Synthesis and Tuning of Non-Conventional Structures for FOPID Controllers Using Genetic Programming

Humberto Ícaro Pinto Fontinele
Institute of Engineering and Sustainable Development – University of International Integration of Afro-Brazilian Lusophony
Fortaleza, Brazil
icarofontienele@unilab.edu.br

Guilherme de Alencar Barreto
Department of Teleinformatics Engineering – Technology Center - Federal University of Ceará
Fortaleza, Brazil
gbarreto@ufc.br

David Ciarlini Chagas Freitas
Federal Institute of Education, Science and Technology of Ceará (IFCE) –
Fortaleza, Brazil
david.ciarlini@ifce.edu.br

***Abstract* — This paper presents a method for self-configuration and optimal tuning of parameters for PID controllers, including fractional-order PID controllers (FOPID), based on the evolutionary mechanism of Genetic Programming (GP). This method enables the synthesis of non-conventional structures and the optimization of the tuning of these controllers by automatically adjusting the gains and fractional exponents of the derivative and integral terms, based on time-domain performance indices. The results obtained demonstrate that the high flexibility in gain and phase margin adjustments, an inherent characteristic of controllers with high-order and fractional structures, when properly tuned, allows for effective performance in the control of complex systems, such as high-order plants and unstable systems. A comprehensive evaluation over various types of plants is carried out, with the proposed approach exhibiting overall superior performance compared to the two SOTA methods for all analyzed plants.**

***Keywords* — genetic programming, non-conventional PID, optimal tuning, self-configurable PID.**

I. INTRODUCTION

The field of control systems, particularly in controller synthesis, has gained prominence due to numerous theoretical and practical advances. These developments have been largely driven by the increasing demands of industry for improvements in production and manufacturing processes [19]. Various control strategies, such as adaptive, neural, and fuzzy control, have been widely studied and applied [1]–[3]. Although these techniques are effective, the PID controller remains the most widely used, primarily due to its simple structure and reliable performance across diverse systems and conditions. However, selecting the appropriate PID structure and performing optimal parameter tuning adjusting the gains and fractional exponents of the derivative and integral terms still remain a non-trivial challenge.

Classical tuning methods, including Ziegler–Nichols [16], Cohen–Coon [17], and Chien–Hrones–Reswick [18], along with cost-function-based techniques (IAE, ISE, ITAE, ITSE), are commonly applied to conventional PID tuning [25]. Yet, these methods exhibit limitations, particularly with high-order systems and non-conventional or fractional-order PID (FOPID) controllers [4]. FOPID controllers extend classical PID by incorporating fractional-order integral and differential operators, offering enhanced flexibility and fine-tuning capabilities, especially in systems with complex dynamics. However, this flexibility introduces additional challenges, as classical tuning methods do not directly apply to FOPIDs [20].

Fractional controllers tend to be less sensitive to parameter variations [5] and offer more degrees of freedom for adjusting gain and phase margins. For instance, studies such as [18] show FOPID controllers improve phase and gain margins compared to classical PID controllers with identical P, I, and D parameters. Tuning FOPID controllers remains a major challenge, prompting various heuristic approaches—e.g., modified Ziegler–Nichols [21], neural networks [22], fuzzy and neuro-fuzzy systems [23], and evolutionary algorithms [6], [24].

Conventional PID structures include series, parallel, or ISA formats. Any deviation from these is generally considered non-conventional. Classical methods are ineffective for tuning such structures or FOPIDs [4], which typically demand dedicated design and optimization strategies. Recent studies suggest that Genetic Programming (GP) can be leveraged to synthesize and optimize PID and FOPID controllers with non-conventional structures, especially for high-order or unstable systems [7][8][9]. However, despite its potential, the use of GP in the synthesis and parameter tuning of FOPID controllers has not been explored to its full extent, unlike other metaheuristics such as genetic algorithms (GA) or particle-swarm optimization (PSO).

In this regard, the present work aims at contributing to expand the design and synthesis of FOPID controllers by proposing an automated GP-based framework, termed GP-FOPID, for synthesizing high-order FOPID controllers. A comparative performance analysis with traditional control methods is also conducted across systems of varying complexity. The main contributions are as follows:

i. To introduce an alternative to classical methods, capable of tuning PID and FOPID controllers of complex structure.

ii. To introduce a novel metaheuristic-based approach for synthesizing multilayer FOPID controllers aimed at high-order systems.

iii. To lay down the foundation for developing novel heuristic approaches to non-conventional controller design.

Subsequent sections present a theoretical overview of fractional calculus, PID architectures, and genetic programming, followed by the proposed method and results. The paper concludes by discussing the advantages of GP-FOPID compared to existing synthesis and tuning methods.

II. THEORETICAL BACKGROUND

A. On Fractional Calculus

Fractional calculus has been largely applied in the control field, offering an alternative approach to standard integer-order control methods by allowing derivatives and integrals to be defined with arbitrary fractional orders. In this context, the fractional-order differentiator is represented by a generic operator, ${}_aD_t^q$, which extends the traditional concepts of differentiation and integration. of this operator is presented. The formal definition of this operator is presented as:

$$ {}_aD_t^q = \begin{cases} \dfrac{d^q}{dt^q}, & \text{real(q)>0} \\ 1, & \text{real(q)=0} \\ \displaystyle\int_a^t (d\tau)^{-q}, & \text{real(q)<0} \end{cases} \quad (1)$$

associated with the system's initial which serves as the foundation for modeling and designing control systems with complex dynamics. The exponent q denotes the fractional order, which can take real or complex values; a is a a constant associated with the system's initial conditions; and t is the time variable related to the current or final instant of the analysis [21].

Two widely used definitions for the formulation of fractional-order derivatives and integrals are the Grünwald–Letnikov (GL) definition and the Riemann–Liouville (RL) definition [11]. Both offer distinct approaches to generalize the classical concepts of calculus, being employed according to the specific characteristics and requirements of the system under analysis.

The GL definition is given as follows:

$$ {}_aD_t^q f(t) = \frac{d^q f(t)}{d(t-a)^q} = \lim_{N\to\infty} \left[\frac{t-a}{N}\right]^{-q} \sum_{j=0}^{N-1} (-1)^j \binom{q}{j} f\left(t-j\left[\frac{t-a}{N}\right]\right) \quad (2)$$

where $\left[\frac{t-a}{N}\right]$ is an integer.

The RL definition is given as:

$$ {}_aD_t^q f(t) = \frac{d^q f(t)}{d(t-\alpha)^q} = \frac{1}{\Gamma(n-q)} \frac{d^n}{dt^n} \int_0^t (t-\tau)^{(n-q-1)} f(\tau) d\tau \quad (3)$$

where $(n-1 \le q < n)$, with n being an integer and q a real number, while $\Gamma(x)$ is the Euler Gamma function. Furthermore, there is another widely used definition for the fractional derivative, introduced by Caputo [13], whose formulation is as follows:

$$ {}_aD_t^q f(t) = \begin{cases} \dfrac{1}{\Gamma(m-q)} \displaystyle\int_a^t \frac{f^{(m)}(\tau)}{(t-\tau)^{q+1-m}} d\tau, & (m-1 \le q < n) \\ \dfrac{d^m}{dt^m} f(t), & q = m \end{cases} \quad (4)$$

B. On FOPID Controllers

The FOPID controller, also known as $PI^\lambda D^\mu$, is a generalization of the classical PID controller. It introduces two new parameters: λ (the order of the integral) and μ (the order of the derivative), which can take fractional values. This provides extra degrees of freedom for modeling system dynamics, allowing for better performance in terms of robustness to noise, uncertainties, and disturbances, as well as improved control of complex systems, especially those with slow or unstable dynamics.

