FEDERAL UNIVERSITY OF TECHNOLOGY — PARANÁ

CARLOS DA CONCEIÇÃO CASTILHO NETO

ADAPTIVE TECHNIQUES OPTIMIZED BY BIO-INSPIRED ALGORITHM FOR THE CONTROL OF A BLDC MOTOR

PONTA GROSSA

2022

CARLOS DA CONCEIÇÃO CASTILHO NETO

ADAPTIVE TECHNIQUES OPTIMIZED BY BIO-INSPIRED ALGORITHM FOR THE CONTROL OF A BLDC MOTOR

Técnicas adaptativas otimizadas por algoritmo bio-inspirado para o controle de um motor BLDC

Dissertation presented as a requirement to obtain Master 's degree in Electrical Engineering of Federal University of Technology — Paraná (UTFPR). Concentration Area: Instrumentation and Control.

Advisor: Prof. Ph.D. Fernanda Cristina Corrêa

PONTA GROSSA

2022

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Ponta Grossa

CARLOS DA CONCEICAO CASTILHO NETO

ADAPTIVE TECHNIQUES OPTIMIZED BY BIO-INSPIRED ALGORITHM FOR THE CONTROL OF A BLDC MOTOR

Trabalho de pesquisa de mestrado apresentado como requisito para obtenção do título de Mestre Em Engenharia Elétrica da Universidade Tecnológica Federal do Paraná (UTFPR). Área de concentração: Controle E Processamento De Energia.

Data de aprovação: 07 de Dezembro de 2022

Dra. Fernanda Cristina Correa, Doutorado - Universidade Tecnológica Federal do Paraná

Dr. Frederic Conrad Janzen, Doutorado - Universidade Tecnológica Federal do Paraná

Dr. Jony Javorski Eckert, Doutorado - Universidade Estadual de Campinas (Unicamp)

Dra. Marcella Scoczynski Ribeiro Martins, Doutorado - Universidade Tecnológica Federal do Paraná

Dedico este trabalho a minha família
e aos meus amigos, pelos momentos
de ausência.

ACKNOWLEDGMENTS

This paper couldn't be completed without the help of several people.These clarifications do not meet all the people who were part of this important phase of my life. Therefore, they are already among those appreciated that are not present words, but they can be sure that they are part of my gratitude.

First of all to God for strengthening me and providing me with all things necessary to achieve this goal. To my wife Monique, my parents and grandparents, for their love, encouragement and total support in all the moments of my life that were fundamental for the accomplishment of this dream.

My mentor, Professor Fernanda, who showed me the ways to be followed, for the trust placed and the real desire to help me understand all the content covered in this work. To all the professors and colleagues in the department, who helped in a direct and indirect way, among them Fabio Galvão Borges.

The Federal University of Technology — Paraná (UTFPR) for providing a great organizational structure, laboratories and tools needed to carry out this study.

Finally, to all who contributed in some way to the realization of this research.

Don't give up... Don't you quit. You keep walking. You keep trying. There is help and happiness ahead... You keep your chin up. It will be all right in the end. Trust God and believe in good things to come. (HOLLAND, 1999).

ABSTRACT

Long operating life, high dynamic response and efficiency, and higher torque-to-weight ratio make the BLDC motor attractive for several applications, such as electric vehicles, drones, etc. Due to its versatility, the BLDC motor can be exposed to applications in which load disturbances, sudden disturbances, and parameter variation occur, making conventional control techniques such as proportional-integral-derivative PID controller not reach its variables with precision and agility. To avoid this inconvenience, the PID controller can improve its performance when used in conjunction with adaptive techniques that collect data from the system's operating environment and perform adjustments based on the condition it is in, dynamically minimizing system failures.Adaptive techniques based on fuzzy logic or Gaussian functions can be an alternative to this impasse. However, the choice of parameters and variables empirically are some obstacles that the designer faces, requiring full knowledge of the system's behavior to which they will be applied. For this, using a metaheuristic optimization algorithm such as Particle Swarm Optimization (PSO) would be a solution, this bio-inspired algorithm searches in a complex universe of multiple solutions the best one for a given problem. This paper aims to compare different control techniques, such as PID, Hybrid Fuzzy-PID Hybrid Fuzzy-PID, and GAPID optimized by PSO for speed control of a BLDC motor, through simulations performed in the Simulink software and its practical implementation in an ESP32 microcontroller.

Keywords: fuzzy; controllers; particle swarm optimization; metaheuristic.

RESUMO

Vida operacional longa, alta resposta dinâmica e eficiência e maior relação torque-peso são características que fazem o motor BLDC ser atrativo para diversas aplicações, como veículos elétricos, drones e etc. Entretanto, devido a sua versatilidade o motor BLDC pode ser exposto a aplicações em que ocorram distúrbios de carga, perturbação súbita e variação de parâmetros, fazendo com que as técnicas de controle convencionais como controlador proporcional-integral-derivativo PID, não alcancem suas variáveis com precisão e agilidade. Para evitar esse inconveniente, o controlador PID pode ter o seu desempenho melhorado quando utilizado em conjunto com técnicas adaptativas que coletam dados do ambiente de operação do sistema e realizam ajustes baseados na condição em que ele se encontra de forma dinâmica minimizando falhas no sistema.Técnicas adaptativas baseadas em lógica difusa ou que utilizam funções Gaussianas podem ser uma alternativa para esse impasse.Porém, a escolha dos parâmetros e variáveis de forma empírica são alguns obstáculos que o projetista enfrenta, requerendo do mesmo pleno conhecimento do comportamento do sistema ao qual serão aplicadas. Para isso o uso de um algoritmo de otimização metaheurística como o Particle Swarm Optimization (PSO) seria uma solução, esse algoritmo bio-inspirado busca em um universo complexo de múltiplas soluções melhor resultado para o dado problema. Tem-se como objetivo realizar neste trabalho a comparação entre diferentes técnicas de controle, como PID, Híbrido Fuzzy-PID e Híbrido Fuzzy-PID e GAPID otimizado pelo PSO para o controle de velocidade de um motor BLDC por meio de simulações realizadas no software Simulink e a sua implementação prática em um microcontrolador ESP32.

Palavras-chave: fuzzy; controladores; otimização por enxame de partículas; metaheurística.

LIST OF ILLUSTRATIONS

LIST OF INITIALS

BLDC	Brushless Direct Current Electric Motor
DC	Direct Current
ESC	Electronic Speed Controller
GAPID	Gaussian Adaptive Proportional, Integral, and Derivative Controller
PI	Proportional–Integral
PID	Proportional–Integral–Derivative
PO	Percentage Overshoot
PSO	Particle Swarm Optimization
RPM	Revolutions per minute
UTFPR	Universidade Tecnológica Federal do Paraná
ZOH	Zero-Order Hold

LIST OF SYMBOLS

K_p	Proportional gain	
K_i	Integral gain	
K_d	Derivative gain	
$u(t)$	Control signal	
$e(t)$	Steady-state error	
c_1	Cognitive parameter	
c_2	Social Scaling Parameter	
$pBest$	Local best position	
$gBest$	Global best position	
T_s	Sampling time	[s]
$G(s)$	Plant transfer function in the S plane	
$P(z)$	Plant transfer function in the Z plane	
$U(z)$	Differential equation	
T_e	Stabilization time	[s]
$K(z)$	Controller transfer function	
E	Error signal	
de	Derivative of the error signal	
e_k	Current sample signal error	
$e_{(k-1)}$	Previous sample signal error	
u_k	Control effort	
$u_{(k-1)}$	Previous control effort	
$R\%$	Resolution	[%]
ω	Inertia weight	
ζ	Damping ratio	
θ_p	Plant angle	
θ_k	Controller angle	
θ_n	Controller numerator angle	
ω_n	Natural frequency	[Hz]
ω_d	Damped frequency	[Hz]
α	Discrete proportional–integral controller zero	
γ	Error	

SUMMARY

1 INTRODUCTION

The replacement of coal with electricity, oil with steam energy, and steel in production techniques were milestones for the beginning of the second industrial revolution.

