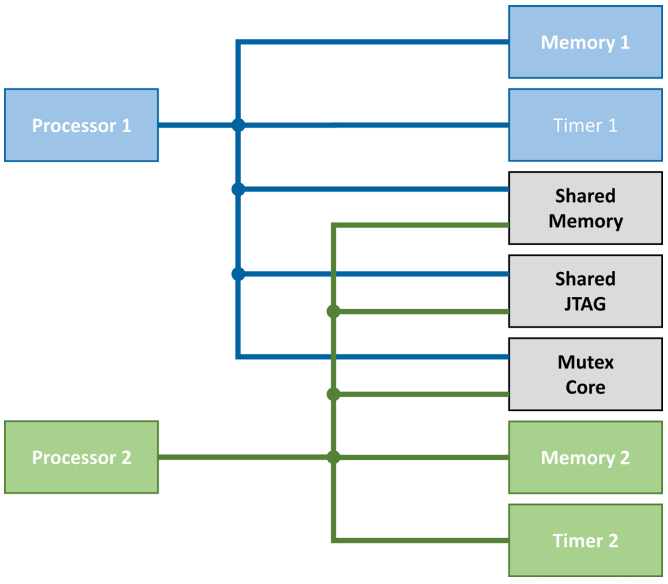

Figure 5. Hardware schematic—Nios II Multicore.

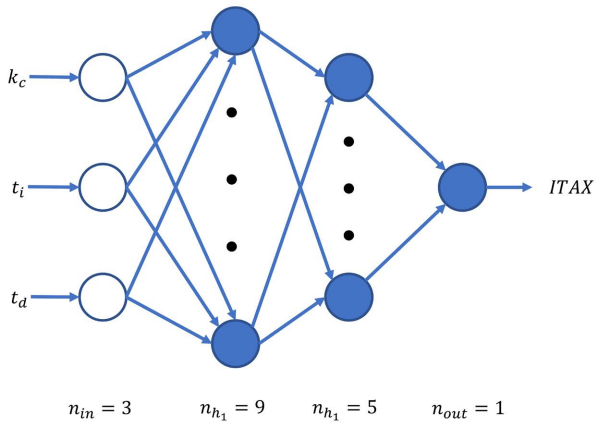

Figure 6. *MLP* for *F_ITAX*(k_c, t_i, t_d) estimation.

Experiments were previously conducted to look for the best generalization performance for MLP-ANN, which was reached by two hidden layers *MLP* with 9 and 5 neurons in the 1st and 2nd hidden layers, respectively. Details about the experiments to define the best topology are reported in [11]. The *MPGA* in this first approach uses the previously trained ANNs to estimate values for *F_ITAE*() and F_ITAY() terms in the fitness function (see Equation (13)). Thus, for each individual, its encoding (k_c, t_i, t_d) is propagated as input through the ANNs to find values for *F_ITAE*() and *F_ITAY*().

It could be too risky to initialize the *PID* control following the system proposed in **Figure 5**, without a feasible adjustment of values for parameters (k_c, t_i, t_d). The values for such control parameters could be first estimated and optimized next, avoiding an infeasible initial adjustment that damages the system to be controlled. Following this idea, the initial parameter adjust, represented as $ind^0 = ind\left(k_c^0, t_i^0, t_d^0\right)$, is set and its *F_ITAE*() and *F_ITAY*() values define the initial fitness value. At this point, MPGA will be in charge to evolve new parameter controls (individuals) able to improve this initial system configuration. There is a communication between MPGA and PID controller algorithm that allows updating the parameter values when the best individual is returned by MPGA. **Figure 7** shows how the data is sent during the communication between the two processes running in our approach, and **Algorithm 2** and **Algorithm 3** have the related pseudocode.

In **Algorithm 3**, the routines (*Mutex*) controlling the resource accesses are first initialized, and the core running the controller algorithm (core 1) keeps waiting for the first individual sent by MPGA to begin the controller algorithm. This first individual is $ind^0 = \left(k_c^0, t_i^0, t_d^0\right)$ which has the initial and feasible set of parameters. The core running MPGA executes the routine *ga_first_lock ()* in **Algorithm 4** that assures the first access to the *Mutex* for the MPGA. Next, the evolution process happens at core 2, aiming to find a better individual to replace the current one (ind^0). This is done by *execute(MPGA)* routine, which triggers the MPGA and updates the best individual found until the *stop_criterion* has been satisfied.

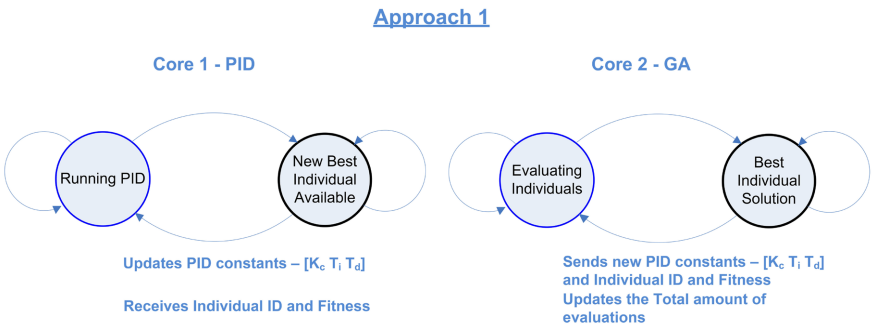

Figure 7. Shared information scheme—Approach 1.

