Journal of Applied Research and Technology

www.jart.icat.unam.mx

Journal of Applied Research and Technology 23 (2025) 240-251

Original

Tuning a PID controller using genetic algorithms

D. López-Reyna[a] • I. López-Reyna[a] • G. González-Badillo[a] •
M. F. Martínez Montejano[b] • R. C. Martínez-Montejano[a]*

[a]*Facultad de Estudios Profesionales Zona Media, Universidad Autónoma de San Luis Potosí, Carretera Rioverde-San Ciro km 4., Ejido puente del Carmen, 79610, Rioverde, S. L. P.*
[b]*Research and Technology, Oak Ridge National Laboratory, 1 Bethel Valley, Rd, Oak Ridge, TN 37830, EEUU.*

Received 09 06 2024; accepted 11 08 2024
Available 06 30 2025

Abstract: This paper details the development of PID controller tuning, based on the implementation of advanced optimization techniques in MATLAB to find the optimal gains for control actions. The methodology used to create the solutions was that of genetic algorithms, an artificial intelligence technique developed in the 1970s and inspired by Darwin's natural selection, within the field of evolutionary computing. Its implementation is based on selection, crossover, and mutation processes, which allow the solutions to iteratively converge towards increasingly optimal results. Two different genetic algorithms were programmed and designed. The first focused exclusively on a single objective, which was the settling time; while the second was based on a multi-objective technique that additionally considered the maximum overshoot, rise time, and delay time. Different fitness functions were developed to create these neural models; subsequently, the gain results obtained from these genetic methods were compared with those proposed by analytical and experimental methods, both in the field of simulation and in physical implementation. The analysis of the responses validated the efficiency and effectiveness of the proposed algorithms for controller tuning, showing better performance with the gains obtained through genetic algorithms.

Keywords: Control, control gains, genetic algorithms.

*Corresponding author.
E-mail address: roberto.montejano@uaslp.mx (R. C. Martínez-Montejano).
Peer Review under the responsibility of Universidad Nacional Autónoma de México.

1. Introduction

Industrial process control originated around 1920 with the early implementation of automatic control in steam boilers. Thanks to the Industrial Revolution, the analysis of control loops intensified, and laws were formulated to describe them more accurately (Borase et al., 2020).

In classical control systems, one of the main elements is the automatic controller, whose main function is to compare the actual output value with the desired value to obtain a difference and reduce deviations, which will improve the system's performance. Although there are various types of controllers designed for industrial applications, this study focuses specifically on the Proportional-Integral-Derivative (PID) controller. It was chosen for its wide versatility to adapt to different systems and its high precision due to the use of the three control actions.

Controllers are regulated by constants that determine the gains in the controller. First, there is the proportional gain Kp that operates as an amplifier that reduces the rise time, then there is the integral gain Ki that eliminates the phase shift generated by the first control action, it also corrects the error in the steady state but can negatively affect the transient response. Finally, there is the derivative gain Kd, whose function is to increase the stability of the system, responding quickly to changes in the error and improving the transient response (Dubey et al., 2022).

There are different methods to adjust these gains. On the one hand, there are experimental methods, such as the one presented by Ziegler and Nichols, which is based on the open-loop system response to a step input. This allows for obtaining a graphic response to identify parameters such as the maximum response height and the delay time; based on these data, predefined formulas are applied to obtain the PID values (Patel, 2020). Another method proposed by these engineers is the "last gain" method, which consists of adjusting a proportional gain that makes the closed-loop system oscillate without it becoming unstable, to determine the period of these oscillations and thus use predefined formulas. Additionally, within this type of method is the Cohen and Coon method, which introduced a self-regulation index and proposed new formulas adapted to systems with more complex dynamics (Suksawat & Kaewpradit, 2021).

On the other hand, there are analytical methods that use mathematical models, such as the Routh-Hurwitz criterion, to ensure the stability of a closed-loop system and establish a range for the PID gains. Additionally, there is the strategy of eliminating at least one stable pole that presents slow dynamics in the plant with a controller zero (Pavan Kumar & Bhimansingu, 2021).

People looking to tune a PID controller need to understand the dynamics of the process. All these methods described, in addition to having benefits, have a major disadvantage and that is that, while they ensure the stability of the system, they do not optimize its operation because some are based on trial-and-error processes. A modern alternative to optimally tune a controller is to use genetic algorithms (GA), which are an artificial intelligence technique capable of providing neural controllers that can learn the behavior of the system by analyzing input and output data. When integrated with other strategies, GAs are very useful, even in the management of complex systems (Suseno & Ma'arif, 2021).

There are various artificial intelligence techniques applied to different fields of research. For example, evolutionary algorithms aim to find a set of parameters that optimize a fitness function, which reflects the "real" value of potential solutions (Rajamani et a., 2021). Additionally, neural networks can be used, which process information in parallel through sets of interconnected neurons; the most widely used neuron in this context is the perceptron, widely used for data classification due to its ability to identify and categorize patterns (Zhang et al., 2022). Another example is fuzzy logic, which is based on using information with few specifications to solve problems, creating a chain of rules based on common sense or the individual's hypotheses, which are integrated with adaptive systems (Saraswat & Suhag, 2023).

The principles of genetic algorithms (GA) were proposed by John H. Holland. These optimization methods are based on the genetic process of living organisms and the Darwinian principle of reproduction and survival of the fittest. Genetic algorithms are useful for generating solutions to problems, which evolve to achieve optimal results, but this is highly dependent on the proper encoding of these solutions (Katoch et al., 2021).

In genetic algorithms (GA), individuals are the possible solutions to the problem, which can be represented as a set of parameters called genes, which when grouped form a sequence called a chromosome. In biological terms, the set of parameters that represents the chromosome is called a phenotype, which contains useful information to build an organism, called a genotype. The adaptation of an individual to a problem depends on the evaluation of the genotype, which can be evaluated from the chromosome using a fitness function designed specifically for each problem. This function must be able to assign a real number that reflects the effectiveness of a solution concerning the problem in question. Therefore, to apply a genetic algorithm it is necessary to have a correct representation of the possible solutions to the problem, a fitness function that determines the viability of said solutions, and the selection of individuals based on their previously given effectiveness. A process of crossing and mutation is also required that allows the reproduction of descendants (Deng et al., 2022).