At that time, several scientists and researchers were fundamental to our current technologies. Morit Hermann von Jacobi stands out in 1839 with the development of a 1000 Watt engine to be coupled to a boat, where it was driven by zinc-platinum batteries that together weighed more than 200 kilograms (DOPPELBAUER, 1822).

Even with all the technology and knowledge we have nowadays, this example illustrates the current industry's challenges, especially concerning electric vehicles. The barriers to developing components, motors, and electric batteries with long life, high performance, and low weight have been the focus of several researchers since then. These factors prevented the large-scale replacement of steam engines by electric motors at that moment. However, it inspired others to produce electric motors with the same power standards and different topologies.

According Doppelbauer (1822), in 1866 Werner von Siemens built an electric machine without a permanent magnet with a power of approximately 30 Watts. This machine could work as a generator using the effect of self-excitation and as a motor as long as a direct current DC was applied at its terminals.

About 30 years after Werner's achievement, the three-phase squirrel-cage induction motor emerges. The Russian Michael von Dolivo Dobrowolsky developed this equipment. The motor was more silent, had an approximate efficiency of 80% and had a longer useful life when compared to previous electric motor models (DOPPELBAUER, 1822).

In the early 1960s, the first concepts of the Brushless Direct Current Electric motor (BLDC) began to appear due to the popularity and advances in the development of solid-state technology. T.G.Wilson invented the precursor of the motor without brushes, and P.H. Trickey called a DC machine with solid-state commutation (XIA, 2012).

During the same period as the discovery and development of electric motors, the need to obtain control over equipment and processes, such as the flow control to regulate a water clock or the speed control of the grinding stone in a mill encouraged the development of several control techniques (FRANKLIN; POWELL; EMAMI-NAEINI, 2013).

Indeed, these techniques were also applied to direct current electric motors seeking speed control to obtain maximum performance and efficiency. Because of the various techniques developed, one stands out for having robustness and easy empirical adjustments, the Ziegler-Nichols method, which is used nowadays in most industrial meshes (KUMAR; SWAIN; NEOGI, 2017).

The increasing number of replacements of DC motors by BLDC motors has been taking place over the years because BLDC motors have a longer service life, lower noise emission, and high dynamic response. Such attributes make the motor BLDC attractive, mainly for electric vehicles, robotics, and aviation.

The simplicity of the PID controller is its strong and weak points. In such applications, the BLDC motor is exposed to different types of load disturbance, which can make the controller unresponsive, not reaching the desired performance accurately in a short response time. In other words, the PID controller may not be the most suitable option for specific cases (GHANY; SHAMSELDIN; GHANY, 2017).

However, the PID controller can improve its performance by applying adaptive techniques such as Fuzzy logic, which is necessary for dynamic systems in unstable environments. In this way, the union of a PID controller to a Fuzzy controller results in a controller known as a Fuzzy-PID hybrid controller. In this hybrid structure, the Fuzzy-PID controller integrates the advantage of both control structures, improving PID control even when plant parameters vary or a disturbance occurs (GOSWAMI; JOSHI, 2018).

In a universe of countless alternatives, another option would be Gaussian Adaptive Proportional, Integral, and Derivative controller (GAPID), generally applied to typical power supplies of medical equipment with the Buck converter. This controller dynamically modifies the PID controller gains to achieve better performance. Even though there are not many examples of the application of this controller to electric motors, it should be adequate, practical, and straightforward to apply to it (PUCHTA *et al.*, 2021).

According to Simões and Shaw (2007) and Puchta *et al.* (2021), there is no algebraic solution for the adaptive parameters tuning of both controllers (GAPID and Fuzzy-PID), which requires the designer's full knowledge of the system whereupon it will be applied, making the choices adopted not being the ideal ones to obtain the maximum controller performance.

Thus, one of the ways to obtain these parameters would be using additional

tools, such as Particle Swarm Optimization (PSO). This bio-inspired algorithm is based on the interaction between several particles with a range of random values in search of the best resolution for the proposed problem.

1.1 Research Delimitation

At the beginning of the 20th century, electric vehicles peaked, representing a share of 33% of the entire automotive chain. However, the discovery of large oil reserves, government incentives, and the mass production created by Henry Ford made combustion vehicles gain ground. In addition, their competitive values and significant range negatively influenced the popularity of electric vehicles, as they were slow with an average speed of 20 km/h and cost twice as much as internal combustion vehicles (MATULKA, 2014).

Over the years, the use of petroleum derivatives resulted in a significant emission of pollutants to the environment and the development of high dependence on this non-renewable energy source. For such adversity, sustainability has become society's focus, seeking alternatives that help the planet, mainly related to urban mobility. In this way, with the advancement of technology, the electric and hybrid vehicles' industry has been increasing in recent years due to significant investments in search to increase the range of the electric vehicle and get a better price (IZO, 2018).

In order to achieve this objective, the vehicle must have batteries with high energy density per mass, high-performance control techniques, an effective energy management system, and an engine that has a high dynamic response, high efficiency, and a higher ratio torque-weight, characteristics that make up a BLDC motor (JAVORSKI ECKERT *et al.*, 2018). Thus, studying control techniques to control the BLDC motor and manage other vehicle systems is increasingly important.

As the topic addressed in this paper is expansive, the *Methodi Ordinatio* technique was applied because it is a simple and effective bibliometric analysis, reducing the search time for articles related to the main research topic (PAGANI; KOVALESKI; RESENDE, 2015).

The *Methodi Ordinatio* consists of nine steps:

- Defining the research intent;
- Preliminary exploratory research in bibliographic databases;
- Definition of Keywords;
- Search and Collection of articles in the databases;
- Filtering;
- Identification of Impact Factor and Number of citations;
- Ordering the scientific relevance of articles using the *InOrdinatio* function;
- Articles download;
- Reading.

It was decided to use three databases to conduct the searches, Scopus, Science Direct, and IEEE. The keywords used were: BLDC Motor, PID Controller, Fuzzy Logic, PSO, Electric Vehicle, Gaussian, GAPID, and DC Motor. However, it was found that it was not possible to identify an article that involved all keywords at once, so we're used combinations like:

- BLDC Motor, PID Controller, Fuzzy Logic and PSO;
- PID Controller, Fuzzy Logic, PSO, Gaussian and GAPID;
- BLDC Motor, PID Controller, Fuzzy Logic, PSO, Electric Vehicle and DC Motor.

Thus, 264 articles were obtained, of which 51 were repeated, and 73 agreed with the theme. After passing through these two filters, the *InOrdinatio* function (Equation 1) was applied, resulting in 36 articles that were studied (Figure 1).

$$InOrdinatio = \left(\frac{F_i}{1000}\right) + \alpha.(10 - (Y_{T_D} - Y_{P_B}) + \sum C_i) \quad (1)$$

Where:

F_i represents Impact Factor;