Algorithm 3. Controlador PID—M1.

```
1.  mutex_initialize();
2.  if(mutex)then
3.      waiting_start_flag();
4.      update();
5.      while(stop_criterion)do
6.          communication();
7.          PID();
8.      end while
9.      while(period_end≠0)do
10.         PID();
11.     end while
12.     end if
```

Algorithm 4. Algoritmo Genético—M1.

```
1.  mutex_initialize();
2.  if(mutex)then
3.      ga_first_lock();
4.      while(stop_criterion)do
5.      sol.fitness← execute(MPGA)
6.      if(sol→fitness < best.fitness) then
7.          communication();
8.      end if
9.  end while
10. end if
```

The communication process occurs when MPGA finds a better individual (*best_individual*). First, core 2 stores data related to *best_individual* in the shared memory region, and the *flags* variables are updated. These variables indicate the individual availability for core 1. On the other hand, since an update is detected by core 1, the new parameter values are set by the PID controller algorithm.

However, as mentioned, the best individual returned by MPGA in Approach 1 was found by a fitness function that applies an MLP-ANN to estimate ITAY and ITAE. This best individual can have a lower fitness value when controlling the system for real or it can even be infeasible in the worst case. Thus, the parameters provided by the new individual are evaluated for a short period (*period_end*) in **Algorithm 3**. During this short period, the fitness value of the best individual sent by **Algorithm 4** is recalculated based on the real control of the system. If the recalculated fitness is better than the current one, the parameters given by the best individual become the current one in core 1. Otherwise, the previous parameters are kept. Also, this short period of evaluation is enough to avoid an infeasible individual damaging the system under control.

3.2. Approach 2

The approach 2 does not apply *MLP-ANN* to estimate fitness values. The method uses the control loop to evaluate all individuals needed by *MPGA*. The main advantage of this approach is to avoid the designing and training of ANN, which demands a previous database of the system under control for training and validation. Another advantage is a better evaluation of the individual since its parameters are effectively used to control the system. However, this approach leads to the main advantage once infeasible individuals can damage the system. In this case, it is necessary to find the trade-off between enough time controlling the system to evaluate the individual properly and enough time to avoid any damage to the system.

In our experiments, we set one wave cycle from the input signal $r[k]$ as the time to evaluate the parameters sent by *MPGA* for both Approach 1 and 2. An initial feasible individual is also set in approach 2 as done for approach 1. However, a s*aving factor* measure is defined to avoid any instability in the system that could compromise its functioning. The *saving_factor* works with upper and bound response limits of the system for signal $y[k]$. If the system's output is greater than the saving factor, the current set of parameters is replaced by the stable one in use by the controller algorithm. **Figure 8** gives an example where the saving factor was useful. The square wave $r[k]$ in blue is superimposed with output $y[k]$.

The saving factor is set to be activated when the difference (*max_diff*) between signals $y[k]$ and $r[k]$ is greater than 2. **Figure 8** reveals instability between 0.2 s and 0.25 s, so the saving factor was activated, leading the control algorithm to replace the unstable parameters with feasible ones, aiming to recover the system control, as we can see by the evolution of $y[k]$ after 0.25 s.

Figure 9 shows the communications process employed by Approach 2.

It is possible to observe that the first core now receives each individual to be evaluated and returns its fitness. The state machine representation of the *MPGA* in the second core illustrates the state changes, aiming at synchronizing with the first core. The second core stays within the inactive period due to the real-time

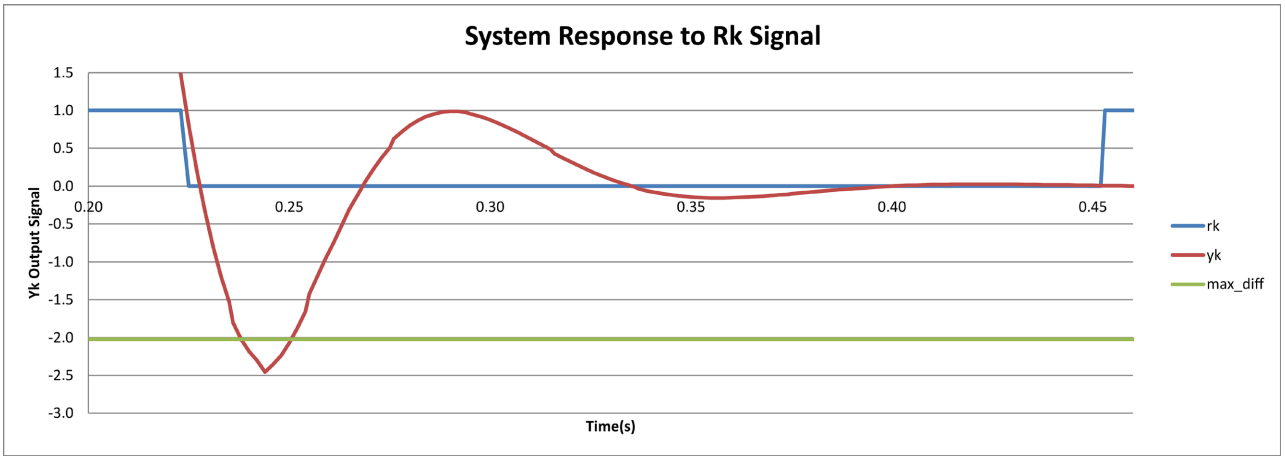

Figure 8. The violation of the saving factor.