This work proposes the tuning of a PID controller using genetic algorithm techniques in Matlab and experimentally in Arduino. The objective is to develop a tool in which a first or second-order plant can be introduced, the parameters to be improved in the plant can be selected and the tool can automatically provide the values of the controller gains. The results obtained are compared with traditional methods, both experimental and analytical, to verify their correct operation and the achievement of the objectives.

2. Materials and methods

For the development of this work, a prototype plant was proposed, to then perform its tuning using traditional methods, to obtain a comparison, which is detailed below.

2.1. Prototype plant

The layout established for the control system is intended to be the basis for comparing the different controller tuning methods that will be discussed later.

An RC cascade circuit was chosen, as shown in Figure 1, which has resistance values of 10KΩ and capacitor values of 100μF. To find its transfer function according to the concept focused on a linear time-invariant system —defined as the quotient of the Laplace transform of the output to the Laplace transform of the input, assuming that the initial conditions are zero— the following function was obtained:

$$\frac{Y(s)}{X(s)} = \frac{1}{s^2+3s+1} \quad (1)$$

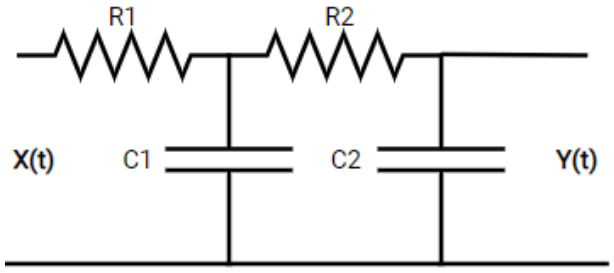

Figure 1. Cascade RC electrical circuit.

2.2. Tuning using the Ziegler and Nichols method of the open loop system

Ziegler and Nichols method was employed by drawing the open loop system diagram using a stepped input of amplitude 5 as illustrated in Figure 2. This facilitated finding an inflection point and drawing a tangent line to the response shown in Figure 3 from which parameters like system height K=5, delay time L=0.4, and time constant τ=2.35 were determined as specified in Figure 4 (Allu & Toding, 2020).

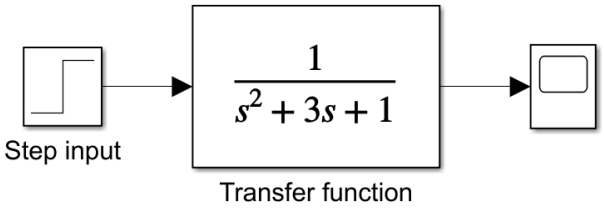

Figure 2. Block diagram of the open loop system.

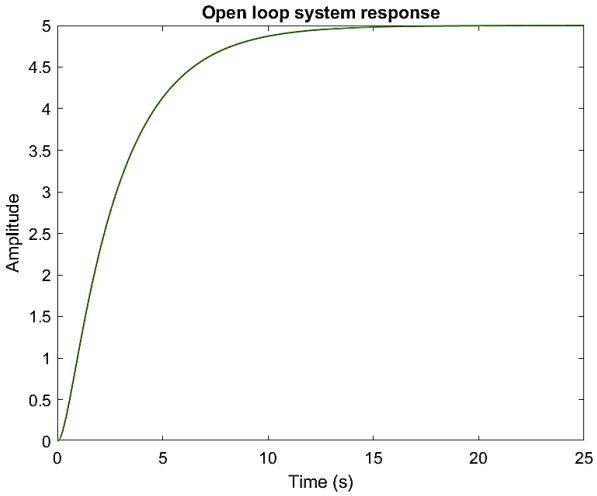

Figure 3. Open loop system response.

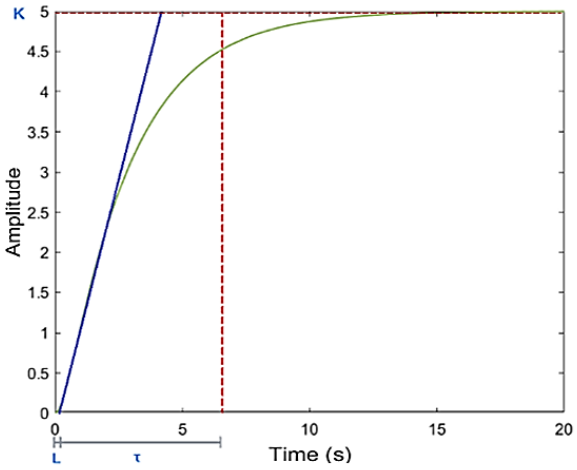

Figure 4. Ziegler-Nichols methodology.

Using the formulas from Table 1, it was possible to characterize the controller, obtaining the gains that regulate it.

$$K_P = 1.41 \quad (2)$$

$$K_I = 1.76 \quad (3)$$

$$K_D = 0.28 \quad (4)$$

Table 1. Gains for the Ziegler-Nichols method.

Controller	K_P	τ_i	τ_d
P	$\frac{\tau}{KL}$	∞	0
PI	$0.9\frac{\tau}{KL}$	$\frac{L}{0.3}$	0
PID	$1.2\frac{\tau}{KL}$	2L	0.5

2.3. Tuning using the Routh-Hurwitz criterion

To establish a comparison between the most common tuning methods, the Routh array (6) was developed starting from the characteristic equation of the system, which results from the product of the controller with the closed-loop plant (5).

$$H(s) = \frac{K_D s^2 + K_P s + K_I}{s^3 + s^2(K_D+3) + s(K_P+1) + K_I} \quad (5)$$

$$\begin{array}{c|cc} s^3 & 1 & K_P+1 \\ s^2 & K_D+3 & K_I \\ s^1 & b_1 & 0 \\ s^0 & c_1 & 0 \end{array} \quad (6)$$

With the support of the formulas defined to determine the constants b1b_1b1 and c1c_1c1, which depend on the other parameters of the array, it was possible to find the ranges for the PID gains necessary to maintain system stability, as this is the focus of the implemented analytical method.

$$K_P = 10 \quad (7)$$

$$K_I = 4 \quad (8)$$

$$K_D = 6 \quad (9)$$

It should be noted that these values were chosen randomly as long as they remained within the range that guarantees the stability of the system. This serves as an example of the different values that these control gains can take.