The FOPID controller can be implemented either in a series structure or in a parallel structure; however, it is commonly represented in the parallel form, as illustrated in Fig. 1 [10]. This configuration explicitly highlights the proportional, integral, and derivative parameters, along with their respective fractional orders, allowing for a direct visualization of the control structure.

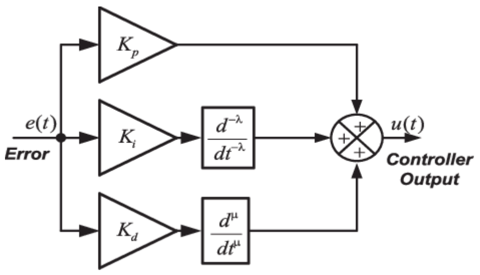
K_p
$e(t)$
Error
K_i
$\frac{d^{-\lambda}}{dt^{-\lambda}}$
K_d
$\frac{d^{\mu}}{dt^{\mu}}$
$u(t)$
Controller Output

Fig. 1 – Standard FOPID. Source: Charan, Nimai & Sahu et al. [10].

Fig. 2 presents the series and parallel structures of the FOPID, providing a representation of the differential equations in the frequency domain.

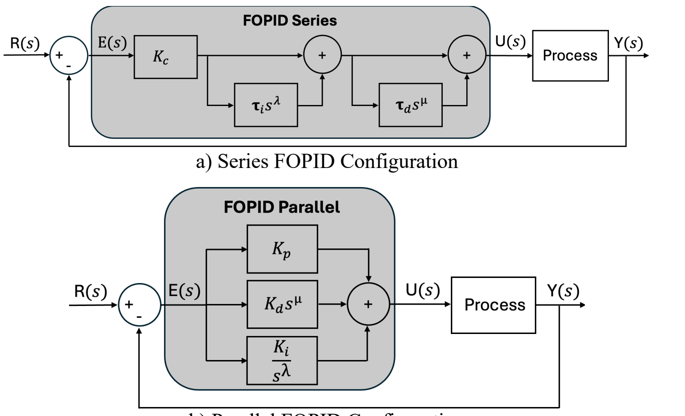

Fig. 2 – Series and Parallel FOPID Configurations. Source: author.

The corresponding transfer functions are expressed in equations (5) and (6).

$$C(s) = K_c \left(\frac{1+\tau_i s^\lambda}{\tau_i s^\lambda}\right)(\tau_d s^\mu + 1) \quad \text{(series PID)(5)}$$

$$C(s) = K_p + \frac{K_i}{s^\lambda} + K_d s^\mu \quad \text{(parallel PID)(6)}$$

where K_p represents the proportional gain, K_d the derivative gain, K_i the integral gain, τ_i time constant of the integral term and τ_d time constant of the derivative term. The variables of the parallel model are related to those of the series model as follows:

$$K_p = K_c\left(1+\frac{\tau_d}{\tau_i}\right) \quad (7)$$

$$K_i = \frac{K_c}{\tau_i} \quad (8)$$

$$K_d = K_c \tau_d \quad (9)$$

Furthermore, FOPID controllers tend to offer larger margins for gain and phase adjustments, allowing stable operation over wider frequency and gain ranges, which facilitates meeting more stringent control criteria. However, tuning these controllers is more complex compared to the classic PID controller, due to the two additional degrees of freedom provided by the derivative and integral fractional orders. This complexity makes the direct application of traditional tuning methods unfeasible [21].

C. *On Genetic Programming*

Genetic Programming (GP) is an evolutionary algorithm in which each individual in the population represents a computer program, whose hierarchical structures can vary in size during the evolutionary process. This flexibility in structure size is the main feature that distinguishes GP from alternative evolutionary algorithms, such as the GA. This variation in size generally causes GP to require more iterations to reveal trends in the results. Furthermore, the freedom to adjust the structure size often requires the implementation of mechanisms that limit the maximum size of individuals [13].

Another major difference between GP and GA is the absence of explicit prior encoding of the problem solution in the former, since each individual in the population is a computer program whose solution, or part of it, is obtained by executing the program. In addition, GP uses individuals with hierarchically structured genetic material, usually in the form of a syntactic tree structure, and not just linearly ordered as in GA chromosomes. [13].

D. *Representation of Individuals in Syntactic Trees*

A syntactic tree consists of symbols that represent functions $F = \{f_1, f_2, \ldots f_n\}$ used to label the internal nodes of the tree, and terminals $\tau = \{t_1, t_2, \ldots t_n\}$, used to label the leaves of the trees. The trees can assume various structures through the recursive combination of functions and terminals.

The functions can represent arithmetic operations, boolean operations, conditional operations, iterative operations, or functions for specific problems, while each terminal is usually a variable or a constant. In the context of this research, for example, each node of the tree can represent a FOPID structure (series or parallel, complete or incomplete), and each terminal can represent a series or parallel interaction between the different FOPID structures present in the nodes.

E. *Closure and Sufficiency Properties*

The sets of functions and terminals must satisfy two essential properties: closure and sufficiency. The closure property requires that each function in the function set must be capable of accepting as an argument any value generated by possible combinations of other arguments. This property is fundamental because it ensures that the processing of any program generated by the tree will produce a result that can be translated into a fitness value, according to the cost function defined for the problem. Failure to meet this property would compromise the execution of the programs, resulting in the failure of the GP system or the generation of unpredictable results [13].

An example of a tree that does not meet the closure property would be one that includes the division operator in the function set and allows zero to be the second terminal in a pair of terminals, which would result in an invalid operation.

Ensuring the closure property in real-world applications is not always a simple task, especially when the problem domain requires the use of different data types. A typical example is the combination of Boolean functions with numerical functions. To handle this complexity, it is possible to use typed functions and genetic elements compatible with the types of operations defined in the GP system. An effective approach in this context is *strongly typed GP*, in which each primitive carries information about its type and the types of arguments it can accept. This forces the construction of syntactic trees with semantically valid combinations, promoting type consistency throughout the evolutionary process. Additionally, strong typing reduces the search space, which often contributes to increasing the efficiency and chances of success of the evolutionary search [14].

The sufficiency property requires that the set of functions and terminals must be capable of theoretically representing a viable solution within the problem domain [14]. In many cases, especially in complex domains, it is not possible to guarantee this property with certainty, requiring prior knowledge and designer experience to define appropriate sets of operators and terminals.