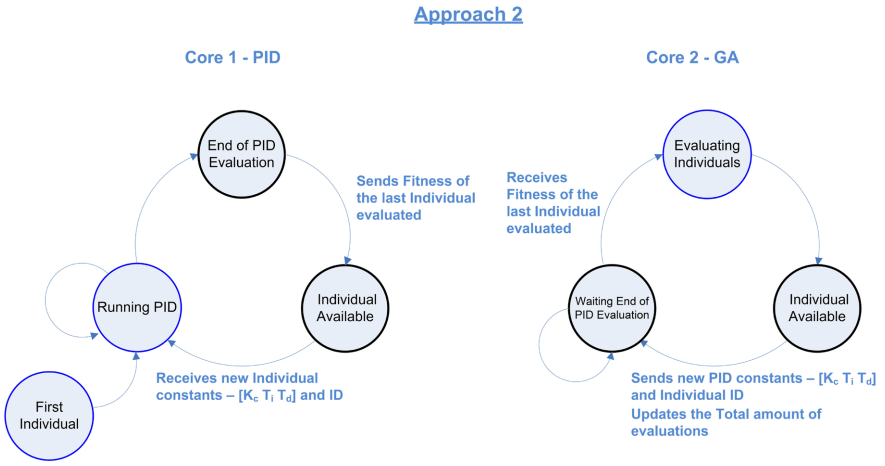

Figure 9. Shared information scheme—Approach 2.

evaluation of the individual (waiting *PID*). When the first core sends back the evaluated individual to the shared region (individual available), the second core wakes up to continue the evolving process.

As explained in Approach 1, when the whole system starts, both cores execute the routines that control access to resources (Mutex). The first core is still responsible for the controller algorithm (core 1) and it waits for the first individual to be sent by core 2. The second core is responsible for the optimization step and it performs the routines that guarantee it to be the first one to access the Mutex (ga_first_lock ()).

The feasible and first individual (ind0) is sent to the first core, and this core initializes the system control process (running_PID). The half cycle of the reference signal $r[k]$ is set as the evaluation period for each individual. If the time spent in the evaluation of an individual by *MPGA* is greater than half of the period, the individual loses the excitation to the step corresponding to him, and the configuration used in this period is replaced by the best individual saved by the condition of rescue. Thus, in order to take advantage of the whole wave, shortly

before the wave change, it was established that the best configuration takes control of the system. The period in which the individual is actually evaluated (period_SEND) is smaller than the evaluation period (period_EVAL). The remaining period is used to calculate and send the fitness of the current individual to the *MPGA*. If the remaining period is too long, there is a loss in the reliability of the fitness found since it is given a reduced time for the evaluation of the individual. On the other hand, if the period is short, the actions prior to the wave shift may not be performed. **Figure 10** illustrates the time periods mentioned and **Algorithm 5** and **Algorithm 6** give the pseudo codes implemented in both cores for approach 2.

It should be emphasized that this approach associates the fitness value after the execution of the *PID* control in real time. More, in the pseudocode of the *MPGA*, the routines responsible for sending and receiving the fitness do not present restriction to the communication as in the approach 1.

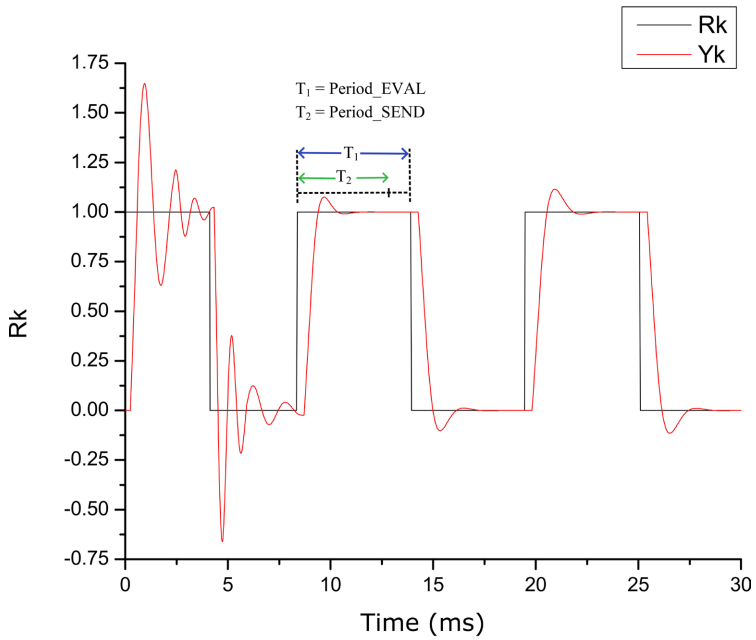

Figure 10. The send and evaluation time period in Approach 2.

Algorithm 5. PID Controller—Approach 2.

```
1.  mutex_initialize();
2.  if(mutex)then
3.      waiting_start_flag();
4.      while(stop_criterion)do
5.          communication(fitness);
6.          fitness = PID();
7.      end while
8.  end if
```

Algorithm 6. Optimization—Approach 2.

```
1.  mutex_initialize();
2.  if(mutex)then
3.  ga_first_lock();
4.  while(stop_criterion)do
5.      for i ← 1 to nPop do
6.      initialize(pop(i));
7.      for j ← 1 to nIndividuals do
8.          send_individual(ind_j);
9.          receive_individual(ind_j);
10.     structure(pop(i));
11.     repeat
12.     for j ← 1 to crossRate*popSize do
13.     (ind1,ind2)←selectedParents(pop(i));
14.     newInd ← crossover(ind1,ind2);
15.     if λ • mutatioRate then
16.         newInd ← mutation(newInd);
17.         send_individual(newInd);
18.         receive_individual(newInd);
19.         insert(newInd, pop(i));
20.     end for
21.     structure (pop(i));
22.     until convergence(pop(i));
23.     executeMigration(pop(i),pop(i+1));
24. end while
25. end if
```

4. Computational Results

In spite of last results were obtained from *FPGA* board [31], a *MATLAB*/*SIMULINK* *[32]* simulator was developed aiming to validate the corresponding algorithms. Therefore, preliminary results obtained from *MATLAB/SIMULINK* simulator are presented in subsection 5.1, and subsequent results obtained from *FPGA* board are presented in subsection 5.2.

4.1. Results from *MATLAB/SIMULINK* Simulator

The control loop represented in **Figure 3** is modeled by the systems implemented in *MATLAB/SIMULINK* as proposed in [29], these models were designed to evaluate the control criteria *ITAE* and *ITAY*.