2.4. Genetic algorithms

Genetic algorithms (GAs) simulate biological evolution processes to solve problems that require optimizing outcomes. These algorithms work with a population of individuals, each representing a solution to the problem. Each solution is assigned a fitness value provided by the fitness function, which is analogous to an organism's ability to compete for resources and survive.

During the execution of the algorithm, parents are selected based on their fitness and then crossed using crossover operators to produce offspring with characteristics of both parents. In addition, a mutation operator is applied to alter some parts of the potential solutions, introducing variability.

Each successive formation creates a new generation, and this process is repeated iteratively until the termination criterion is met, which could be reaching the maximum number of generations or finding the most optimal solution to the problem at hand.

To generate an accurate coding of the GA, the flow chart presented in Figure 5 was followed, which includes the necessary steps that will be explained later.

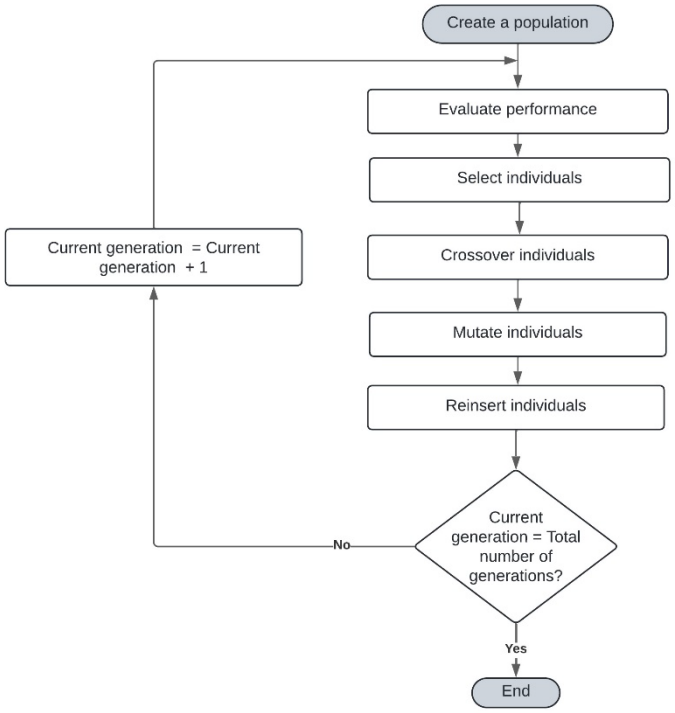

Figure 5. Flowchart for the implementation of genetic algorithms.

2.4.1. Initial population

This process involves generating initial solutions that will serve as starting points for the GA to begin its search for an optimized solution. As previously mentioned, solutions are represented by chromosomes, which are composed of genes that can be encoded as integers, binaries, floats, etc. The most common way to initialize the population is randomly within a range of possible values; this technique can expand the space of explored solutions.

2.4.2. Fitness functions

One of the most crucial stages to achieve good performance with a genetic algorithm is the evaluation of candidate solutions through the generation of fitness functions. The main objective is to ensure that individuals close to each other in the search space have very similar fitness values. Therefore, each individual should be assigned a real number that reflects their fitness, which will help improve the subsequent stages of the GA.

To program the fitness functions that will govern each GA, the flowchart presented in Figure 6 was considered. Initially, using the command window in Matlab, the PID controller gains were assigned according to the block diagram in Simulink (Figure 2). Later, during the simulation of the program, critical data such as system output, controller output, and error were extracted. Finally, these values were used to apply the Matlab functions described in Table 2 to calculate the critical parameters of the system response.

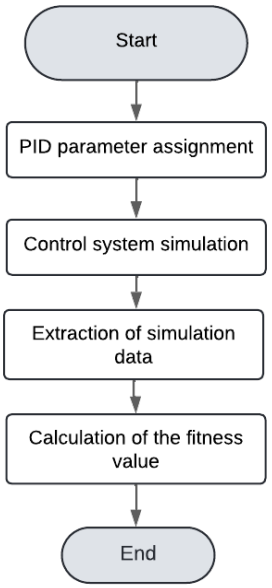

Figure 6. Flowchart for generating the adaptation functions.

Table 2. Gains for the Ziegler-Nichols method.

Matlab function	Description
SettlingTime	Calculate the settling time of the system within a specific threshold close to the final value, which can be 2% or 5%.
Overshoot	Calculate the maximum peak or maximum overshoot that the system's response can reach.
RiseTime	Calculate the time required for the system to rise from 10% to 90% of the final value.
Find	This function was used to locate the first moment when the response reached 50% of the final value in order to obtain the delay time.

One of the additional purposes of this work is to show the differences between a single-objective GA and a multi-objective GA. To this end, a multi-objective fitness function was designed, incorporating all the criteria detailed in Table 2. To achieve this, linear weighting (Scoring) was implemented, a selection method to maximize the satisfaction of the desired results; this involves defining a value function that represents preferences by assigning weights to the various factors that make up the function.

The weights assigned to each parameter are given according to the desired importance, ensuring that the total sum is 100%. This ensures that each criterion contributes proportionally to the value of the multi-objective fitness function (MFF). The sum of these weights is shown below:

$$MFF = (0.4\,T_P) + (0.3\,M_P) + (0.2\,T_s) + (0.1\,T_d) \quad (10)$$

Where:
T_P= Settling time
M_P= Maximum overshoot
T_s= Rise time
T_d= Delay time

On the other hand, for the fitness function of the single-objective GA, only the settling time will be considered, so the direct weight assigned to it in MATLAB is 100%.

2.4.3. Selection of Individuals

This process goes hand in hand with the previous stage, as individuals are chosen based on their performance to provide better results. Selection is a critical stage, as it determines how quickly the algorithm can converge to a solution found in the population.