In the method proposed in this work, the guarantee of the closure and sufficiency properties is achieved by adopting, as nodes of the syntactic tree, the series concatenation functions (represented by the symbol '+') and parallel concatenation (represented by '/'). Each terminal of the tree receives a value associated with a specific processing unit, P, I, D, I^{λ} ou D^{μ}. The gain values and fractional orders are randomly generated during the population initialization, respecting predefined limits for both the values and the depth of the tree, as detailed in the next section.

For the parameter that do not receive assigned values during the initialization phase, a value of zero is assigned, ensuring the validity of the closure property and avoiding failures in the execution of the generated individuals.

Table 1 presents the grammar defined for the construction of the syntactic trees used in the proposed method. This grammar establishes the formal rules that guide the composition of nodes and terminals, ensuring the generation of valid individuals throughout the evolutionary process.

TABLE 1 – SYMBOLS OF THE SYNTAX TREE

Type	**Simbol**	**Representation**
Layer	(	Start of layer
	)	End of layer
Node	+	Series interaction function
	/	Parallel interaction function
Terminal	P	Proportional parameter
	I	Integral parameter
	D	Derivative parameter
	I^{λ}	Fractional integrator parameter
	D^{μ}	Fractional derivative parameter

For illustration purposes, Fig. 3 shows the structure of a controller represented by a three-layer syntactic tree, corresponding to the expression: (/PI (+(/(/PD) (/ID)) PI^{λ})). This representation highlights the hierarchical organization and the combination between series and parallel concatenation operators, reflecting the modular composition of the controller's functional blocks.

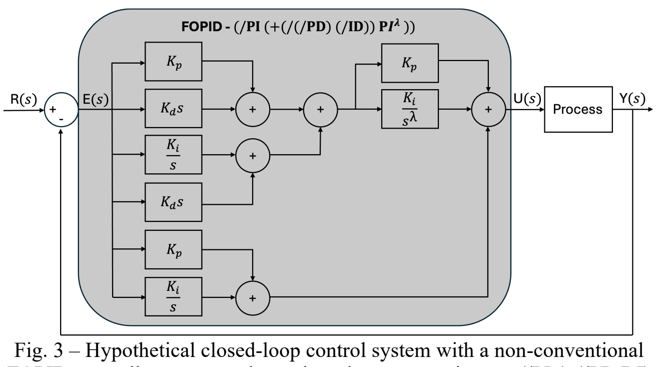

Fig. 3 – Hypothetical closed-loop control system with a non-conventional FOPID controller, structured as a three-layer syntactic tree: (/PI (+(/PD DI) IP^lambda). Source: Author

F. *Population Initialization*

Before the first evolutionary round of GP, the population must be initialized by generating the initial structures of the programs to be evolved. In tree-based GP, the most common initialization methods are: grow, full, and ramped half-and-half [13]. These methods are governed by a key parameter: the maximum tree depth, denoted as d.

In the grow method, each tree is built recursively from a root symbol randomly selected from the function set F. For each function of arity n, n children are assigned, chosen randomly from the union $F \cup \tau$, where τ is the set of terminals. This process continues until the tree reaches depth d. At depth $d-1$, only terminals are selected, ensuring the depth constraint.

Although the grow method guarantees the tree does not exceed d levels, it produces trees with highly variable shapes and sizes, often asymmetric with uneven branches. The resulting structure depends on the ratio of functions to terminals:

- With many terminals, trees tend to be smaller and shallower.
- With more multi-argument functions, branches often reach the depth limit, producing denser, more complete trees.

The full method, in contrast, uses only functions from *F* until depth *d* is reached, after which terminals fill the leaf nodes. This results in trees with uniform, symmetrical structures where all branches reach the maximum depth.

However, relying solely on grow or full methods can limit structural diversity in the initial population. To address this, Koza and Poli [12] proposed the ramped half-and-half method. It divides the population into groups with depths ranging from 1 to d. At each depth, half the trees are generated using full and the other half using grow, yielding a population with diverse sizes and shapes, from balanced to highly asymmetric, and promoting genetic diversity early in evolution.

The subsequent steps of GP — such as selection, crossover, and mutation — follow principles similar to GA. These will not be detailed here, as they are extensively covered in the literature [7].

III. THE PROPOSED METHOD

This work proposes a method to automate the synthesis of FOPID controllers, generating unconventional structures through a GP mechanism based on syntactic trees, named henceforth Genetic Programing FOPID (GP-FOPID).

In each node of the tree, an FOPID structure is allocated in the parallel configuration (as illustrated in Fig. 1), which may be either complete or incomplete. The arcs connecting the nodes represent serial or parallel interactions between these parallel FOPID structures.

The performance evaluation of the generated controllers is performed in the time domain, and the fitness of the individuals i is measured based on a normalized cost function, as defined next as.

$$f_R(i)=\min\left(\left(\frac{\alpha_1}{100*ss+0.00001}\right),\ 200\right)+ \min\left(\left(\frac{\alpha_2}{100*abs(ee)+0.00001}\right),200\right)+ \min\left(\left(\frac{\alpha_3}{ts+0.0001}\right),200\right)+\min\left(\left(\frac{\alpha_4}{ta+0.0001}\right),200\right)+ \min\left(\left(\frac{\alpha_5}{itae+0.0001}\right),200\right) \quad (10)$$

in which the terms of the cost function represent the following performance indicators in the time domain: ss is the overshoot, ee is the steady-state error, ts is the rise time, ta is the settling time (considering the 5% range), and $itae$ is the integral of time-weighted absolute error. The coefficients α_1, α_2, α_3, α_4, and α_5 allow adjusting the relative weights of each term, according to the desired performance requirements for the application.

This method aims to maximize the cost function, which is defined as the sum of five terms, each associated with a performance indicator in the temporal domain. To mitigate potential distortions in the overall metric caused by indicators with values close to zero, an upper limit of 200 points is imposed on each term. This restriction prevents disproportionately high contributions from such indicators, promoting a more balanced representation among the terms of the function. Thus, the theoretical maximum value that the cost function can attain is 1000 points.

A small constant value (0.0001) is added to the denominator of each term to avoid division by zero, since some indicators, such as the steady-state error (ee), can assume a null value. The selection of the weighting factors should be made based on the specific control priorities of each system being analyzed.

After preliminary experiments, the values of the weighting factors presented in Table 2 were the ones that allowed achieving good results for controlling most of the evaluated plants.

TABLE 2 – WEIGHTING FACTORS OF THE COST FUNCTION

Weight Factors	α_1	α_2	α_3	α_4	α_5
Value	2	4	1	2	5

The initial population is generated using the *grow* method and consists of 20 individuals, each represented by a chromosome of 50 characters with decimal encoding. Of these, 25 characters are allocated to represent the parameters for the parallel structure of the controller, while the remaining 25 correspond to the series structure. Each controller parameters (K_p, K_i, K_d, λ and μ) is encoded using five characters: the first two represent the integer part of the value, and the following three correspond to the fractional part. The initial values assigned to the parameters are randomly generated following a normal distribution, subject to the following upper limits: 10 for the proportional parameter, 5 for the integral and derivative parameters, and 1 for the fractional exponents λ and μ, respectively. For uninitialized nodes, a value of zero is assigned to the proportional, integral, and derivative parameters, and a value of one is assigned to the fractional-order exponents. This encoding enables the synthesis of FOPID-type controllers, which may be composed of complete or partial blocks, organized in series or parallel structural configurations.