The first results provided by these models were useful to evaluate the control criteria *ITAE* and *ITAY*, which validate the algorithm implemented in *FPGA* displayed in **Table 1**.

Table 1. Performance of the discrete equivalent system when the controller is tuned by the optimal values obtained in continuous time.

	PID	Continuous	Discrete	Δ(%)
ITAE	0.749 0.963 0.434	1.658	1.783	7.5
ITAY	0.664 0.903 0.479	1.150	1.262	9.7
ITAE_ITAY	0.714 0.964 0.424	1.429	1.543	8.0

As referred before, our methodologies aim at testing the performance of the *FPGA* implementation when the optimization proceeds in real continuous time. Therefore, we use the former investigation where the minima were obtained in continuous time [13] (**Figure 1**), to evaluate the distortion introduced into the discrete equivalent system, **Figure 3**. In other words, the problem could be formulated in the following terms: Let's x_m be the minimum found for the function f according to the approach [13] and f_d the discrete equivalent of f, so, we will evaluate the deviation between $f(x_m)$ and $f_d(x_m)$. According to the current approach, f is taken successively as control performance measures $ITAE$, $ITAY$ with $0.5ITAE + 0.5ITAY$ as presented in section 3, and x_m is the PID_m vector minimum evaluated for the corresponding functions. Finally, $f_d(x_m)$ is the discrete counterpart of the control performance measures, which is evaluated from the discrete loop presented in **Figure 3** tuned by PID_m.

The results obtained are displayed in **Table 1**; so, as reference values, we used the minima obtained by gradient method [13] for $ITAE$, $ITAY$ and $0.5ITAE + 0.5ITAY$; the corresponding minimized (x_m) are displayed in *PID* column and the correspondent minima for continuous time ($f(x_m)$) are shown in column *Continuous*.

It should be noted that several values for sampling time T were tested, so, we concluded that the maximum value of T allows acceptable behavior using the emulation control technique is $T = 0.1\,\mathrm{s}$. Therefore, the column *Discrete* displays $f_d(x_m)$; finally, last column shows the difference in percentage between $f(x_m)$ and $f_d(x_m)$.

4.2. Results from *FPGA Board*

The current approach was tested using *FPGA* Stratix II: EP2S60F672C3 Altera kit [31].

The preliminary tests performed into *MATLAB/SIMULINK* environment were

confirmed using *FPGA* board. Therefore, starting with sampling time $T = 0.1\,\text{s}$, we aim at study the behavior of the optimization and tuning procedure when we enlarge *T*. To perform this, it was used a square wave as reference $r[k]$, and the output $y[k]$, the response was observed during 30 s simulation, the control criterion to be optimized is *ITAE* and the *MPGA* was setup with the following parameters:

- independent populations: 3
- total individuals for each population: 7
- Crossover: 100%
- Mutation: 90%

Several values of sampling time were tested; the most significant results are presented in this paper. For $T = 0.1\,\text{s}$, we obtain the output displayed in **Figure 12** superimposed to the reference square wave.

It is clear that at the beginning the tuning is poor due to a large overshoot and settling time, however, after about 10 s the optimization converges leading to an acceptable overshoot.

Enlarging the sampling time we find an opposite behavior; for $T_s = 0.25\,\text{s}$ the output is displayed in **Figure 11**.

This graphic shows that $T_s = 0.25\,\text{s}$ is inadequate to this tuning because it leads to instability; despite this, the genetic algorithm was able to capture the tuning as demonstrated by the running after about 15 s. For this reason, in the FPGA implementation, it wasn't used sampling time greater than 0.1 s.

For both cases the procedure converges to
$PID_m = [k_c \quad t_i \quad t_d] = [0.581 \quad 0.942 \quad 0.108]$.

Due to the control criteria in use, *ITAE* and *ITAY* it is compulsory that half period of the square wave $r[k]$, be large enough to accommodate the settling time of the control system. If the square wave period is not large enough, the values obtained in real time for *ITAE* and *ITAY* are wrong and the corresponding fitness evaluated online is also wrong.

On the other hand, due to the fact that control parameters be evaluated online, the time disposed to do it is crucial. Therefore, we should avoid the next disturbance during the evaluation of the parameters; this can be reached by enlarging the square wave period.

4.2.1. Settling Time Consequences

It was set width between 5 s and 7 s for the test window. To show the influence of the frequency disturbance (square wave period, *T*) on the optimization procedure, we present **Figure 12** for two opposite situations, $T = 0.4\,\text{s}$ and $T = 0.2\,\text{s}$.

According to **Figure 12**, the square wave transitions for $T = 0.2\,\text{s}$ cuts the transient, consequently, the performance measures will be wrong evaluated. The fitness will be a mistake, the optimization procedure fails and the tuning will be wrong.

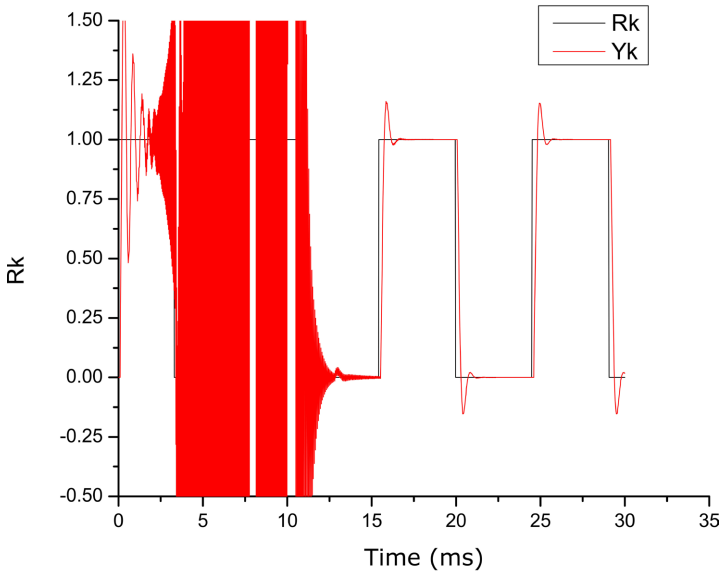

Figure 11. Square wave reference superimposed to the corresponding output for $T_s = 0.25\,\text{s}$.