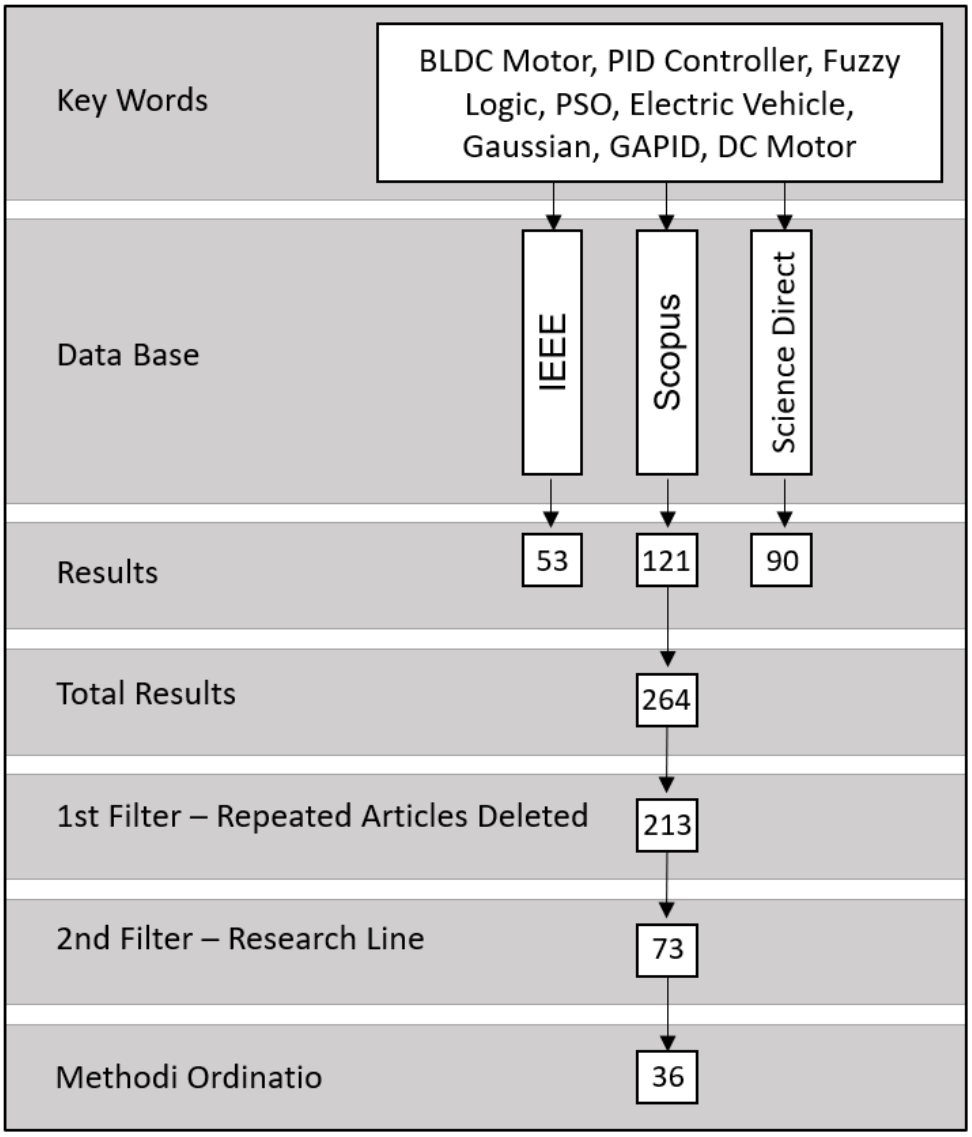
C_i represents Number of citations;

Y_{T_D} represents Year to Date;

Y_{P_B} represents Year Published;

In addition to the articles researched, the books of the principal authors on the subjects that make up the theme were used to enrich the paper.

Figure 1 – Article filtering process



Source: Own authorship (2022)

As shown in Frame 1, it is possible to analyze that none of the papers involved the comparison between a PID controller, a Hybrid Fuzzy-PID controller and a GAPID all implemented in an ESP32 microcontroller, corroborating the academic relevance of this research.

1.2 Main objective and Specific objectives

1.2.1 Main Objective

Recently, a growing number of studies have been carried out with the purpose of presenting different alternatives for control techniques applied to electric vehicles. This

also occurs because electric vehicles have several topologies, such as in-wheel direct drive, where each front or rear wheel has a BLDC motor attached. In this topology, it is extremely important to develop a controller that can adapt to different types of situations, such as performing curves in which the outer wheel has a rotating speed higher than the inner wheel or load variations that may occur due to sudden acceleration and braking (CHHLONH; RIAWAN; SURYOATMOJO, 2019).

Thus, the main objective of this work is to compare different control techniques, such as PID, the hybrid Fuzzy-PID and GAPID, and optimize using the bio-inspired adaptive technique, Particle Swarm Optimization, for the speed control of a BLDC motor.

1.2.2 Specific Objectives

- Conduct a literature review, approaching the main characteristics and differences between the DC electric motor and the BLDC motor and the control techniques;
- Develop an experimental test bench;
- Carry out comparisons between PID, Fuzzy-PID, and GAPID in simulations performed in the software Simulink;
- Develop the PID, Fuzzy-PID, and GAPID hybrid controllers and implement them in the ESP32 microcontroller;
- Optimize the Fuzzy-PID and GAPID hybrid controllers through the PSO, perform the simulation in Simulink and implement them in the ESP32 microcontroller;
- Compare the results obtained in the practical implementation with those of the simulation.

1.3 Dissertation Structure

This dissertation is divided as follows:

Chapter 2 approaches the structures of the DC motor and BLDC motor are described with their respective functions and differences, the fundamentals of control and the details of the PID, Fuzzy controller, Fuzzy-PID hybrid controller, GAPID controller, and the particle swarm optimization algorithm.

In Chapter 3, the reasons for choosing the test bench components, the designed methodology of each control technique and the optimization method adopted, and, finally, MatLab and Simulink simulations.

In Chapter 4, the approach of the simulations of each proposed controller presents the results obtained in the test bench, where tests with and without load were carried out, the results obtained in the simulation and the experimental one, and their comparisons.

Chapter 5 presents the conclusions obtained with the development of this study are presented, besides future studies about the adaptive control.

Frame 1 – Comments on relevant articles.

Title	Comments/Results
(2022) - Metaheuristics-Based Optimization of a Robust GAPID Adaptive Control Applied to a DC Motor-Driven Rotating Beam with Variable Load	• Comparison between Genetic Algorithm (GA) and Particle Swarm Optimization (PSO); • Optimization of the Gaussian Adaptive PID control (GAPID); • Control of a DC motor with load variation; • GAPID presented low overshoot, fast response and robustness to load changes; • Comparison between PID controller and GAPID controller; • Simulation and Implementation results.
(2022) - The performance of the Optimization and Regenerative Braking systems by using PI controlling technique for Electric Vehicle (EV)	• Regenerative Braking systems; • Closed loop feedback control system using PI controller technique; • Simulink model with load; • Importance of BLDC motor for electric vehicles
(2022) - Optimizing BLDC motor drive performance using particle swarm algorithm-tuned fuzzy logic controller	• Performs the optimization of a Fuzzy PID Hybrid controller through Particle Swarm Optimization (PSO); • Uses 3 performance indicators, integral time absolute error (ITAE), integral time square error (ITSE) and integral square error (ISE); • Addresses advantages of Particle Swarm Optimization and Fuzzy PID Hybrid controller; • Examples of rule base; • The controller optimized by PSO obtained good results for load variation; • Simulation only and no implementation.
(2022) - The Fuzzy PID Controller Performance in BLDC Motor Rotor Speed Variable	• Highlights advantages and disadvantages of the PID controller for controlling the speed of a BLDC motor; • Fuzzy-PID Hybrid Controller Applications; • Performs the comparison between the Hybrid Fuzzy-PID Controller and PID Controller for the speed; control of a BLDC motor; • The hybrid Fuzzy PID controller achieved good results, including a 100% overshoot attenuation; • Simulation only and no implementation.
(2022) - A Novel Design Methodology and Numerical Simulation of BLDC Motor for Power Loss Reduction	• Focus on Efficiency; • Load and Speed variation; • The Fuzzy Controller presented better performance regarding ripple torque reduction and good speed; • Simulation and implementation.