To avoid the generation of improper controllers, a restriction was added: whenever a tree represents an invalid controller, it is automatically replaced by a tree equivalent to a unit proportional controller ($P = 1$), thus ensuring the stability of the evolutionary process.

The tree depth is limited to 3 layers to avoid synthesizing controllers with excessively high-order transfer functions, which may lead to unnecessary computational costs in real-word applications. The parameters K_p, K_i, K_d, λ and μ are also constrained to specific positive intervals: K_p ranges from 0 to 20, K_i and K_d from 0 to 10, and λ and μ from 0 to 2. These limits prevent the generation of controllers with overly large values, which could cause instability or saturation in real systems, especially under noise. As each tree node represents a parallel FOPID structure with μ limited to 2, it can model up to a second-order system. Therefore, a full tree can encode a controller of up to eighth order. A crossover probability of 70% and a mutation rate of 30% are adopted. The generational distance is 90%, which implies that the 10% fittest individuals are kept and replicated in the next generation. The selection method used was tournament selection, with one-point crossover and a stopping criterion set at 25 generations. It was observed that exceeding this number of generations does not justify the increase in computational cost, as it does not significantly improve the performance of the synthesized controllers.

To evaluate performance, the proposed method's synthesized controllers were applied to control plants of second to fourth order using a unit step input. Simulations ran in R2024b Matlab (MathWorks®) on a Mac M1, 8GB RAM.

A comparative study was conducted with classic PID controllers, tuned by the first Ziegler-Nichols method and a computational optimization approach based on closed-loop stability and robustness, considering setpoint tracking and disturbance rejection in the frequency domain. Robustness was assessed via gain and phase margins, using Matlab's PID autotuner, which automatically adjusts PID parameters to balance performance and robustness. The autotuner works by performing a frequency-response estimation experiment, injecting test signals into the plant and tuning PID parameters based on the estimated response. The results are presented in the next section.

VI. RESULTS AND DISCUSSIONS

This section presents the results from performance comparison tests between the proposed method and the two reference methods mentioned earlier. Performance was evaluated on several LTI systems, with their transfer functions listed in Table 3.

TABLE 3 – TRANSFER FUNCTIONS OF THE SIMULATED PLANTS USED IN THE EXPERIMENTS

Plant	Transfer Functions
2nd order – overdamped	$\frac{1}{s^2 + 4s + 1}$
2nd order – critically damped	$\frac{1}{s^2 + 2s + 1}$
2nd order – underdamped	$\frac{1}{s^2 + s + 1}$
2nd order non-minimum phase	$\frac{-s + 3}{s^2 + 5s + 2}$
3rd order – 3 real roots	$\frac{1}{s^3 - 6s^2 + 11s - 6}$
3rd order – 1 real and 2 complex roots	$\frac{1}{s^3 + 6s^2 + 11s - 6}$
4th order – 2 real and 2 complex roots	$\frac{s - 1}{(s - 1)(s - 2)(s - 2 + 1i)(s - 2 - 1i)}$
4th order – 4 positive real roots	$\frac{s - 1}{(s - 1)(s - 2)(s - 3)(s - 4)}$
4th order – 4 complex roots	$\frac{s - 1}{(s - 1 + i)(s - 2 + i)(s - 3 + 1i)(s - 4 + 1i)}$

After applying the three tuning and synthesis methods, the values obtained for the parameters K_p, K_i, K_d, λ and μ are presented in Table 4.

TABLE 4 – K_p, K_i, K_d, λ, AND μ PARAMETERS ADJUSTED BY THE EVALUATED METHODS

System	Method	K_p	K_i	K_d	λ	μ
2nd order – overdamped	ZN	17.327	0.530	0.1325	-	-
	PID Tune	1.885	0.833	0.496	-	-
	PG – 1st	0.000	0.000	0.445	1.000	1.000
	PG – 2nd	3.552	1.701	0.000	0.898	1.000
	PG – 3rd	0.000	0.000	0.000	1.000	1.000
2nd order – critically damped	ZN	7.792	0.728	0.182	-	-
	PID Tune	2.046	1.454	0.700	-	-
	PG – 1st	10.958	3.786	2.981	1.454	1.643
	PG – 2nd	0.513	3.723	2.238	1.199	0.352
	PG – 3rd	0.000	0.000	0.000	1.000	1.000
2nd order – underdamped	ZN	4.4040	0.8600	0.2150	-	-
	PID Tune	2.0951	1.1920	0.9206	-	-
	PG – 1st	0,000	0.000	0.000	1.000	1.000
	PG – 2nd	0,000	0,972	1.900	1.000	1.000
	PG – 3rd	0,000	0,000	0.000	1.000	1.000
2nd order non-minimum phase	ZN	3.0796	1.0681	0.2670	-	-
	PID Tune	1.2946	0.6292	0.000	-	-
	PG – 1st	0.000	0.000	0.034	1.000	1.000
	PG – 2nd	0.000	0.571	0.034	1.000	0.544
	PG – 3rd	0.000	0.000	0.000	1.000	1.000
3rd order – 3 real roots	ZN	Not applicable				
	PID Tune	0.4472	0.0051	9.8359	-	-
	PG – 1st	10.641	0.5920	4.8350	1.898	0.6830
	PG – 2nd	10.484	3.838	0.000	0.365	1.000
	PG – 3rd	0.000	0.000	3.596	1.000	1.864
3rd order – 1 real and 2 complex roots	ZN	3.860	0.8607	0.2152	-	-
	PID Tune	14.709	24.264	2.2293	-	-
	PG – 1st	10.011	1.1600	0.111	1	1
	PG – 2nd	10.011	1.1600	0.111	10.100	1
	PG – 3rd	1.001	1.611	1.001	10.101	10.100
4th order – 2 real and 2 complex roots	ZN	0.0760	18.800	4.7000	-	-
	PID Tune	0.2659	0.0029	6.1624	-	-
	PG – 1st	0.000	0.000	4.320	1.000	1.000
	PG – 2nd	0.000	0.575	4.320	0.292	0.274
	PG – 3rd	0.000	0.000	0.000	1.000	1.000
4th order – 4 positive real roots	ZN	0.0251	18.970	4.742	-	-
	PID Tune	0.4868	0.0058	10.272	-	-
	PG – 1st	0.000	0.000	1.607	1.000	0.757
	PG – 2nd	0.000	0.000	0.000	1.000	1.000
	PG – 3rd	0.000	0.000	0.000	1.000	1.000
4th order – 4 complex roots	ZN	Not applicable				
	PID Tune	-3.692	-0.027	-128,2	-	-
	PG – 1st	0.000	0.000	4.360	1.000	1.000
	PG – 2nd	6.810	4.609	5.382	1.000	1.000
	PG – 3rd	0.000	0.000	0.000	1.000	1.000

In this table, ZN refers to the Ziegler-Nichols method, PID Tune refers to Matlab's PID autotuner method and PG-1st, PG-2nd, and PG-3rd correspond to the parameters of the first, second, and third layers, respectively, of the non-conventional FOPID tuned using the method proposed in this work, called FOPID based on Genetic Programming (GP-FOPID).