There are several selection techniques, such as roulette selection, where each individual is assigned a value proportional to their fitness or adaptation, which causes the individuals with the best fitness to be selected. Tournament selection aims to choose a random number of individuals and select the best of this group by directly comparing fitness values between individuals. In addition, rank selection forces the best individual to be selected as a parent, making this a technique with a certain degree of elitism.

The type of selection used to encode the GAs was tournament selection, to achieve greater precision in the results because it has the important advantage of expanding the exploration of possible solutions, which can be understood with the simple example shown in Figure 7.

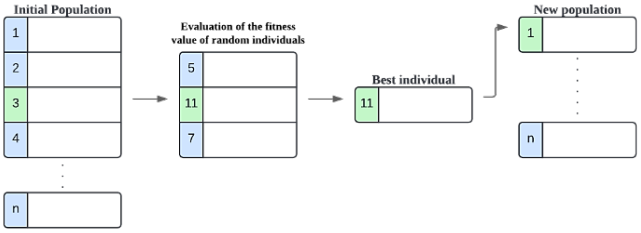

Figure 7. Tournament selection method.

2.4.4. Crossing individuals

Once individuals have been selected to be parents, they must be crossed to create new offspring containing genetic information from both parents. Similar to selection, there are different techniques to apply the crossover operator. One of the most traditional is the one-point crossover, where chromosomes are split at a random point to produce two initial segments and two final segments that will form the new individual. Another technique is the two-point crossover, where the chromosome is split into three parts, allowing the offspring to contain the central subchain from one parent and the lateral subchains from the other, enabling a more varied and complete combination. On the other hand, there is the uniform crossover (Figure 8), which, with the help of a crossover mask, gives the same probability that a gene comes from either of the two parents; this method was chosen for GA programming because it can preserve high genetic diversity.

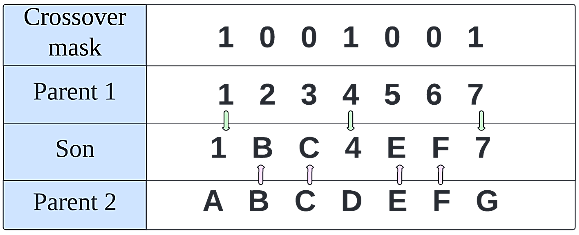

Figure 8. Uniform crossing method.

2.4.5. Mutation of individuals

The mutation operator is responsible for increasing or decreasing the solution space and, in turn, generating genetic variability between individuals. There are several mutation techniques, such as Gaussian mutation, which modifies the value of an individual using a Gaussian random number method. Uniform mutation is another example of this operator, which replaces the value of an individual at random with a higher or lower value.

However, the type of mutation chosen is the one provided by default by MATLAB to work with GAs: adaptive mutation. This method is responsible for increasing or decreasing the mutation rate to improve genetic diversity; as its name indicates, the operator adapts to the problem to achieve better results in the search space.

2.4.6. Programming

Programming was done in MATLAB following the block diagram in Figure 9. To solve the optimization problems, the functions of the optimoptions tool were used, defining the genetic algorithm using the default function @ga.

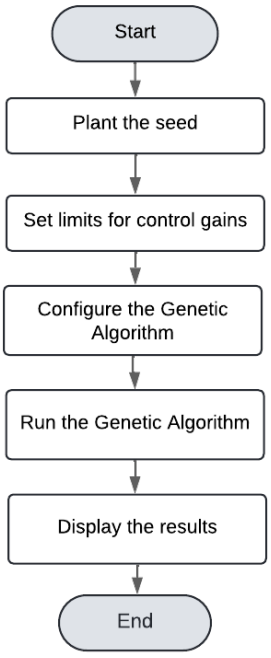

Figure 9. Flowchart for the generation of the GA.

Starting with the initialization of the population, because the technique used is random, Matlab provides the size of the said population, generally calculated as double the search variables; in this case, the population is initialized with 6 individuals. To set the search values between each execution of the program, a seed was established to avoid parameter variation, achieved through the rng (seed) function. After this step, the lower and upper limits for the gains were established, being from 0 to 10 units respectively, because responses with small and positive values are sought.

Later, within the optimal options function, the genetic algorithm was configured, detailing the selection, crossover, and mutation operators selected after the literature review. These were defined using the @selectiontournament, @crossoverscattered, and @mutationadaptfeasible functions, respectively. Additionally, the number of iterations to be produced was specified; as part of the work, a comparison of results was made with maximum generation parameters of 1, 5, and 10 iterations.

Once these parameters were configured, the GA was executed, considering that it must be applied to the single-

objective or multi-objective fitness function. Finally, to observe the results, the three elements, Kp, Ki, and Kd, were printed in vector form.

2.5. Interface

To complete the development of this study, a GUI (Graphical User Interface) was generated to have a simple interaction with the designed genetic algorithms. Using the GUIDE design editor (Figure 10), which automatically generates the MATLAB code for the construction of the interface, the application's behavior was modified.

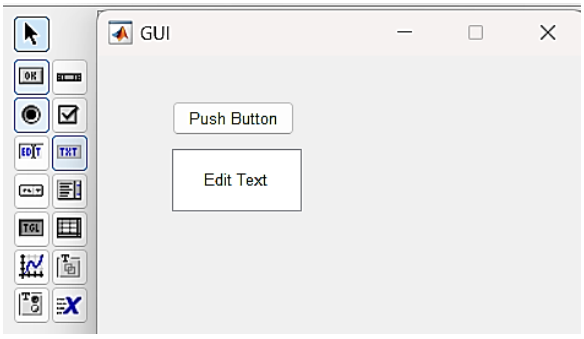

Figure 10. Matlab GUIDE designer.

The interface design aims to be intuitive and understandable, facilitating both the configuration and the analysis of the plant to be controlled. The coefficients of the transfer function are entered in the “denominator” and “numerator” fields, using square brackets to ensure a clear and structured text entry. These coefficients are sent to the block diagram, which serves as a basis for executing the iterations of the genetic algorithm.

Once the transfer function has been loaded, it is possible to analyze it and apply control using the selected algorithm, observing the system responses individually or in a general graph that allows comparing the behaviors.

In addition, two contextual help buttons have been included next to the algorithms. These buttons detail the key parameters in the implementation of each algorithm, providing the user with the necessary information to make an informed choice.