(2021) - Four In-Wheel BLDC Motors Speed Control in EV Based on Hybrid Fuzzy-PI Controller Visual on GUI	• Hybrid Fuzzy-PI Controller for BLDC Motors Speed Control; • Comparison between PI controller, Fuzzy controller and hybrid fuzzy-PI controller; • In-wheel direct drive for the Electric Vehicles (EV); • the hybrid fuzzy-PI controller is the most suitable for EVs; • Simulation only and no implementation (future work).
(2019) - Modeling and Simulation of Independent Speed Steering Control for Front In-wheel in EV Using BLDC Motor in MATLAB GUI	• Speed of each front wheel controlled by Fuzzy controller; • Setup of the Fuzzy controller empirically; • Fuzzy Logic is able to handle nonlinearities and uncertainties without need of mathematical model; • Rule Base example; • The Fuzzy Controller presented good results, no overshoot and small error; • Simulation only and no implementation (future work).
(2018) - Brushless DC motor tracking control using self-tuning fuzzy PID control and model reference adaptive control	• Hybrid Fuzzy-PID Controller for Speed Control of BLDC Motor; • Advantages of BLDC motor; • Rule Base example; • Sudden disturbance and parameters variations; • Comparison between Hybrid Fuzzy-PID Controller and Model Reference Adaptive Control (MRAC) with PID compensator; • MRAC performed better than Hybrid Fuzzy-PID Controller; • Simulation only and no implementation.

Source: Own authorship (2022)

2 THEORETICAL BACKGROUND

2.1 DC Motor

Due to their versatility, direct current motors are present in almost every aspect of our life, indirectly or directly, from a complex robotic arm for assembly in factory production that requires precision and speed to a simple laptop cooling system. This fact occurs because, in the DC motor, it is possible to choose several alternatives for the method of excitation of the field windings, which influences the dynamic behavior of the machine and, consequently, the steady-state characteristics. In other words, depending on the adopted topology, it is possible to obtain several options for torque versus speed ratio (FITGERALD; KINGSLEY JR; UMANS, 2008).

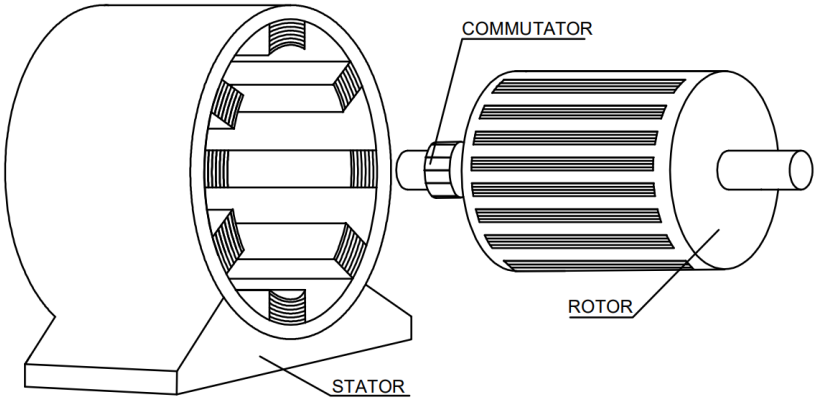
Among the alternatives, the field winding can be connected in series, parallel, or series and parallel to the armature.In addition, it is also possible to choose how the rotor is powered. The current is applied to the brush's rotor in brushed DC motors. In a brushless DC motor, the rotor has a permanent magnet.

2.1.1 Brushed DC Motor Structure

The direct current (DC) motor comprises four elements (Figure 2): the stator, the rotor, the split ring commutator, and the brushes. The stator is a ferromagnetic material surrounded by a set of turns made of copper, where a magnetomotive force is produced. The rotor, whose objective is to allow the passage of the magnetic flux produced by the stator, is an electromagnet where the mechanical motor action occurs in which it consists of a core, usually of silicon steel, by a set of coils connected to the split ring commutator, which in turn transmits current electrical power at the moment its terminals come into contact with the brushes (KOSOW, 1993). In this switching process, sparks are generated between the split ring commutator and brushes, in addition to detrition on both components, thus generating the need for frequent maintenance (DEL TORO, 1994).

Two ways to excite the DC motor are energizing the field coils via a separate DC source or having the current flow through the field winding supplied by the motor. However, for this to occur, there must be residual magnetism in the machine, so the self-

Figure 2 – Brushed DC Motor Structure.



Source: Own authorship (2022)

excitation process can begin (FITGERALD; KINGSLEY JR; UMANS, 2008). Self-excited motors can be classified into:

- Shunt: Field and armature windings are connected in parallel. The speed of a Shunt DC motor is constant and does not deviate with varying mechanical loads.
- Series: Field and armature windings are connected to the power supply. Series motors always rotate in the same direction, regardless of the voltage source. Its speed varies with mechanical load.
- Compound: divided into cumulative or additive and differential or subtractive. The cumulative motor unites the characteristics of shunt and series motors, but for variation in speed with a load of the relative number of ampere-turns in the shunt and series fields.

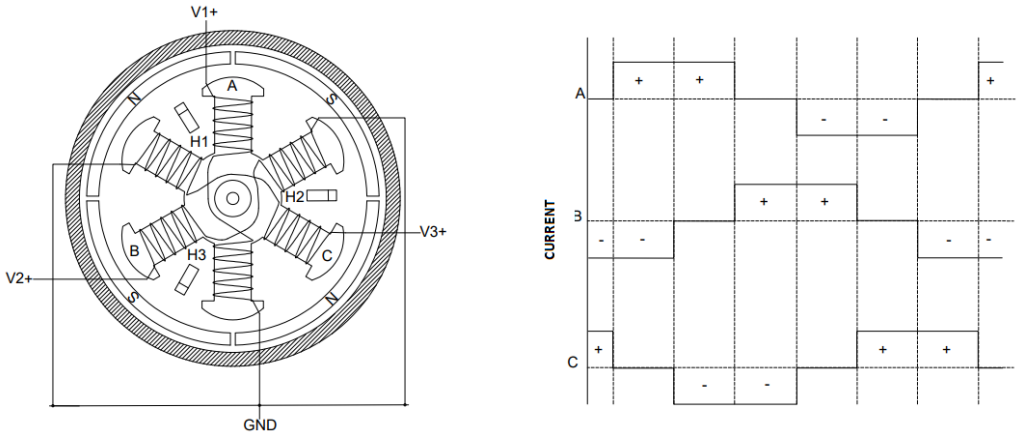
2.1.2 BLDC Motor Structure

The brushless direct current motor (BLDC) comprises a three-phase stator and a rotor with permanent magnets on its surface. Its operating principle is similar to that of a synchronous alternating current motor, but its power supply is of direct current (EL-SAMAHY; SHAMSELDIN, 2018).

One of the alternatives to increase the motor's efficiency is to connect two opposite coils in series, doubling the force of attraction and repulsion. In this way, six intervals are required for the rotor to perform a complete rotation, according to Figure 3.

Note that at the same instant, the current is positive at point A, negative at point

Figure 3 – BLDC Motor and Hall sensor pulses.



Source: Own authorship (2022)

B, and zero at point C. Thus, it is possible to use the same current to energize two different phases simultaneously, making a star connection (Y).

As stated before, at least six intervals are necessary for the motor to complete a revolution. For this phenomenon to occur, one of the solutions would be the use of an ESC, electronic speed controller, which uses an arrangement of transistors that work synchronously, driving the coils that must attract and repel the rotor according to its position, which can be obtained by the use of Hall sensors or the use of counter-electromotive force (NARMADA; AROUNASSALAME, 2014).