For qualitative analysis, the figures 4 to 29 present the responses of the controllers tuned and synthesized by the three methods, in response to a unit step.

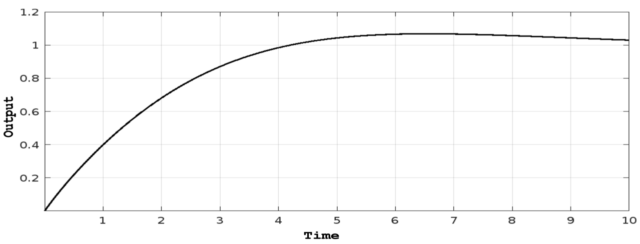

Fig. 4 - Step Response of a 2nd Order Overdamped Syst. (Matlab's PID Tuner)

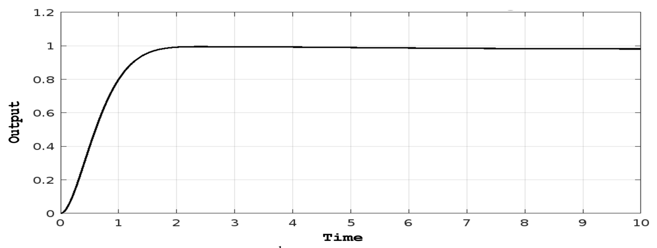

Fig. 5 - Step Response of a 2nd Order Overdamped System (GP-FOPID)

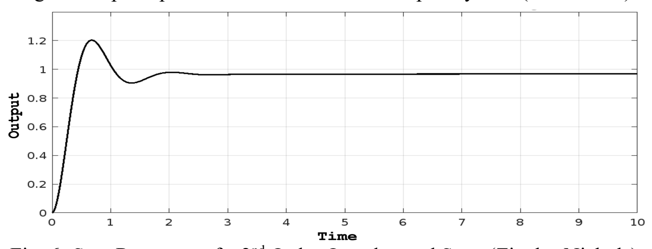

Fig. 6- Step Response of a 2nd Order Overdamped Syst. (Ziegler-Nichols)

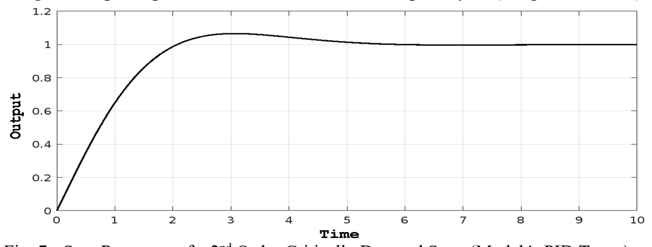

Fig. 7 - Step Response of a 2nd Order Critically Damped Syst. (Matlab's PID Tuner)

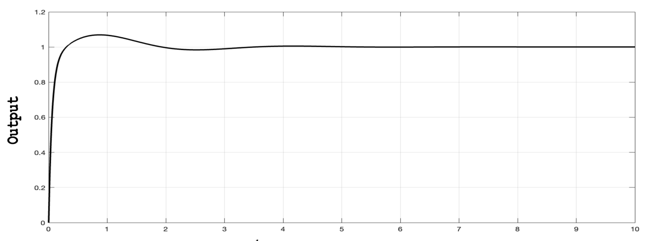

Fig. 8 - Step Response of a 2nd Order Critically Damped Syst. (GP-FOPID)

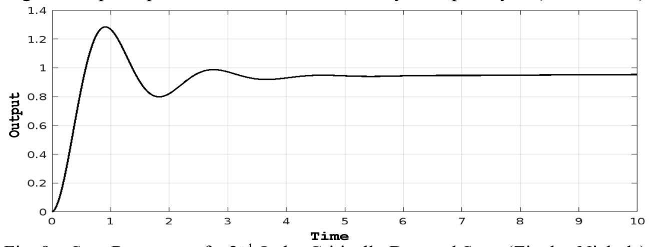

Fig. 9 – Step Response of a 2nd Order Critically Damped Syst. (Ziegler-Nichols)

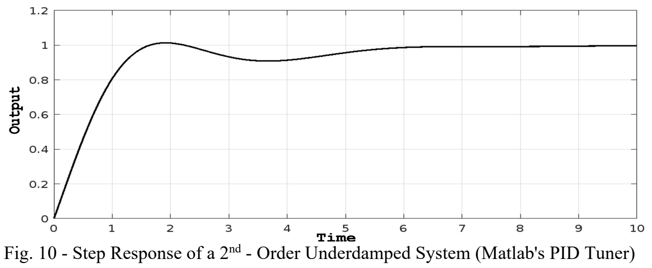

Fig. 10 - Step Response of a 2nd - Order Underdamped System (Matlab's PID Tuner)

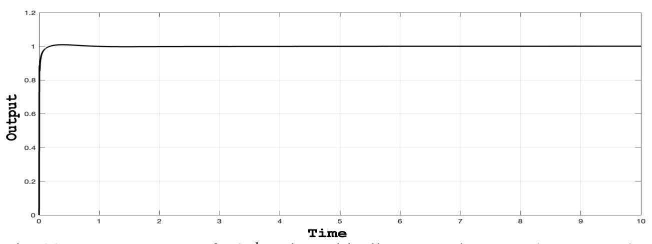

Fig. 11 - Step Response of a 2nd Order Critically Damped System (GP-FOPID)

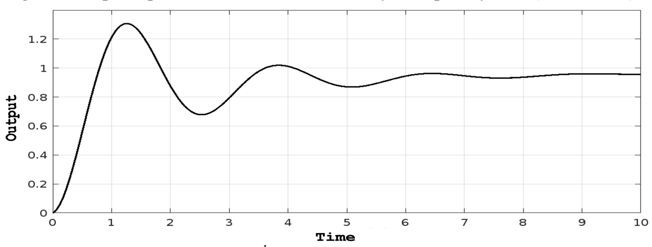

Fig. 12 - Step Response of a 2nd Order Underdamped System (Ziegler-Nichols)

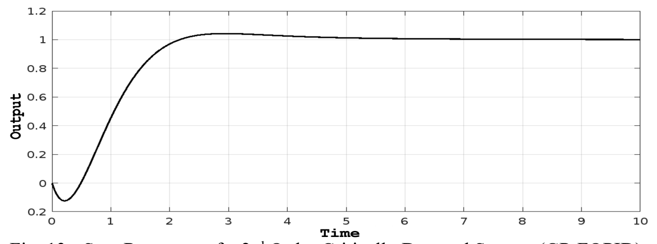

Fig. 13 - Step Response of a 2nd Order Critically Damped System (GP-FOPID)