3. Results

First, the GUIDE interface design is presented, which includes the text fields and buttons described above to make the project more understandable and functional. As an example, the proposed plant application (1) with 10 development iterations is shown (Figure 11).

As mentioned above, the responses of the systems with different values in the PID control gains were compared, which were obtained by applying both genetic algorithms with maximum generations of 1, 5 and 10.

To verify that the fitness value was improving (decreasing), the @gaplotbestf function was used, which generates a graph of the fitness value in relation to the generations created by the algorithm. As can be seen in Figure 12, the average and best fitness values begin to converge as the generations increase.

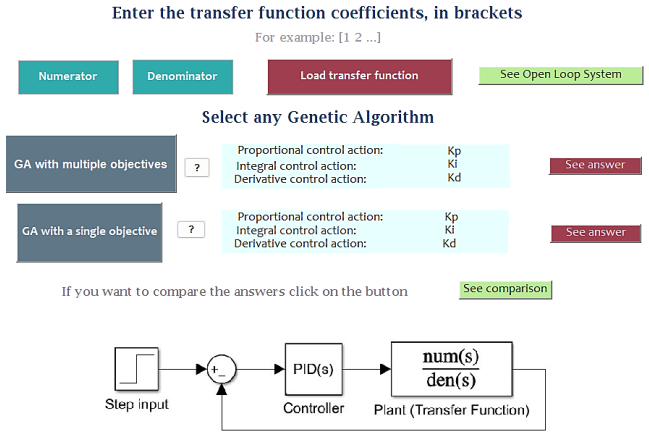

Figure 11. Graphical user interface.

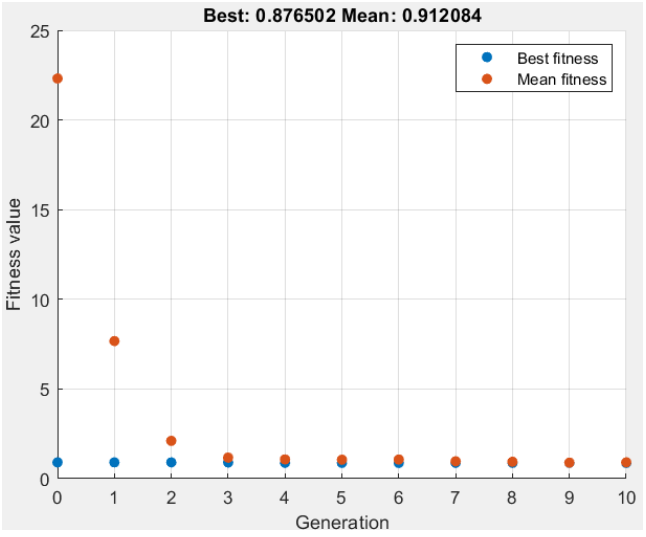

Figure 12. Graph of the evolution of the fitness value over 10 generations in the single-objective algorithm.

Figure 13 shows the result of the execution of the interface to obtain the gains with 10 generations produced, taking into account that the selected plant is the one developed previously (1). It is also possible to show the comparison of the responses of both the open loop system and the system with the PID controller obtained recently, using the “See comparison” button.

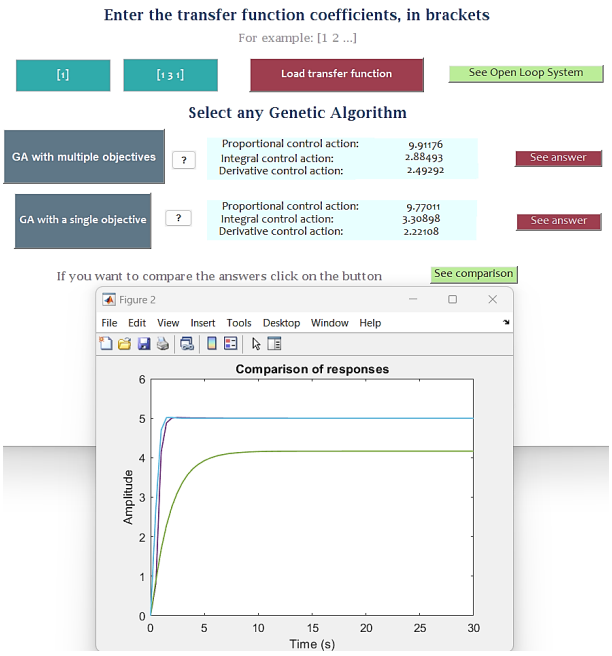

Figure 13. System response given control gains with 10 maximum generations.

Table 3 presents the results obtained by the GAs and the analytical and experimental methods discussed previously. It is observed that the integral and derivative gain values in the single-objective genetic algorithm (GA) remain constant. This is because the focus of said GA is on minimizing the settling time. In this case, a balance is generated between both control actions, since by increasing Kp, the settling time tends to decrease; while, by increasing Ki, the settling time increases. This allows us to predict that the settling time will not vary significantly between iterations. On the other hand, the gains obtained in the multi-objective GA show variation in the three parameters, which shows that more important metrics of the natural response of the system are considered. In addition, as the iterations progress, the responses begin to converge. This can be seen in the table, where the values obtained in 5 and 10 generations show less variation compared to the results after 1 generation. Table 4 describes the evaluation of the response characteristics calculated using the functions shown in Table 2. It is important to note that a 2% threshold was used, within which the system is considered stable, and this time was obtained in a simulated and ideal manner.

This table illustrates how the settling time benefits significantly from using the controllers provided by the genetic algorithm compared to traditional methods. Focusing on the comparison between the single-objective GA and the multi-objective GA, the former shows greater overshoots than the latter, as the multi-objective GA takes this criterion into account to a greater extent when evaluating the system. However, the delay time and rise time do not show significant changes, as their weight in the fitness function is very low.

Table 3. Characteristics of the system responses to the applied controllers, through simulation.