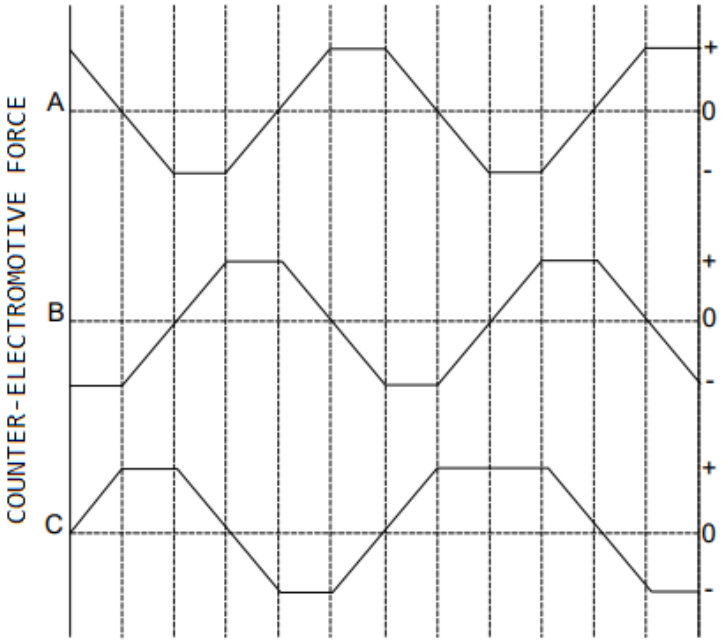
In the Hall sensors method, the sensors are usually arranged every 120° or every 60° so that when the magnetic field coming from the rotor approaches a sensor, it emits a high logic level signal for one pole and a low logic level signal for the opposite pole as shown in Figure 3.

A current in the opposite direction is generated in the coils that are not energized at a given moment through the counter-electromotive force. As a consequence, an induced voltage is generated, identified by the controller present in the ESC, which performs calculations to predict which coils it must turn on or off, as shown in Figure 4.

2.1.3 Nonlinear Phenomena in BLDC motor

Currently, several studies have been carried out with the purpose of understanding the nonlinear characteristics of the BLDC motor, many of them are based on theoretical analyses instead of experimental, because in order to reach and observe the

Figure 4 – Counter-Electromotive Force Signal.



Source: Own authorship (2022)

chaotic state, which is the manifestation of nonlinearities and which occurs in an instant of time, the motor needs to be operating under specific conditions. In some papers, the chaotic state was caused by the internal characteristics of the motor and load variations (LI *et al.*, 2018).

Adaptive control techniques such as the Fuzzy controller are ideal for non-linear systems. In the case of Fuzzy controller, which uses the Universal Approximation Theorem and, when well-designed, fits for most cases where nonlinearities occur. With this technique, it is possible to maintain an acceptable level of control system performance when large and unknown changes in model parameters happen (WANG, 1993).

2.1.4 DC Motor X BLDC Motor

The brushed direct current motor has undesirable effects such as the projection of sparks and carbon particles coming from it and the generation of acoustic noise (VARGHESE; ROY; THIRUNAVUKKARASU, 2014).

Despite its limited reliability linked to the need for constant maintenance of its brushes due to operating wear and the need for commutators, the DC motor still has advantages, such as good efficiency and linear behavior, discarding the need to use techniques of complex control (ARIS *et al.*, 2016).

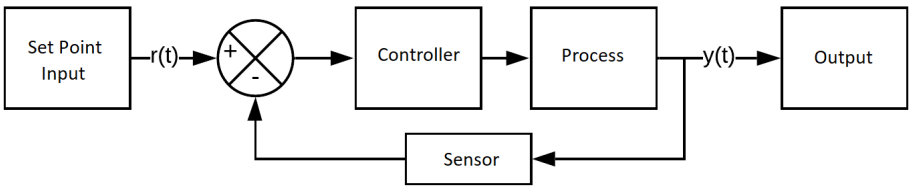
Unlike DC motors, BLDC motors have many advantages: long operating life, high dynamic response, high efficiency, better speed vs. torque characteristics, more comprehensive speed range, and higher torque-to-weight ratio (EL-SAMAHY; SHAMSELDIN, 2018).

This way, the BLDC motor requires reduced maintenance and maintains the same power as a DC motor occupying a smaller volume. These and other characteristics mentioned above make the replacement of DC motors with BLDC motors more attractive and used in several industrial applications (KUMPANYA; THAIPARNAT; PUANGDOWNREONG, 2015).

2.2 PID Controller

Controllers are subsystems that act on a given system or plant to achieve preestablished results. When correctly sized, controllers can increase the efficiency of the system in which they operate. When used in a closed-loop (Figure 5), where the system feedback with the output signal occurs, the reduction of the output error to the input signal can occur (OGATA, 2011).

Figure 5 – Closed-loop Control System.



Source: Own authorship (2022)

Because it is simple, robust, and has few adjustment parameters, 90% of industrial loops apply PID controllers (CHOPRA; SINGLA; DEWAN, 2014).

According to (NISE; SILVA, 2002), the control signal provided by the PID controller depends on three parameters, which are given by Equation 2:

$$u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt} \quad (2)$$

Where:

$u(t)$ represents the control signal;

$e(t)$represents the Steady-state error, which is the difference between the Set-point $r(t)$ and the Output $y(t)$;

K_p represents the Proportional gain;

K_i represents the Integral gain;

K_d represents the Derivative gain.

Each parameter, K_p, K_d e K_i, allows changing the controller's behavior applied to the plant. For example, high gains can make the controller act with fast changes in the output signal. On the other hand, low gains result in a controller with a more passive characteristic, having little influence on the system (OGATA, 2011).

Knowledge of each parameter's influence on the system's performance, such as the Percentage Overshoot (PO), which represents how much the maximum peak value exceeds the final value, is essential. So that when it is necessary to make more precise adjustments, the same can operate as expected. In this way, according to Frame 2, it is possible to verify the effect caused in the plant by the variation of the gain of each controller parameter (KUMAR; SWAIN; NEOGI, 2017).

Frame 2 – Controller x Plant Ratio.

Parameter	Settling Time	Overshoot	Error
Proportional (K_p)	Small Change	Increase	Decrease
Integral (K_i)	Increase	Increase	Zero
Derivative (K_d)	Decrease	Decrease	Small Change

Source: Kumar, Swain, and Neogi (2017)

2.3 Fuzzy Logic

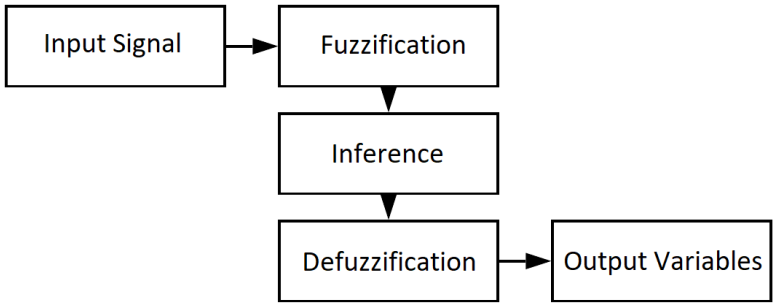
To assist in the search for the solution to specific problems, fuzzy logic seeks to approach human thought, leaving boolean logic and seeking answers by dealing with the concept of partial truth. It happens because the basis of fuzzy systems is the Fuzzy set theory, in which its elements have a degree of membership determined by the analysis of the membership functions (DE AZEVEDO; BRASIL; OLIVEIRA, 2000).

One of the main advantages of using the Fuzzy controller is that the designer can control a system based on its behavior without raising its mathematical model (FENG *et al.*, 2002). However, for this to occur, the designer must have complete knowledge of

the system's functioning to adjust the parameters correctly, making the control effort to handle each input correctly to achieve the expected objectives.

For the processing of numerical variables sent to the controller to happen, for example, to process a signal sent by a sensor, it is necessary to start a process that consists of transforming this numerical information into linguistic variables to carry out decision-making based on pre-established rules associated with a numerical value necessary to control the plant. These steps can be defined as fuzzification, inference, and defuzzification, as shown in Figure 6 (CHOI *et al.*, 2005).

Figure 6 – Fuzzy logic Structure.



Source: Own authorship (2022)

The fuzzification step transforms the input data, numerical variables, into linguistic variables, where a pre-processing of categories takes place to reduce the number of processes. Then, in inference, decisions are made based on the If-Then conditional, defined by a base of previously established rules, defining the actions to be taken on a given occasion. The last step of signal processing, defuzzification, ensures the exact interpretation of the linguistic variables obtained in the inference phase into numerical values (CHOI *et al.*, 2005).