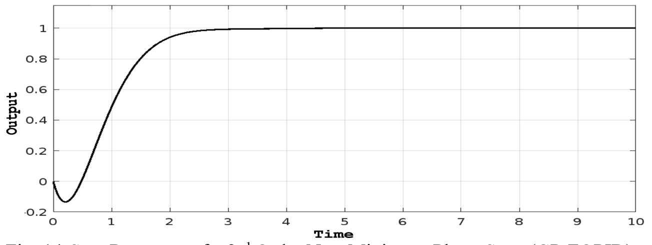

Fig. 14-Step Response of a 2nd Order Non-Minimum Phase Syst. (GP-FOPID)

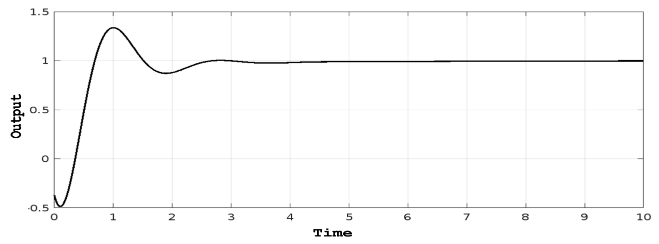

Fig. 15 - Step Response of a 2nd Order Non-Minimum Phase Syst. (Ziegler-Nichols)

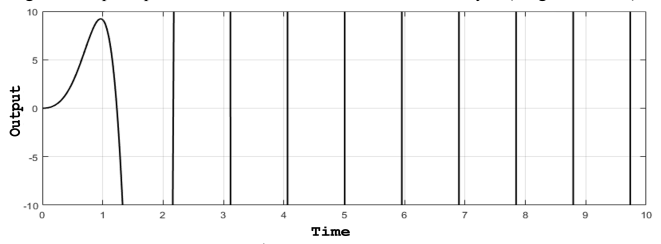

Fig. 16 – Step Response of a 3rd Order Syst. - 3 Real Roots (Matlab's PID Tuner)

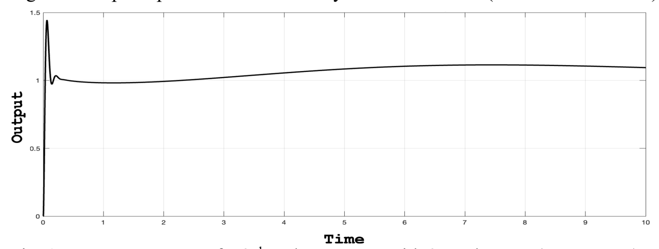

Fig. 17 - Step Response of a 3rd Order System With 3 Real Roots (GP-FOPID)

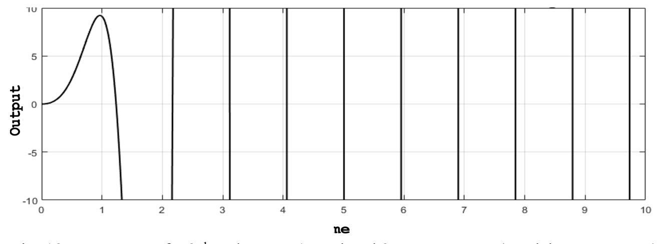

Fig. 18-Step Resp. of a 3rd Order Syst.1 Real and 2 Comp. Roots (Matlab's PID Tuner)

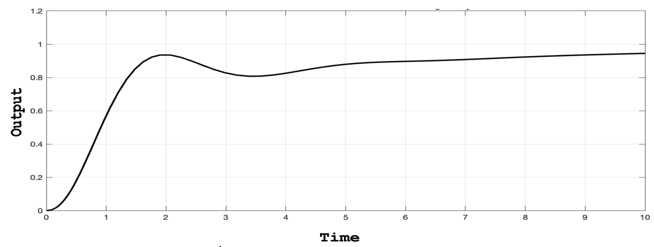

Fig. 19 – Step Resp. of a 3rd Order Syst. - 1 Real And 2 Complex Roots (GP-FOPID)

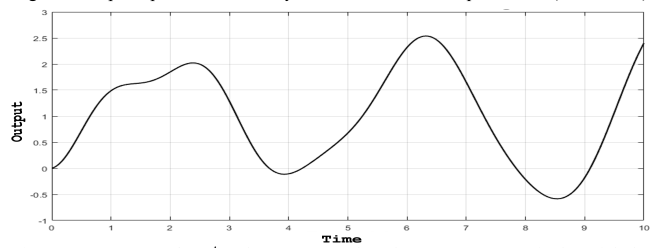

Fig. 20 - Step Resp. of a 3rd Order Syst.-1 Real And 2 Comp. Roots (Ziegler-Nichols)

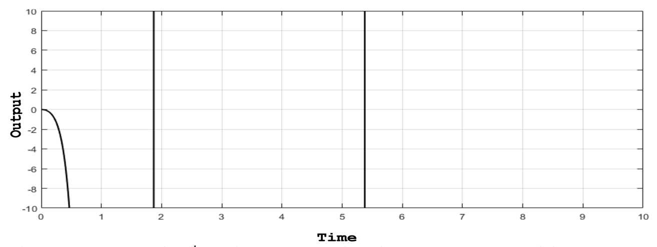
Fig. 21-Step Resp. of a 4th- Order Syst.-2 Real And 2 Comp. Roots (Matlab's PID Tuner)

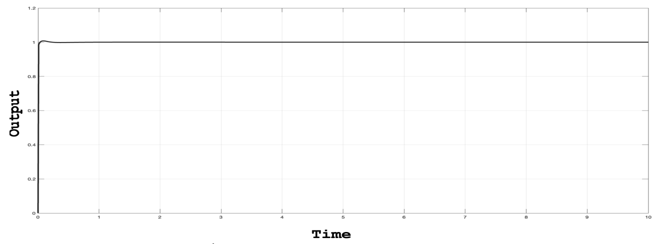
Fig. 22 - Step Resp. of a 4th - Order Syst. - 2 Real And 2 Complex Roots (GP-FOPID)

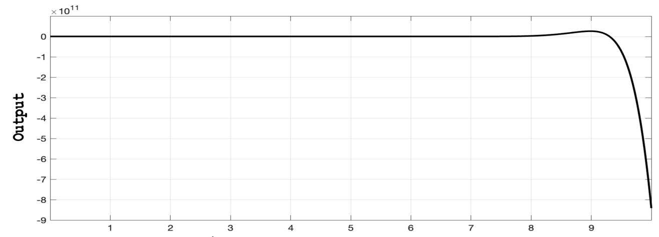
Fig. 23 - Step Resp. of a 4th Order Syst.- 2 Real And 2 Comp. Roots (Ziegler-Nichols)

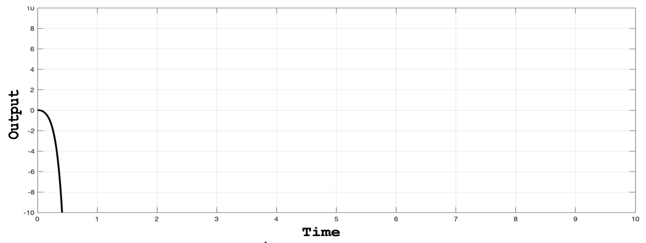
Fig. 24 - Step Response of a 4th Order System - 4 Real Roots (Matlab's PID