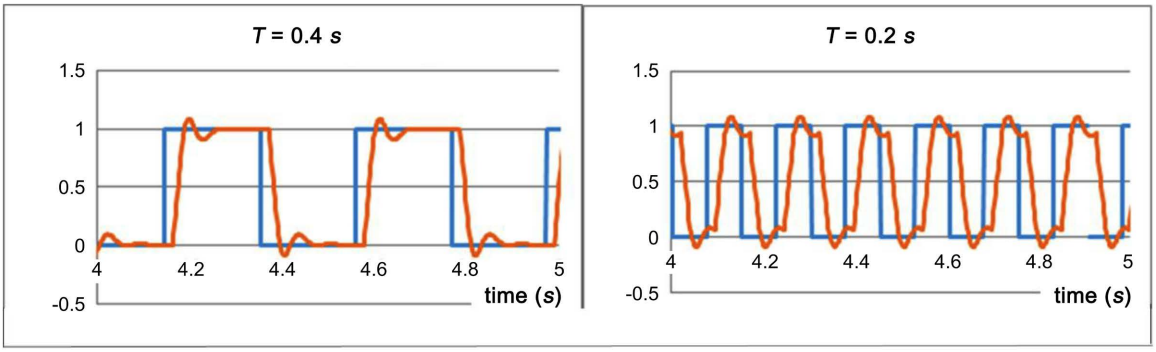

Figure 12. Square wave reference $r[k]$ superimposed to the corresponding output $y[k]$ for distinct values of period T.

Several tests were performed for the system with a transfer function $G(s) = \frac{e^{-s}}{s+1}$; it was used $T_s = 0.1\,\text{s}$ for sampling time. The controller was tuned for the optimal value of $ITAE\_ITAY$ evaluated by previously trained NN; in this case we obtained fitness $F_o = 2.1611$ corresponding to the individual $PID_o = [0.9467 \quad 1.4306 \quad 0.2540]$.

The suitability of the T for fitness evaluation is reported in Table 2.

The optimal online evaluated fitness for T large enough to extinct the transient is 2.1672, so, we can conclude that NN presents a relative error $100 \times \frac{|F_o - 2.1672|}{2.1672} = 0.28\%$.

As expected, reducing T leads to a decreasing (and incorrect) value of the fitness. In the 3rd column, we can see the relative error between the fitness evaluated for each T and the correct one evaluated numerically. Therefore, we can

Table 2. On-line fitness accuracy and square wave period.

$T(s)$	Fitness on-line	Relative error (%)
0.5612	2.1672	0.00
0.2638	2.1672	0.00
0.2203	2.1672	0.00
0.1969	2.1671	0.01
0.1853	2.1646	0.12
0.1577	2.1340	1.54
0.1411	2.0567	5.10
0.1121	1.9992	7.75
0.0750	1.3823	36.22

conclude for the tested plant that it is needed square wave plateau greater than or equal to $L = 0.19686/2$ s for the fitness evaluated online has been considered correct. In fact, rows corresponding to values greater than or equal to L present a relative error negligible.

So, if disturbances occur not faster than L s, the tuning methodology is accurate.

We also note that for every test the optimal individual,
$PID_o = [0.9467 \quad 1.4306 \quad 0.2540]$, was reached due to the fact that *NN* was properly trained.

4.2.2. Influence of Sampling Time on the Computational Cost

As referred before, the choice of sampling time T_s is crucial for the performance of the tuning. In a sequence of the tests performed before in *MATLAB/SIMULINK* and resumed in **Table 2**, now, it is important to evaluate the consequences of the computational effort due to an accurate T_s into the tuning performance.

Using the same methodology as before, crescent values of T_s were used in *FPGA* and preliminary results showed that $T_s \le 0.1\,\text{s}$ is compulsory to obtain stable systems. On the other hand, a more refined sampling requires a longer computational time, therefore, it is needed that square wave transition occurs later to ensure accurate tuning. In **Table 3** we show some values of T_s used and the corresponding values of T needed for an accurate fitness.

In the 4th column, we display the relative error between the fitness evaluated online and the true fitness numerically evaluated. In spite of the error decreasing when the T_s decrease, for all the stable examples, the relative error is negligible. It is important to note that shorter values of T_s requiring larger values of T to assure that online fitness are acceptable. In all the cases, the optimal
$PID_o = [0.9467 \quad 1.4306 \quad 0.2540]$ is reached.

Table 3. Square wave period needed for different values of sampling time.

$T_s(\text{ms})$	$T(\text{ms})$	Fitness on-line	Relative error (%)
100	202.82	2.1671	2.04
50	381.75	2.1429	0.90
25	812.67	2.1310	0.33
10	2009.67	2.1238	0.00

4.2.3. Time-Variant System

This methodology accommodates time-variant systems, this means that for the transfer function in use, $G(s)=\frac{e^{-\lambda s}}{s+a}$, two parameters can be changed, the pole a and time delay λ, therefore, the robustness of the tuning is exemplified by varying the pole around the test value: $a=1$.

Based on the results obtained in the previous section, we chose the less restrictive value of sampling time: $T_s=100$ ms. The optimal individual $PID_o=[0.9467 \quad 1.4306 \quad 0.2540]$ is constant for all the cases; however, if it is accurate for $a=1$, it is not so accurate when we move the pole away. This is shown by results in **Table 4**, where, for each location of the pole, we evaluate the relative error between the fitness obtained on-line and the one obtained through *NN*.