Two models are usually used in Fuzzy systems, the classical and the interpolation ones. The classics have for each rule a fuzzy term within a fixed set of convex terms that can be graphically represented by triangular, trapezoidal, and bell functions. Among the most common models, the Larsen model and the Mamdani model are worth mentioning (SILVA; DATTA; BHATTACHARYYA, 2002).

On the other hand, interpolation models usually present a different conclusion for each rule through a strictly monotonic function, the most common being the Takagi-Sugeno model and the Tsukamoto model. In these models, the values obtained by

each rule for each control variable are unique, where a global control action is obtained through a weighted average of the individual values obtained (JÚNIOR *et al.*, 2005).

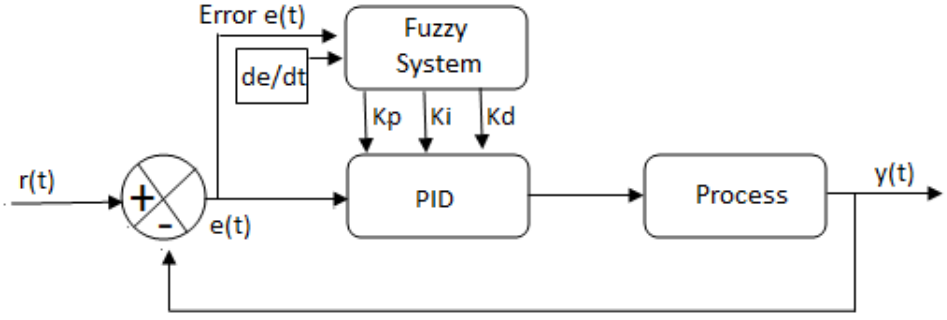
In research performed with the purpose of optimizing the BLDC motor drive performance, the PID controller produced hysteresis effects and created uncertainty problems and to avoid such situations the Fuzzy controller was implemented. After this application, a considerable improvement in the control of the system was noticed, reaching favorable results (SHI *et al.*, 2022). In another example, which the purpose was to seek a power loss reduction of a BLDC motor, it was noted that the Fuzzy controller contributed to a better control adaptability for load variations, reducing the ripple and the system response time (KUMAR; CHANDRASEKARAN, *et al.*, 2022).

2.4 Fuzzy-PID Hybrid Controller

Despite being robust and easy to apply, the PID controller has some limitations in specific applications, such as in systems that do not behave linearly or in situations where the dynamics of the plant vary constantly. Such situations can impact the response time of the controller. One of the alternatives for such a situation would be the union of the PID controller and the Fuzzy logic.

In this controller, the application of Fuzzy logic is coupled to a PID controller to adjust its parameters automatically in an online process (Figure 7). In other words, if there are changes in the plant dynamics, for example, load variations, the PID controller gains are adjusted through Fuzzy logic to adapt to this change in real-time (GOSWAMI; JOSHI, 2018).

Figure 7 – Basic structure of a Fuzzy-PID hybrid controller.



Source: Own authorship (2022)

This union can stabilize the system faster than a classical PID controller, de-

creasing the accommodation time to reach the steady-state (GEETHA; THANGAVEL, 2016).

Several researches based on the application of this controller in electric vehicles were carried out and demonstrated that this type of control technique, when applied to control the speed of a BLDC motor, presents greater reliability when compared to PID or Fuzzy controller (CHHLONH; KIM, *et al.*, 2021).

Another study has indicated that the hybrid fuzzy PID controller was able to decrease the rise time, the settling time and even attenuate the overshoot to 100% (SUTARNA; PURWANTI; SUHADHA, 2022).

2.5 GAPID Controller

Gaussian adaptive control is based on the use of Gaussian functions with well-defined upper and lower limits, adjustable concavities, and smooth derivatives, which causes the gains to be gradually increased or decreased as the stationary error approaches zero, avoiding an unexpected system behavior which might be caused by discontinuous gain transitions (BORGES *et al.*, 2022).

These functions are defined as shown in Equation 3.

$$f(\gamma) = K_1 - (K_1 - K_0)e^{-q.\gamma^2} \tag{3}$$

Where:

γ represents the error;

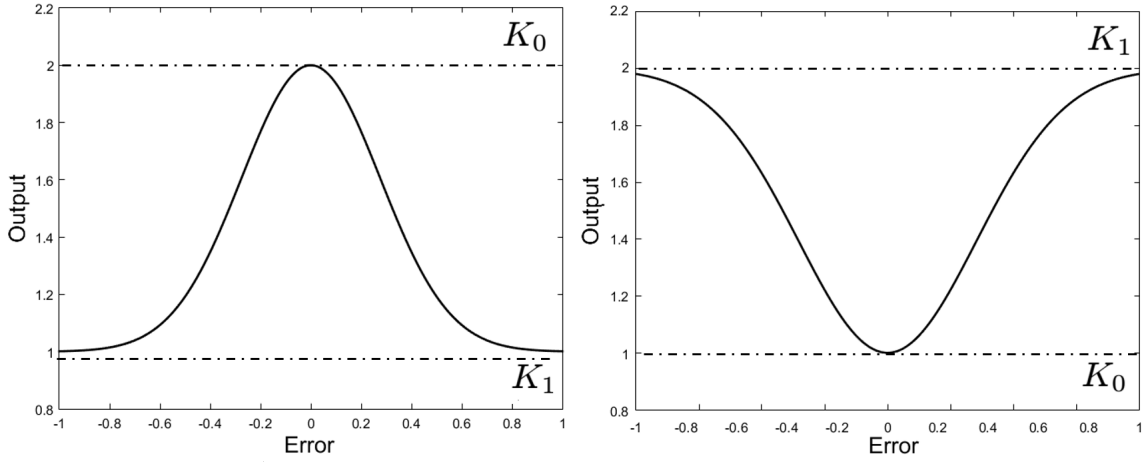
K_1 and K_0 represent the function limits;

$-q$ represents the regulator of the concavity of the Gaussian curve.

The $-q$ regulator, represented by Equation 4, adjusts the position of the range of input values where the transition between K_0 and K_1 occurs. K_0 has a more significant influence when the stationary error approaches zero and K_1 when the error is significant. Note also that when $K_0 < K_1$, the concavity is facing upwards, and when $K_0 > K_1$, the concavity is facing downwards as represented by Graph 1.

$$q = -ln\left(\frac{K_1 - f(\gamma)}{K_1 - K_0}\right).\frac{1}{\gamma^2} \tag{4}$$

Graph 1 – Gaussian functions.



Source: Own authorship (2022)

As the controller is based on a linear PID controller, each branch has its Gaussian function with its respective values. That means there would be nine parameters which are K_{p_1}, K_{i_1}, K_{d_1}, K_{p_0}, K_{i_0}, K_{d_0}, q_p, q_i and q_d represented by Equation 5.

$$\begin{aligned} fK_{p_1} &= K_{p_1} - (K_{p_1} - K_{p_0})e^{-q_p \cdot \gamma^2}; \\ fK_{i_1} &= K_{i_1} - (K_{i_1} - K_{i_0})e^{-q_i \cdot \gamma^2}; \\ fK_{d_1} &= K_{d_1} - (K_{d_1} - K_{d_0})e^{-q_d \cdot \gamma^2}. \end{aligned} \tag{5}$$

Generally, the Gaussian curve for the proportional gain is turned upwards, since its objective is to accelerate the system's transient response. In other words, in the first instant T, the maximum stationary error demands a high gain value. In contrast, the integral gain curve is assumed to have opposite behavior, in other words, a downward concavity. As the controller approaches the integral gain set point, it should increase to the point to correct this margin of error and keep the system stable.