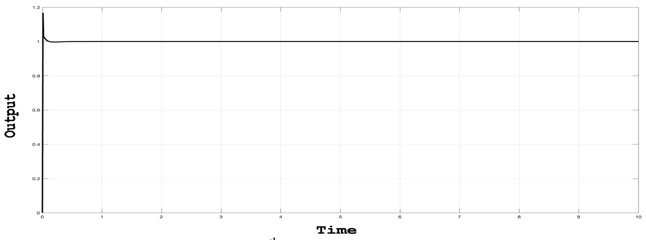
Fig. 25– Step Response of a 4th Order Syst. 0 4 Real Roots (GP-FOPID)

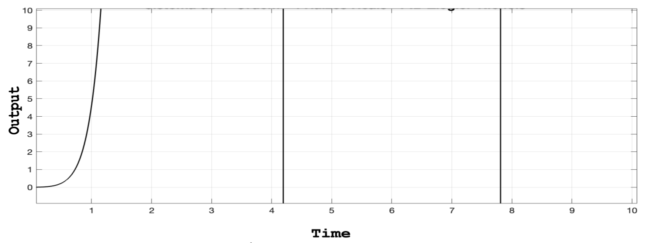
Fig. 26– Step Response of a 4th Order System - 4 Real Roots (Ziegler-Nichols)

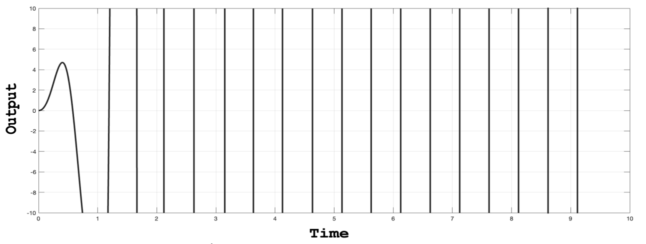
Fig. 27 - Step Resp. of a 4th Order Syst. - 4 Complex Roots (Matlab's PID Tuner)

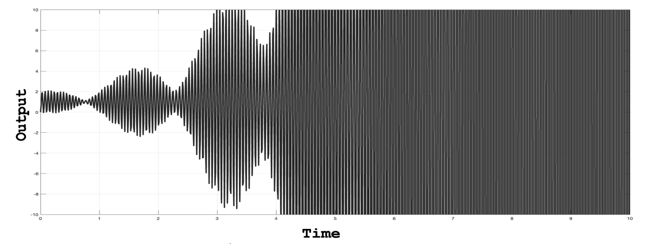
Fig. 28 - Step Resp. of a 4th Order System - 4 Complex Roots (GP-FOPID)

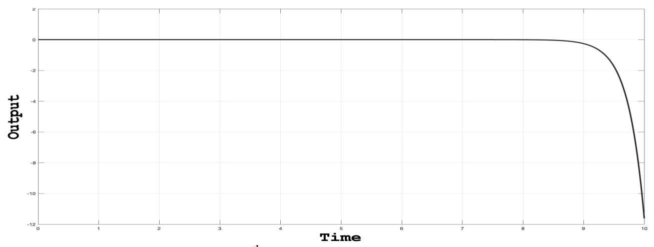
Fig. 29- Step Response of a 4th Order Syst. - 4 Comp. Roots (Ziegler-Nichols)

For illustrative purposes, Figure 30 presents the synthesized topology, obtained through the PG-FOPID method, for controlling the third-order system with three real roots. Table 5 specifies the tuned values for each parameter of the controller.

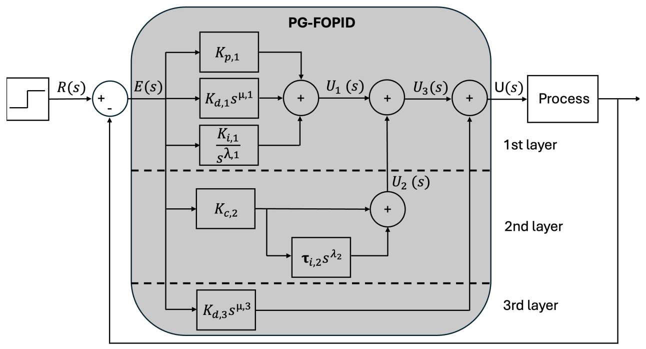

Fig. 30 – FOPID Topology – PG-FOPID – 3rd Oder System – 3 Real Roots

TABLE 5 - FOPID VALUES – PG-FOPID – 3RD ODER SYSTEM – 3 REAL ROOTS

Parameter	Layer	Value
$K_{p,1}$	1	10.641
$K_{d,1}$		0.5920
$K_{i,1}$		4.8350
μ_1		1.898
λ_1		0.6830
$K_{c,2}$	2	10.484
$K_{i,2}$		3.838
λ_2		0.365
$K_{d,3}$	3	3.596
λ_3		1.864

For a quantitative analysis, the performance metric used is the cost function, which was adopted as the fitness of the individuals in the PG-FOPID method, as described in Equation 10, using the same weighting factors presented in Table 2 to ensure a fair comparison between the methods. The results obtained are presented in Table 5.

TABLE 6 – COMPARATIVE NUMERICAL RESULTS IN TERMS OF THE FITNESS FUNCTION IN EQUATION (10)

System	Method	Fitness
2nd order – overdamped	Ziegler-Nichols	48.1636
	PID Auto Tunning	77.6615
	PG-FOPID	**202.8753**
2nd order – critically damped	Ziegler-Nichols	35.8670
	PID Auto Tunning	281.4058
	PG-FOPID	**313.814**
2nd order – underdamped	Ziegler-Nichols	29.8936
	PID Auto Tunning	215.9152
	PG-FOPID	**316.038**
2nd order – non-minimum phase	Ziegler-Nichols	40.0156
	PID Auto Tunning	152.7850
	PG-FOPID	**272.9448**
3rd order – 3 real roots	Ziegler-Nichols	Not applicable
	PID Auto Tunning	29.9231
	PG-FOPID	**220.2414**
3rd order – 2 complex roots and 1 real root	Ziegler-Nichols	13.6432
	PID Auto Tunning	29.9231
	PG-FOPID	**202.0176**

4th order – 2 real roots and 2 complex roots	Ziegler-Nichols	12.5667
	PID Auto Tunning	5.8673
	PG-FOPID	**800.1008**
4th order – 4 positive real roots	Ziegler-Nichols	14.2168
	PID Auto Tunning	20.1520
	PG-FOPID	**489.6046**
4th order – 4 complex roots	Ziegler-Nichols	15.4225
	PID Auto Tunning	62.9710
	PG-FOPID	**143.1670**

VI. CONCLUSIONS AND FURTHER WORK

The results show that the proposed FOPID controllers, synthesized via Genetic Programming (GP), outperformed both the classical Ziegler-Nichols method and the Matlab's PID tuner approach across all simulated plants. These findings underscore the limitations of conventional-structure PID controllers, especially in higher-order systems with complex roots. The proposed method effectively handled dynamic systems up to third order, with both real and complex roots, and a fourth-order system with two complex roots. However, like the other approaches, it failed to control the fourth-order system with four complex roots, revealing limitations that must be addressed for handling more complex systems.