Looking at the examples from the previous table, we conclude that worse tunings appear for 1st and last row because the pole is far from the location for which the ***NN*** was trained; despite this, we can point out the neighborhood $[0.75 \quad 1.25]$ where the tuning could be considered robust.

To illustrate the distortion of the tuning, we present the response superimposed to the square wave input for the cases correspondent to the 1st and last row, **Figure 13**.

The overshoot in the case where $a=0.25$ and the undershoot for $a=2$ shows that the tuning is poor, moreover, the square wave period is not enough to extinct the transient regime. Thus, the fitness evaluated in both cases are wrong. On the other hand, for the pole $a=1.25$ the response is best damped (see **Figure 14**).

We can also observe (see **Figure 16**) that the half period of the square wave is greater than the settling time, so, the fitness evaluation online is correct. This behavior is contrary to what happens for $a=0.25$ and $a=2$, (see **Figure 15**).

4.2.4. THE Tuning without a Model

The need of a model when we deal with systems in real time was explained before, however, include it could be computationally hard and not always they are accurate. Therefore, testing the methodology without a model is important either theoretically field or practical field. Several tests were performed and the square wave response is depicted in the following figure for the most relevant time windows.

Table 4. Relative error for a time-variant plant.

a	Fitness on-line	NN fitness	Relative error (%)
0.25	16.8660	13.2697	27.10
0.50	5.5468	5.7089	2.84
0.75	2.9083	2.9083	0.00
1.00	2.1671	2.1671	0.00
1.25	3.1753	3.2800	3.19
1.50	4.6706	5.0289	7.12
1.75	6.1856	7.0028	11.67
2.00	7.6669	9.1826	16.51

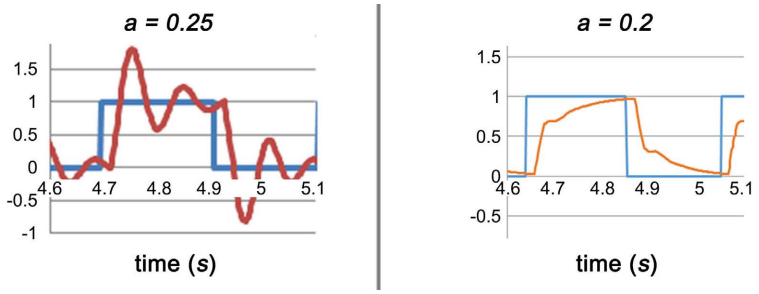

Figure 13. Square wave reference $r[k]$ superimposed to the corresponding output $y[k]$ for distinct pole locations.

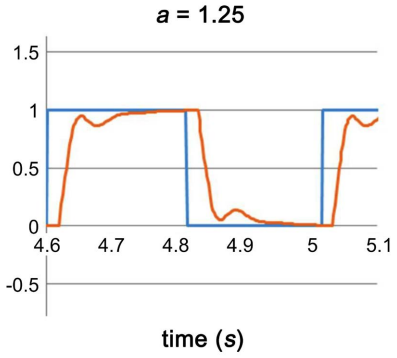

Figure 14. Square wave reference $r[k]$ superimposed to the corresponding output $y[k]$ for pole $a = 1.25$.

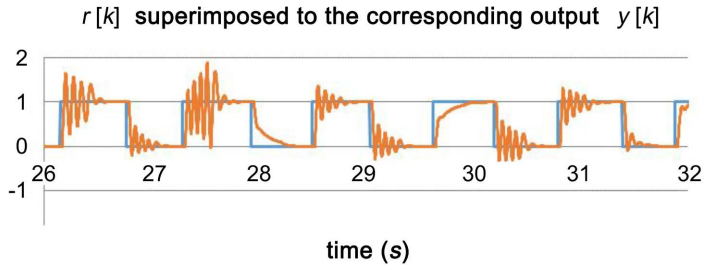

Figure 15. Square wave response when the tuning is made without a model, $t \in [26 \quad 32]$.

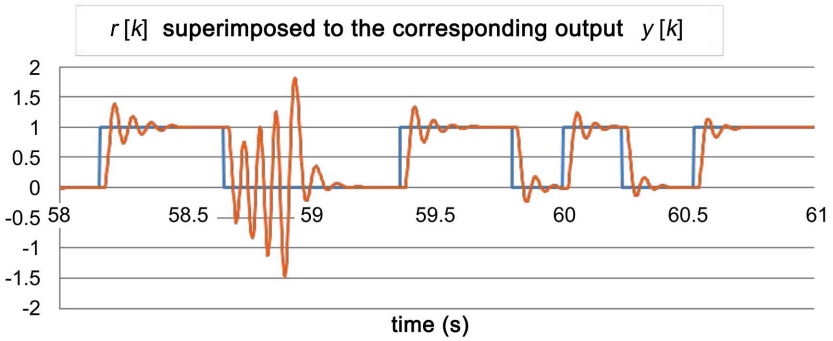

Figure 16. Square wave response when the tuning is made with a model.

The system was initialized with *PID* parameters randomly chosen around the Ziegler and Nichols values; this procedure assures us of the stability of the control loop. Next, the genetic algorithm takes place and each individual generated tunes the controller. This procedure presents as a consequence of getting unstable solutions. To overcome this undesirable behavior, a security factor was implemented as follows. The best individual is always memorized and if the security factor detects an unstable behavior, the best individual is recalled and the stability is replaced. This situation is illustrated in **Figure 15**.

The observation of this figure shows that an unstable individual appears in the interval $\left[27 \quad 28\right]$ s. However, due to the security factor, a stable situation is reached immediately and the system will remain stable for the subsequences individuals.