This type of controller, when correctly parameterized, proved to be a good control alternative for non-linear systems, supplying the deficiencies of the widely used controller, the PID controller (BORGES *et al.*, 2022).

2.6 Particle swarm optimization

Particle swarm optimization (PSO) was initially proposed in 1995 by James Kennedy and Russell Eberhart, in which they sought to describe the collective behavior of

groups of animals in a mathematical way, where the individual behavior of each member that composes the group is analyzed and the social impact it has on its neighbors (KENNEDY; EBERHART, 1995).

As it is based on biological models, the algorithm uses basic rules to generate competitive and/or cooperative behavior among individuals to find the best solution for a given problem (GARCIA-GONZALO; FERNANDEZ-MARTINEZ, 2012).

Over time, several researchers sought ways to increase the performance of the PSO. Among the improvements, it is worth highlighting the stability analysis and the understanding of the dynamics of the particle swarm, which is based on the group's influence on the particle.

This cultural adaptation can be summarized in three principles: The individual and collective perception of the particle, the comparison between individuals, and the imitation of the best particles (EBERHART; SHI; KENNEDY, 2001).

In this way, because it is simple and robust, this method has been successfully applied in several areas of engineering to find solutions to various problems (GARCIA-GONZALO; FERNANDEZ-MARTINEZ, 2012).

2.6.1 Optimization through PSO

To optimize a problem, it is necessary to initialize a randomly distributed population composed of individuals or particles with unique positions x_i and speeds v_i. These particles are represented by a vector whose dimension is the domain of the fitness function that is evaluated by each particle in each iteration, resulting in a change in the position and velocity of each individual. The number of iterations and the number of individuals influence the amount of processing used and the accuracy of the result obtained.

Therefore, the decision taken by a given individual is linked to its performance in the past, together with the performance of its neighbors.

In this way, for a better understanding, the optimization process can be separated into seven steps:

1. Generating an initial population where each one has a position x_i which is the value that the particle has at the moment, and a velocity v_i which is the value that changes the individual's response at each iteration;

2. Calculating the fitness function returning as a result of how far each particle is from the target;
3. Finding the (*pBest*) which is the best position so far of each particle;
4. Finding the (*gBest*) which is the comparison between the particles, whose objective is to find the best particle in the population at the moment;
5. Updating the velocity of each particle, whose equation can be represented by Equation 6, where r_1 and r_2 are random values between 0 and 1 and c_1 and c_2 arbitrary values between 0 and 4. In this step, it can add a variable that can be decisive in the stability condition of the algorithm, the inertia factor (ω), whose purpose is to carry out an exploratory phase in a moment of execution. As the iterations go by, its value decreases, reaching the specialization part, in other words, trying to find a balance between local and global skills of the PSO;
6. Determining the position of each particle as shown in Equation 7;
7. Evaluating each particle by finding the pBest and gBest.

$$V_i(t+1) = \omega v_i + c_1 r_1 (pBest - x_i) + c_2 r_2 (gBest - x_i) \tag{6}$$

$$x_i(t+1) = x_i(t) + V_i(t+1) \tag{7}$$

Where:

ω represents the inertia weight;

v_i represents the previous velocity of the particle;

c_1 represents the cognitive parameter;

r_1 represents a random number;

$pBest$ represents the local best position;

c_2 represents the social scaling parameter;

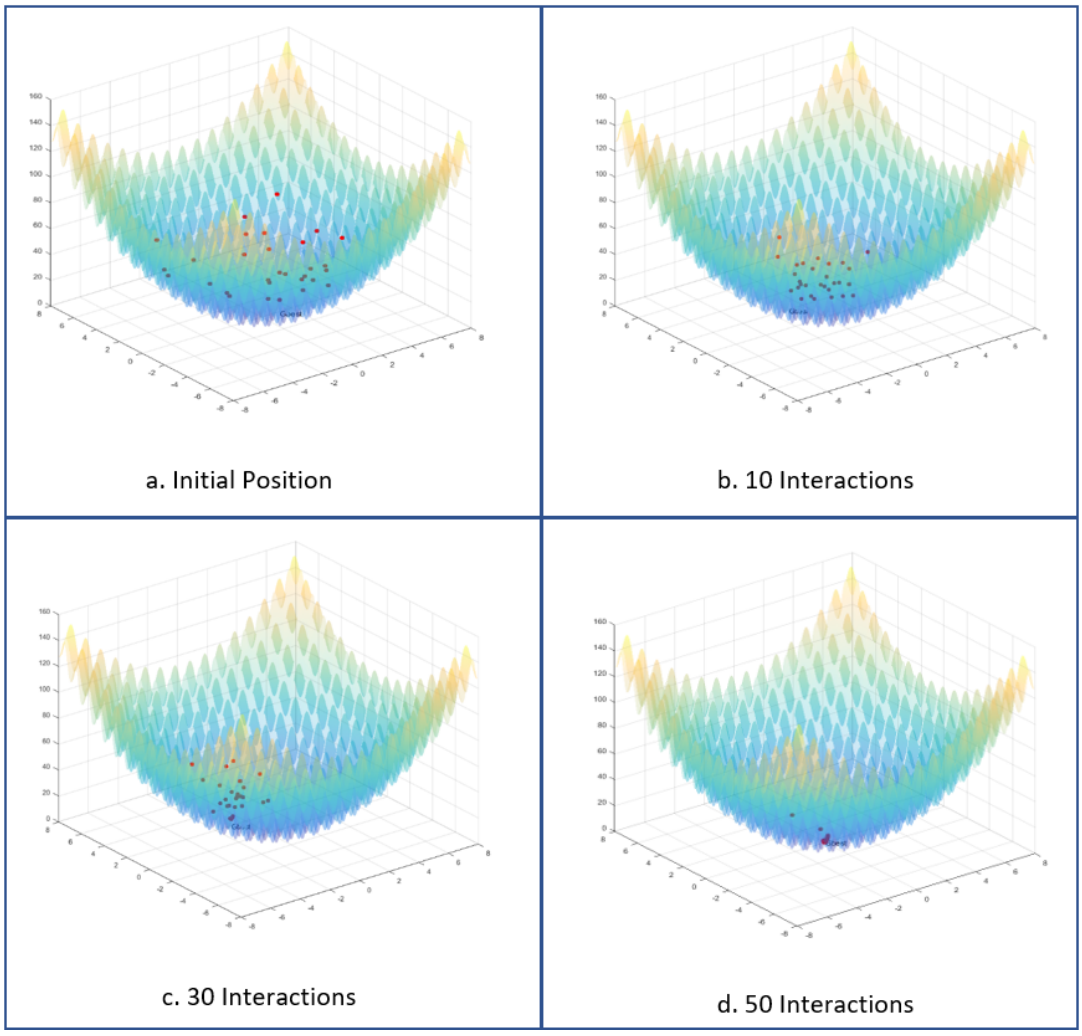
r_2 represents a random number;

$gBest$ represents the global best position;

x_i represents the previous position of the particle.

For better understanding, Figure 8 illustrates the behavior of particles at each iteration to solve a problem in which 30 particles were generated with 50 iterations.

Figure 8 – Arrangement of PSO particles.