From the exposed, multilayer FOPID controllers with non-conventional structures show strong potential for controlling complex dynamic systems, though their design and implementation remain challenging. In this context, the GP-FOPID method offers a promising, fully automated solution for synthesizing and tuning advanced FOPID controllers.

Relaxing certain constraints used in this study — such as tree depth and parameter bounds for P, I, D, λ, and μ — could allow for synthesizing higher-order controllers, which may better handle fourth-order or more complex systems. However, more complex controllers may introduce issues like noise sensitivity, instability, and actuator saturation.

Future work may explore the GP-FOPID approach for higher-order controller synthesis and integrate complementary techniques, such as predictive models and frequency-domain analysis, to enable a more robust and comprehensive evaluation.

ACKNOWLEDGMENT

This work is supported by financial assistance from CNPq (n°. 313251/2023-1) and CAPES (Funding Code 001).

REFERENCES

[1] S. M. Sam and T. S. Angel, "Performance optimization of PID controllers using fuzzy logic," *2017 IEEE International Conference on Smart Technologies and Management for Computing, Communication, Controls, Energy and Materials (ICSTM)*, Chennai, India, 2017, pp. 438–442, doi: 10.1109/ICSTM.2017.8089200.

[2] [2] J. Zhao and M. Xi, "Self-tuning of PID parameters based on adaptive genetic algorithm," *IOP Conference Series: Materials Science and Engineering*, vol. 782, no. 4, p. 042028, 2020, doi: 10.1088/1757-899X/782/4/04202

[3] D. Maddi, A. Sheta, D. Davineni, and H. Al-Hiary, "Optimization of PID controller gain using evolutionary algorithm and swarm intelligence," *2019 10th International Conference on Information and Communication Systems (ICICS)*, Irbid, Jordan, 2019, pp. 199–204, doi: 10.1109/IACS.2019.8809144.

[4] A. M. López, J. A. Miller, C. L. Smith, and P. W. Murrill, "Tuning controllers with error-integral criteria," *Instrumentation Technology*, vol. 14, pp. 57–62, 1967.

[5] H. Chao, Y. Luo, L. Di, and Y. Q. Chen, "Roll-channel fractional order controller design for a small fixed-wing unmanned aerial vehicle," *Control Engineering Practice*, vol. 18, no. 7, pp. 761–772, 2010.

[6] R. C. Eberhart and Y. Shi, "Comparison between genetic algorithms and particle swarm optimization," in *Proceedings of the IEEE International Conference on Evolutionary Computation*, Anchorage, AK, USA, May 1998, pp. 611–616.

[7] A. Soltoggio, "A comparison of genetic programming and genetic algorithms in the design of a robust, saturated control system," in *Lecture Notes in Computer Science*, vol. 3242, pp. 174–185, 2004, doi: 10.1007/978-3-540-24855-2_16.

[8] M. A. Keane, J. R. Koza, e M. J. Streeter, "Automatic synthesis using genetic programming of improved PID tuning rules," *Preprints of the 2003 Intelligent Control Systems and Signal Processing Conference*, A. E. Ruano, Ed., 2003, pp. 494–499.

[9] S. Das, I. Pan, S. Das, e A. Gupta, "Genetic algorithm based improved sub-optimal model reduction in Nyquist plane for optimal tuning rule extraction of PID and $PI^{\lambda}D^{\mu}$ controllers via genetic programming," *ISA Trans.*, vol. 51, pp. 237–261, 2012.

[10] N. Charan, B. Sahu, D. Bagarty, P. Das, e M. Debnath, "A novel application of ALO-based fractional order fuzzy PID controller for AGC of power system with diverse sources of generation," *Int. J. Electr. Eng. Educ.*, pp. 1–23, 2019, doi: 10.1177/0020720919829710.

[11] K. B. Oldham e J. Spanier, *Fractional Calculus: Theory and Applications of Differentiation and Integration to Arbitrary Order*, New York – London: Academic Press, 1974.

[12] J. R. Koza e R. Poli, *A Genetic Programming Tutorial*, 2003. [Online]. Disponível em: http://www.genetic-programming.com/jkpdf/burke2003tutorial.pdf

[13] L. Vanneschi e R. Poli, "Genetic Programming — Introduction, Applications, Theory and Open Issues," in *Handbook of Natural Computing*, vol. 2, Springer, 2012, ch. 24, pp. 709–739.

[14] P. Nordin, R. E. Keller, e W. Banzhaf, *Genetic Programming: An Introduction: On Automatic Evolution of Computer Programs and Its Applications*, Burlington, MA, USA: Morgan Kaufmann, 1998, pp. 1236–1239.

[15] J. G. Ziegler e N. B. Nichols, "Optimum settings for automatic controllers," *Trans. ASME*, vol. 64, pp. 759–768, 1942.

[16] G. H. Cohen e G. A. Coon, "Theoretical consideration of retarded control," *Trans. ASME*, vol. 75, pp. 827–834, 1953.

[17] K. L. Chien, J. A. Hrones, e J. B. Reswick, "On the automatic control of generalized passive systems," *Trans. ASME*, vol. 74, pp. 175–185, 1952.

[18] P. Anantachaisilp e Z. Lin, "Fractional order PID control of rotor suspension by active magnetic bearings," *Actuators*, vol. 6, no. 1, p. 4, 2017, doi: 10.3390/act6010004.

[19] Garcia, Claudio. Controle de processos industriais: estratégias convencionais. Vol. 1. Editora Blucher, 2021.

[20] M. Reza and A. Nemati, "On Fractional-Order PID Design," Applications of MATLAB in Science and Engineering, Sep. 2011, doi: 10.5772/22657.

[21] Valério, D., & da Costa, J. S. *Tuning of fractional PID controllers with Ziegler–Nichols-type rules.* Signal Processing, 86(10), 2771–2784, 2006.

[22] El-Rifaie, Ali & Abid, Slim & Ginidi, Ahmed & Shaheen, Abdullah. Fractional Order PID Controller Based-Neural Network Algorithm for LFC in Multi-Area Power Systems. Engineering Reports. 7. 10.1002/eng2.70028, 2025.

[23] Ghany, M.A.A., Bahgat, M.E., Refaey, W.M. *et al.* Type-2 fuzzy self-tuning of modified fractional-order PID based on Takagi–Sugeno method. Journal of Electrical Systems and Inf Technol 7, 2, 2020. https://doi.org/10.1186/s43067-019-0009-9.

[24] Muftah, M.N.; Faudzi, A.A.M.; Sahlan, S.; Shouran, M. Modeling and Fuzzy FOPID Controller Tuned by PSO for Pneumatic Positioning System. *Energies*, *15*, 3757, 2022. https://doi.org/10.3390/en15103757.

[25] P. Oliveira, G. Barreto, and G. Thé, "A general framework for optimal tuning of PID-like controllers for minimum jerk robotic trajectories," *Journal of Intelligent & Robotic Systems*, vol. 99, 2020, doi: 10.1007/s10846-019-01121-y.