Method	Generation	Kp	Ki	Kd
Single-objective GA	1	9.488	3.3090	2.2211
	5	9.7389	3.3090	2.2211
	10	9.7701	3.3090	2.2211
Multi-objective GA	1	9.4889	3.3090	2.8975
	5	9.9118	2.9474	2.6179
	10	9.9118	2.8849	2.4929
Rout-Hurwitz criterion	Unique	10	4	6
Ziegler-Nichols	Unique	1.41	1.76	0.286

Table 4. Characteristics of the system responses to the applied controllers, through simulation.

Method	Generation	T_p(s)	M_p(%)	T_s(s)	T_d(s)
Single-objective GA	1	0.9036	1.865	0.6236	0.2674
	5	08799	1.9327	0.6089	0.2769
	10	0.8762	1.9409	0.6071	0.2769
Multi-Objective GA	1	1.0979	0.2775	0.6613	0.2343
	5	1.0058	0.00023	0.6280	0.2591
	10	0.9702	4.38e-05	0.62067	0.2774
Routh-Hurwitz	Unique	1.8167	1.0458		
Ziegler-Nichols	Unique	9.8272	266.4567	1.6453	1.2035

The settling time is a crucial factor in the single-objective genetic algorithm. When analyzing the tenth iteration, it is observed that the reduction in the settling time compared to the multi-objective genetic algorithm, the analytical method, and the experimental method is 9.68%, 52.03%, and 91.08%, respectively. This decrease in the settling time also impacts the rise time, which for this algorithm is lower than the others, showing improvements of 3.12%, 21.02%, and 63.10%, respectively.

Regarding the overshoot parameters, the multi-objective algorithm stands out compared to the others. Regarding the single-objective GA, the analytical method and the experimental method present reductions in the overshoot of 99.99%, 99.58%, and 99.99%, respectively.

On the other hand, in terms of delay time, the analytical method demonstrates a better response. It is worth noting that the values of the gains Kp and Ki are very similar in all algorithms, although Ki shows a higher value, which is associated with the improvement in this parameter compared to the single-objective GA, the multi-objective GA and the

experimental method, with decreases of 48.39%, 47.50%, and 88.12%, respectively.

Figures 14 and 15 show the system responses with the controllers specified by the three generations taken. As observed, in both cases, the signal between the fifth and the tenth generation begins to look very similar, as their fitness values are closer to each other, as seen in Figure 12. Figure 16 provides a clearer visualization of the differences between each of the four methods used throughout this study.

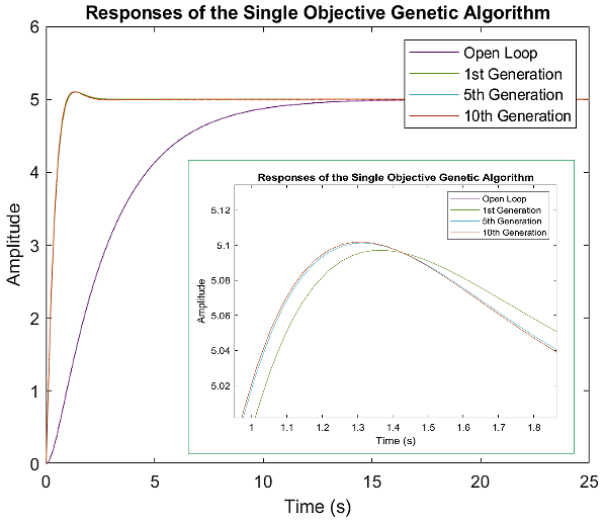

Figure 14. Comparison of system responses with simple GA applied in different generations.

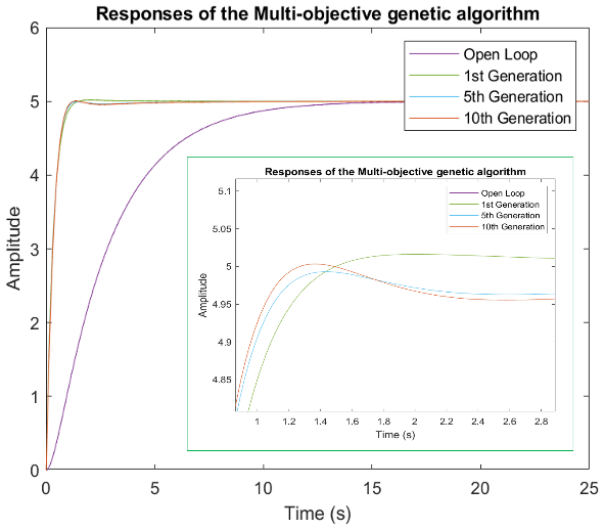

Figure 15. Comparison of system responses with multiple GA applied in different generations.

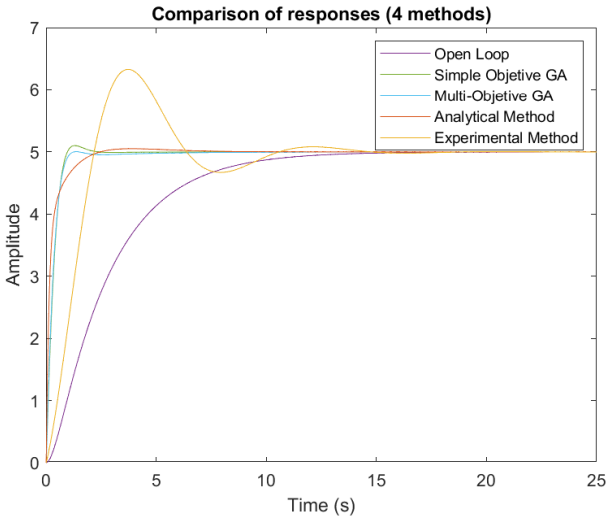

Figure 16. Comparison of system responses to different controllers.

These results were also obtained using Arduino and its ease of connection to Simulink. To visualize the comparisons clearly, Table 5 is prepared, where each parameter was obtained from scale measurements in Figures 18 and 19, which present more realistic results of the work carried out. The overshoot percentage stands out, which is significantly higher in traditional methods compared to optimization methods. On the other hand, the settling time presents very similar values between the different approaches, although its final choice will depend on the specific application of the control system. This will allow the most appropriate method to be selected based on other key characteristics. In general, both the rise time and the delay time remain in very similar ranges between the different methodologies.

Table 5. Characteristics of the system responses to the applied controllers using Arduino.