The latter time window is also presented in **Figure 16**. Since the genetic algorithm is a random process, it is expected that unstable individuals appear anywhere, in this case, it happens in the interval $\left[58 \quad 59\right]$ where once again the security factor avoids great damage.

The observation of this example shows us that tuning performance was globally improved when we compare the behaver for $t \in \left[26 \quad 32\right]$ with $t \in \left[58 \quad 61\right]$.

5. Concluding Remarks

This work presents a *PID* implementation in digital software and an optimized tuning assisted with *MATLAB/SIMULINK* simulator.

Regarding the real time implementation, the inclusion of a plant model inside the tuning system could be very useful, so, a *NN* trained off-line for the continuous plant was used and the results demonstrate its usefulness.

Before the hardware implementation, it is important to understand the behavior of the equivalent discrete system when the *PID* controller was previously tuned for the correspondent continuous. The *MATLAB/SIMULINK* implementation shows that the distortion noted is small when the equivalent discrete is tuned by the parameters optimized for the continuous one. Thus, it is proof that the approach according to *NN* trained off-line in continuous time will be used

for tuning the controller online is feasible.

The sampling time is an important parameter either for the stability of the discrete equivalent system or the computational effort in a real-time implementation. Therefore, the *MATLAB/SIMULINK* implementation is prepared to test several sampling time values and select the more convenient ones. These preliminary tests were performed and the value obtained was considered for the next step, *FPGA* implementation.

The *FPGA* implementation accommodates time-variant plants, so, for big changes in the plant, instability could appear. A security factor was implemented to prevent plant damage to overcome this undesirable behavior.

Tests running in FPGA can confirm the ones performed in *MATLAB/SIMULINK* simulator and the extent of issues related to real-time online tuning.

For now, the control criteria implemented are *ITAE*, *ITAY,* and the weighted sum of both, however, any control criterion could be accommodated according to our methodology.

The time between square wave transitions is crucial for the accuracy of control criteria evaluated online, so, we can conclude this feature influences the robustness of the tuning when the goal is dealing with time-variant plants. Therefore, an accurate online evaluation is compulsory either to a robust optimization or an adaption of the plant model.

After testing the sampling time in terms of stability using *MATLAB/SIMULINK* simulator, it is important conclusions about how it influences the computational cost inside *FPGA* board. Therefore, we conclude that a shorter sampling time enlarges the computation time, limiting the rate of transitions of the square wave. In fact, it is undesirable that a disturbance occurs before the parameters evaluated from the previous one tune the controller. The balance between the square wave period and the system's settling time is considered in our methodology.

Our tests for time-variant plants accommodate changes in the pole location, so, we can conclude about the dimension of the neighborhood centered into the pole where the tuning remains feasible.

The usefulness of a model is obvious; however, the powerfulness of the genetic algorithms used demonstrates that automatic tuning without a model could be possible under certain conditions.

Our current research explores the power of the tools developed in both environments, *MATLAB/SIMULINK* simulator, and *FPGA* board.

Now, knowing that our methodology accommodates time-variant plants, it is important to make a survey about the most representative transfer functions of industrial plants and construct a test database with them. After, a few models should be trained and evaluated for their validity over the transfer function database.

Our methodology accommodates several control criteria, so the tuning will be prepared for optimization of other criteria like overshoot, settling time, or rise

time; the methodology can accommodate multi-objective optimization.

Acknowledgements

This project was supported by "Programa Ensinar com Pesquisa" of the State University of São Paulo, which provided a scholarship for the first author to execute the project "Estudo e Desenvolvimento de Hardware Evolutivo em FPGA".

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Åström, K. and Hägglund, T. (2006) Advanced PID Control. ISA [Instrumentation, Systems, and Automation Society], Research Triangle.

[2] Vilanova, R. and Visioli, A. (2012) PID Control in the Third Millennium: Lessons Learned and New Approaches. Springer, London. https://doi.org/10.1007/978-1-4471-2425-2

[3] Li, Y., Ang, K.H. and Chong, G. (2006) Patents, Software, and Hardware for PID Control: An Overview and Analysis of the Current Art. *IEEE Control Systems Magazine*, **26**, 42-54. https://doi.org/10.1109/MCS.2006.1580153

[4] Ziegler, J.G. and Nichols, N.B. (1942) Optimum Settings for Automatic Controllers. *Transactions of the ASME*, **64**, 759-768.

[5] Kocijan, J. (2008) Survey of the Methods Used in Patents on Auto-Tuning Controllers. *Recent Patents in Electrical Engineering*, **1**, 201-208. https://doi.org/10.2174/1874476110801030201

[6] Li, Y., Ang, K.H. and Chong, G. (2006) PID Control System Analysis and Design. *IEEE Control Systems Magazine*, **26**, 32-41. https://doi.org/10.1109/MCS.2006.1580152

[7] Campi, M.C., Lecchini, A. and Savaresi, S.M. (2002) Virtual Reference Feedback Tuning: A Direct Method for the Design of Feedback Controllers. *Automatica*, **38**, 1337-1346. https://doi.org/10.1016/S0005-1098(02)00032-8

[8] Campi, M.C., Lecchini, A. and Savaresi, S.M. (2003) An Application of the Virtual Reference Feedback Tuning Method to a Benchmark Problem. *European Journal of Control*, **9**, 66-76. https://doi.org/10.3166/ejc.9.66-76

[9] Lecchini, A., Campi, M.C. and Savaresi, S.M. (2002) Virtual Reference Feedback Tuning for Two Degree of Freedom Controllers. *International Journal of Adaptive Control and Signal Processing*, **16**, 355-371. https://doi.org/10.1002/acs.711

[10] van Heusden, K., Karimi, A. and Bonvin, D. (2011) Data-Driven Model Reference Control with Asymptotically Guaranteed Stability. *International Journal of Adaptive Control and Signal Processing*, **25**, 331-351. https://doi.org/10.1002/acs.1212

[11] Lima, J. (2004) Sintonia automática de controladores PID: Uma abordagem neuro-genética. PhD Thesis, The University of Algarve, Faro.