Method	Generation	T_p(s)	M_p(%)	T_s(s)	T_d(s)
Single-objective GA	1	12.83	34.5	1.2	0.83
	5	13.33	36	1	1.16
	10	13.58	36.6	0.83	1.16
Multi-Objective GA	1	15.5	37	1.3	1
	5	14.62	32	1.21	1.08
	10	14.58	31.6	1.18	1.13
Routh-Hurwitz	Unique	10	36.84	1.64	1.2
Ziegler-Nichols	Unique	15.6	54.73	1.28	1.29

In a more realistic context regarding the system results and when analyzing the tenth generation, it is highlighted that the multi-objective genetic algorithm presents reductions in the maximum overshoot and delay time parameters compared to the single-objective genetic algorithm, the analytical method and the experimental one, with percentage differences of 13.6%, 14.22% and 42.26% for the overshoot, and 2.58%, 5.8% and 12.4% for the delay time, respectively.

The single-objective genetic algorithm stands out in this comparison for its notable decrease in the rise time compared to the multi-objective GA, the analytical method, and the experimental one, with reductions of 29.66%, 49.39%, and 35.15%, respectively.

Finally, one of the most important criteria to evaluate is the settling time. Thanks to the implementation of an analytical method, a stable response was achieved in approximately 10 seconds. This represents percentage differences compared to the other methods, such as the single-objective genetic algorithm, the multi-objective genetic algorithm, and the experimental method, with values of 26.36%, 31.14%, and 35.89%, respectively.

The circuit in Figure 1 was implemented and through a block arrangement in Simulink, PWM inputs were specified, and the controllers through a sum of gains and analog outputs to read the signal provided by the system, as can be seen in Figure 17.

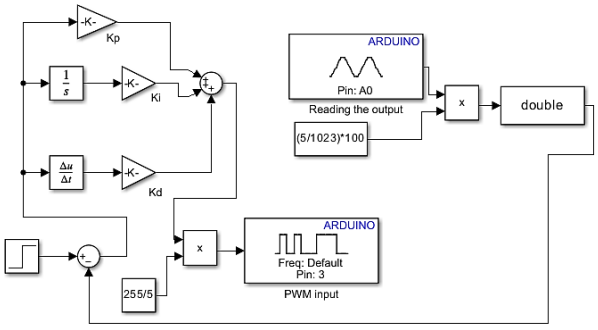

Figure 17. Block diagram for the implementation of the system using Arduino.

The signals shown in Figure 18 demonstrate how the settling time and the percentage of overshoot of the single-objective GA signals increase slightly with each generation taken. However, this changes when applying the controllers from the multi-objective GA, as both criteria decrease significantly, as shown in Figure 19.

Another observation supporting the theory of the controllers is that as the proportional gain value increases, the rise time decreases, but the delay time increases.

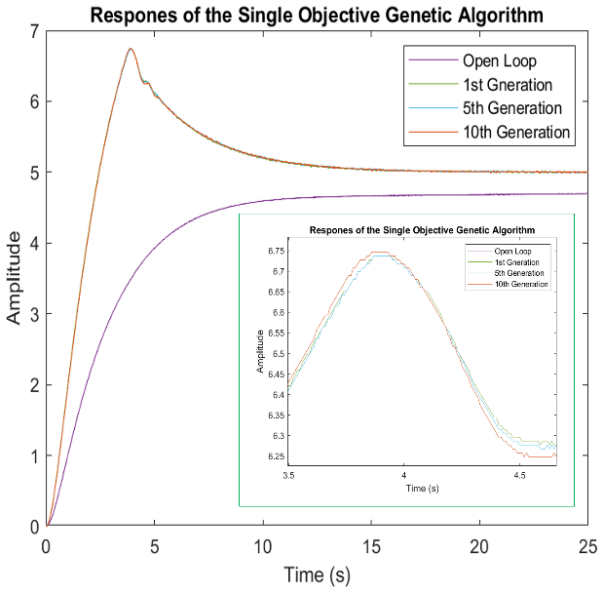

Figure 18. Comparison of system responses with the simple GA applied using Arduino.

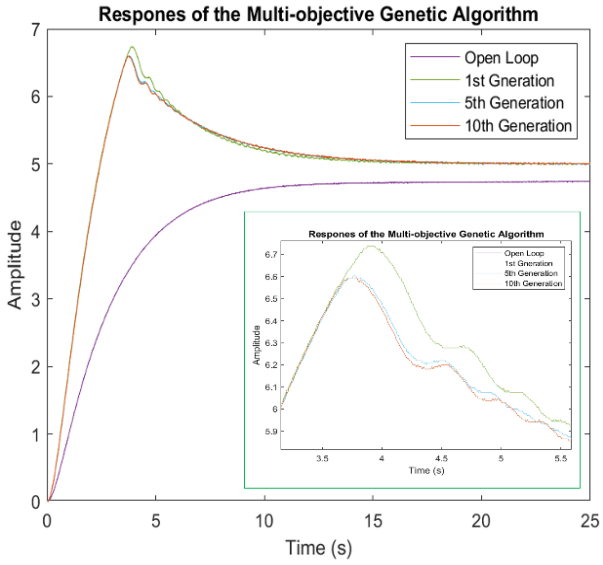

Figure 19. Comparison of the system responses with the multiple GA applied using Arduino.

Figure 20 shows the signals from the applied methods to determine, in a more realistic environment, which method offers the best response. It can be seen that the analytical method provides values similar to those determined by the neural models, resulting in very similar responses. In contrast, the experimental method, as observed in the simulation, shows several oscillations before settling at the desired value.

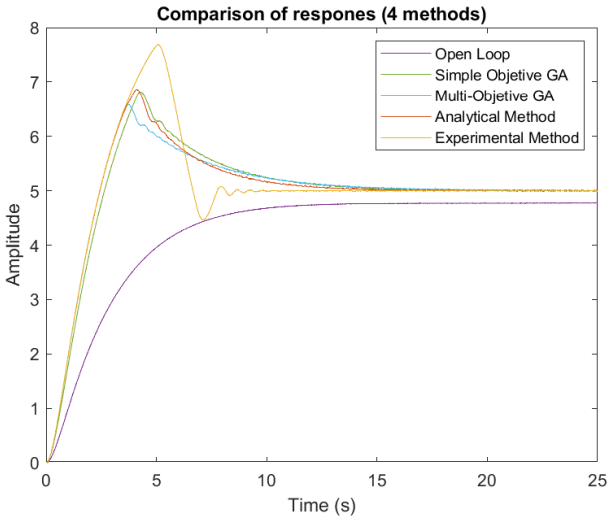

Figure 20. Comparison of system responses to different controllers applied using Arduino.

4. Conclusions

To develop genetic algorithms in MATLAB, the main characteristics of classic control system responses were considered, such as settling time, maximum overshoot, rise time, and delay time. Additionally, various existing techniques were evaluated to construct each stage of these neural models. One of the most important aspects was to clearly define the parameters for the development of the fitness function from the outset, as it verifies that the performance of the solution is optimal for solving control problems.

It was demonstrated that traditional methods can be time-consuming and inaccurate due to their wide variability in possible controller gains, as seen with experimental methods. As observed in the graphs, their application to the specified plant generated high oscillations and overshoots compared to other responses.

The significant effectiveness of using genetic algorithms to improve system response was evident. Although the developed open-loop system did not naturally exhibit noise or oscillations, the settling time was improved both ideally (simulated) and practically through the implementation of the plant and Arduino.

Furthermore, one of the focuses of this study was the comparison between a single-objective GA and a multi-objective GA. Given the PID values, the results showed little overall variation, but a more noticeable difference in both settling time and overshoot.

The use of multiple objectives for the evaluation of the system in the natural state showed an improvement in the results when considering the added weight in the total function using the scoring technique, thus prioritizing the key metrics. Regarding the execution time, it is worth noting that it is the same in both algorithms, approximately 18 seconds per iteration since both focus on a single evaluation function. This highlights the importance of using artificial intelligence techniques to find these values. From another perspective, these values can be programmed directly into the microcontroller or PLC that controls the process, without requiring additional tools, thus improving the response of an industrial system, which leads to greater efficiency and cost reduction.

Based on the graphical results, improvements were declared not only in the output signal of the control systems but also in the resolution time of control problems, which can be tedious and, at an industrial level, could affect the performance of the system being worked on.

As future work, it is proposed to complement the optimization of PID tuning with hybrid optimization methods such as Particle Swarm Optimization or Simulated Annealing, as well as to implement it in larger and more complex systems, such as gain control in Atomic Force Microscopy (which is based on PID gains), motor speed control with PLCs, energy balancing in electrical distribution microgrids, phase-locking algorithms for energy synchronization, differential robots in trajectory tracking, to mention a few.

Conflict of interest

The authors have no conflict of interest to declare.

Acknowledgments

The authors would like to thank the Unidad Académica Multidisciplinaria Zona Media of UASLP for supporting this work.

Funding

The authors received no specific funding for this work.

References

Allu, N., & Toding, A. (2020). Tuning with Ziegler Nichols method for design PID controller at rotate speed DC motor. In *IOP Conference Series: Materials Science and Engineering* (Vol. 846, No. 1, p. 012046). IOP https://doi.org/10.1088/1757-899X/846/1/012046

Borase, R. P., Maghade, D. K., Sondkar, S. Y., & Pawar, S. N. (2020). A review of PID control, tuning methods and applications. *International Journal of Dynamics and Control*, *9*, 818-827. https://doi.org/10.1007/s40435-020-00665-4

Deng, W., Zhang, X., Zhou, Y., Liu, Y., Zhou, X., Chen, H., & Zhao, H. (2022). An enhanced fast non-dominated solution sorting genetic algorithm for multi-objective problems. *Information Sciences*, *585*, 441-453. https://doi.org/10.1016/j.ins.2021.11.052

Dubey, V., Goud, H., & Sharma, P. C. (2022). Role of PID control techniques in process control system: a review. *Data Engineering for Smart Systems: Proceedings of SSIC 2021*, 659-670. https://doi.org/10.1007/978-981-16-2641-8_62

Katoch, S., Chauhan, S. S., & Kumar, V. (2021). A review on genetic algorithm: past, present, and future. *Multimedia tools and applications*, *80*, 8091-8126. https://doi.org/10.1007/s11042-020-10139-6

Patel, V. V. (2020). Ziegler-nichols tuning method: Understanding the pid controller. *Resonance*, *25*(10), 1385-1397. https://doi.org/10.1007/s12045-020-1058-z

Pavan Kumar, Y. V., & Bhimasingu, R. (2021). Design of voltage and current controller parameters using small signal model-based pole-zero cancellation method for improved transient response in microgrids. *SN Applied Sciences*, *3*, 1-17. https://doi.org/10.1007/s42452-021-04815-x

Rajamani, M. P. E., Rajesh, R., & Willjuice Iruthayarajan, M. (2021). Design and Experimental Validation of PID Controller for Buck Converter: A Multi-Objective Evolutionary Algorithms Based Approach. *IETE Journal of Research*, *69*(1), 21–32. https://doi.org/10.1080/03772063.2021.1905564

Saraswat, R., & Suhag, S. (2023). Type-2 fuzzy logic PID control for efficient power balance in an AC microgrid. *Sustainable Energy Technologies and Assessments*, *56*, 103048. https://doi.org/10.1016/j.seta.2023.103048

Suksawat, T., & Kaewpradit, P. (2021). Comparison of Ziegler-Nichols and Cohen-Coon tuning methods: implementation to water level control based MATLAB and Arduino. *Engineering Journal Chiang Mai University*, *28*(1), 153-168.

Suseno, E. W., & Ma'arif, A. (2021). Tuning of PID controller parameters with genetic algorithm method on DC motor. *International Journal of Robotics and Control Systems*, *1*(1), 41-53. https://doi.org/10.31763/ijrcs.v1i1.249

Zhang, L., Li, S., Xue, Y., Zhou, H., & Ren, Z. (2022). Neural network PID control for combustion instability. *Combustion theory and modelling*, *26*(2), 383-398. https://doi.org/10.1080/13647830.2022.2025908