[12] Hasanien, H.M. (2013) Design Optimization of PID Controller in Automatic Voltage Regulator System Using Taguchi Combined Genetic Algorithm Method. *IEEE Systems Journal*, **7**, 825-831. https://doi.org/10.1109/JSYST.2012.2219912

[13] Toledo, C., Lima, J. and Arantes, M. (2012) A Multi-Population Genetic Algorithm Approach for PID Controller Auto Tuning. 17*th IEEE International Conference on Emerging Technologies & Factory Automation* (ETFA' 2012), Kraków, 17-21 September 2012, 1-8. https://doi.org/10.1109/ETFA.2012.6489620

[14] Eiben, A.E. and Smith, J.E. (2003) Introduction to Evolutionary Computing. Springer Verlag, Heidelberg. https://doi.org/10.1007/978-3-662-05094-1

[15] Buriol, L., França, P.M. and Moscato, P. (2004) A New Memetic Algorithm for the Asymmetric Traveling Salesman Problem. *Journal of Heuristics*, **10**, 483-506. https://doi.org/10.1023/B:HEUR.0000045321.59202.52

[16] Moscato, P., Mendes, A. and Berretta, R. (2007) Benchmarking a Memetic Algorithm for Ordering Microarray Data. *Biosystems*, **88**, 56-75. https://doi.org/10.1016/j.biosystems.2006.04.005

[17] Toledo, C.F.M., França, P.M., Kimms, A. and Morabito, R. (2009) A Multi-Population Genetic Algorithm Approach to Solve the Synchronized and Integrated Two-Level Lot Sizing and Scheduling Problem. *International Journal of Production Research*, **47**, 3097-3119. https://doi.org/10.1080/00207540701675833

[18] Toledo, C.F.M., Arantes, M.S., Oliveira, R.R.R. and Almada-Lobo, B. (2013) Glass Container Production Scheduling through Hybrid Multi-Population Based Evolutionary Algorithm. *Applied Soft Computing*, **13**, 1352-1364. https://doi.org/10.1016/j.asoc.2012.03.074

[19] Khamlich, S., Khamlich, F., Atouf, I. and Benrabh, M. (2021) Performance Evaluation and Implementations of MFCC, SVM and MLP Algorithms in the FPGA Board. *International Journal of Electrical and Computer Engineering Systems*, **12**, 139-153. https://doi.org/10.32985/ijeces.12.3.3

[20] Powell, A., Savvas-Bouganis, C. and Cheung, P.Y.K. (2013) High-Level Power and Performance Estimation of FPGA-Based Soft Processors and Its Application to Design Space Exploration. *Journal of Systems Architecture*, **59**, 1144-1156. https://doi.org/10.1016/j.sysarc.2013.08.003

[21] Formentin, S., Campi, M.C. Savaresi, S.M. (2014) Virtual Reference Feedback Tuning for Industrial PID Controllers. *IFAC Proceedings Volumes*, **47**, 11276-11280. https://doi.org/10.3182/20140824-6-ZA-1003.01260

[22] Zhao, C.N., Xue, D.Y. and Chen, Y.Q. (2005) A Fractional Order PID Tuning Algorithm for a Class of Fractional Order Plants. *Proceedings of the* 2005 *IEEE International Conference Mechatronics and Automation*, Vol. 1, Niagara Falls, 29 July-1 August 2005, 216-221. https://doi.org/10.1109/ICMA.2005.1626550

[23] Manabe, S. (2002) A Suggestion of Fractional-Order Controller for Flexible Spacecraft Attitude Control. *Nonlinear Dynamics*, **29**, 251-268 https://doi.org/10.1023/A:1016566017098

[24] Torvik, P.J. and Bagley, R.L. (1984) On the Appearance of the Fractional Derivative in the Behavior of Real Materials. *Journal of Applied Mechanics*, **51**, 294-298. https://doi.org/10.1115/1.3167615

[25] Nakagawa, M. and Sorimachi, K. (1992) Basic Characteristics of a Fractance Device. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, **E75-A**, 1814-1819.

[26] Nie, Z.Y., Wang, Q.G., Liu, R.J. and Lan, Y.H. (2016) Identification and PID Control for a Class of Delay Fractional-Order Systems. *IEEE/CAA Journal of Automatica Sinica*, **3**, 463-476. https://doi.org/10.1109/JAS.2016.7510103

[27] Yin, Z.B., Qian, H., Xiao, A.P., Wu, J. and Liu, G. (2011) The Application of Adap-

tive PID Control in the Spray Robot. 2011 4*th International Conference on Intelligent Computation Technology and Automation*, Shenzhen, 28-29 March 2011, 528-531. https://doi.org/10.1109/ICICTA.2011.145

[28] Caleb, J. and Kannan, M. (2017) Efficient VLSI Implementation of the C-MANTEC Conn Algorithm by Using PID Controllers. *Circuits and Systems*, **8**, 253-260. https://doi.org/10.4236/cs.2017.811018

[29] Lima, J. and Ruano, A. (2000) Neuro-Genetic PID Autotuning: Time Invariant Case. *IMACS Journal of Mathematics and Computers in Simulation*, **51**, 287-300.

[30] Franklin, G., Powell, J. and Workman, M. (1998) Digital Control of Dynamic Systems. Addison-Wesley, Boston.

[31] Ogata, K. (1995) Discrete-Time Control Systems. Prentice-Hall International, Inc., Hoboken.

[32] Intel® FPGAs and Programmable Devices-Intel® FPGA (2022). https://www.intel.com/content/www/us/en/products/programmable.html