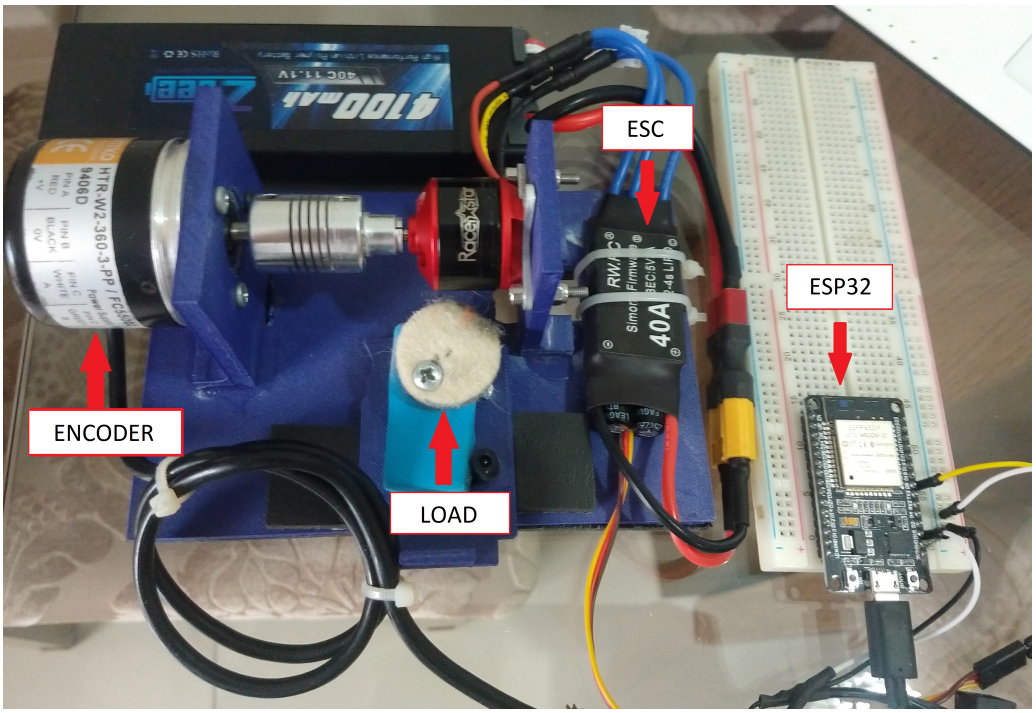
Source: Own authorship (2022)

3 THEORETICAL AND EXPERIMENTAL DEVELOPMENT

For the development of the control system, it was decided to use the BLDC Racerstar BR2212 1800KV motor, a 40 amps electronic speed controller (ESC), the HTR-W2-360-3PP encoder to close the control loop that emits 360 pulses per revolution, and an ESP32 microcontroller to control the motor and monitor these pulses.

In order to produce the load variation necessary in the experimental tests, a felt disk was directed to the rotor, as shown in Figure 9:

Figure 9 – Test Bench.



Source: Own authorship (2022)

3.1 mathematical model of the system

Before starting the design and development of controllers, obtaining a mathematical model representing the behavior of the system in a real environment is necessary. Furthermore, obtaining all the values of the constructive parameters related to the BLDC motor is not a simple task, it requires several tests with specific equipment.The easiest way is to represent the system as a black box. Therefore, it is necessary to carry out experimental tests with a mathematical representation closest to reality can be raised (NISE; SILVA, 2002).

to obtain the steady-state speed and, consequently, the maximum number of pulses emitted by the encoder in the sampling time T_s.

Five measurements were carried out to find the adequate sampling time for the system, where the motor rotates at maximum speed every thirty seconds for each period, being: 1 s, 500 ms, 275 ms, 200 ms, 100 ms, and 50 ms. Moreover, arithmetic means with their respective resolutions $R\%$ were taken from these (Equation 8), resulting in the following Table 1.

$$R\% = 100\frac{1}{Pulses} \tag{8}$$

Table 1 – Maximum pulse count and resolution per sampling time.

T_s **(ms)**	**Pulse Average/**T_s	**Resolution %**
1000	121000	0.000826
500	60500	0.001653
275	33275	0.003005
200	24200	0.004132
100	12100	0.008264
50	**6050**	**0.016529**

Source:Own Authorship (2022).

The following calculations were performed to validate the data obtained in Table 1 and to verify if the speed obtained through the encoder portrays the reality of the motor manufacturer.

Knowing that the encoder emits 360 pulses at each rotation, and 121000 pulses were obtained in a period T_s of 1000 ms, we have Equation 9.

$$\text{Revolutions} = \frac{121000}{360} \approx 336 \tag{9}$$

Converting to RPM:

$$\text{Motor Speed} = 336 \cdot 60\,\text{seconds} \approx 20167\,\text{rpm} \tag{10}$$

In this way, as presented by the manufacturer in the BLDC Racerstar BR2212 motor datasheet in Table 2, for each volt applied, the motor rotates at 1800 rpm. In other words, at its nominal voltage of 11.1 volts, the maximum speed is approximately 20000 rpm.

Table 2 – Racerstar BR2212 BLDC motor datasheet.

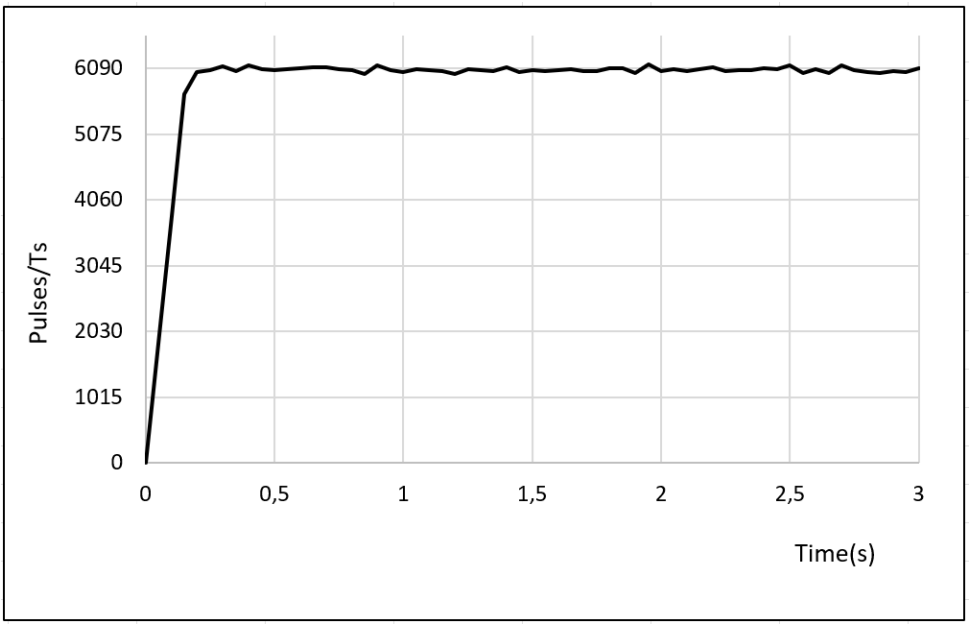
Description:	
Brand Name:	Racerstar
Item Name:	BR2212 brushless motor
KV:	1800
Operating Voltage	2-4S
Weight:	52g
Recommended Prop:	8060

Source:Manufacturer Website (2022).

As open-loop tests confirmed data provided by the manufacturer and based on experimentally obtained results (Table 1), a period of 50 ms was adopted because it has the best resolution, which is 0.016529% and 6050 pulses per T_s.

That way, the BLDC motor open-loop speed curve is obtained (Graph 2), showing that it behaves like a first-order system. Thus, the necessary parameters to be determined are the static gain A and the time constant τ values.

Graph 2 – Motor BLDC open-loop response curve - T_s = 50 ms.



Source: Own authorship (2022)

Knowing that first-order systems can have their time constant measured at the moment the system response takes to reach 63% of its maximum speed (NISE; SILVA, 2002). It is possible to go along with the steps below. 63% of maximum speed is approximately 3812 pulses, as described in Equation 11.

$$\text{Motor Speed} = 6050 \cdot 0.63 \quad \approx \quad 3812\,\text{pulses} \tag{11}$$

According to Graph 2, 3812 pulses represent a τ value between 0.05 and 0.1 seconds. The period of 0.05 seconds was chosen because it approximates the motor response. As a result, the curve was obtained by replacing the values of A and τ in Equation 12, as shown in Graph 3.

$$y(t) = A(1 - e^{\frac{-t}{\tau}}) \tag{12}$$

Where:

$y(t)$: Motor speed at the instant t;

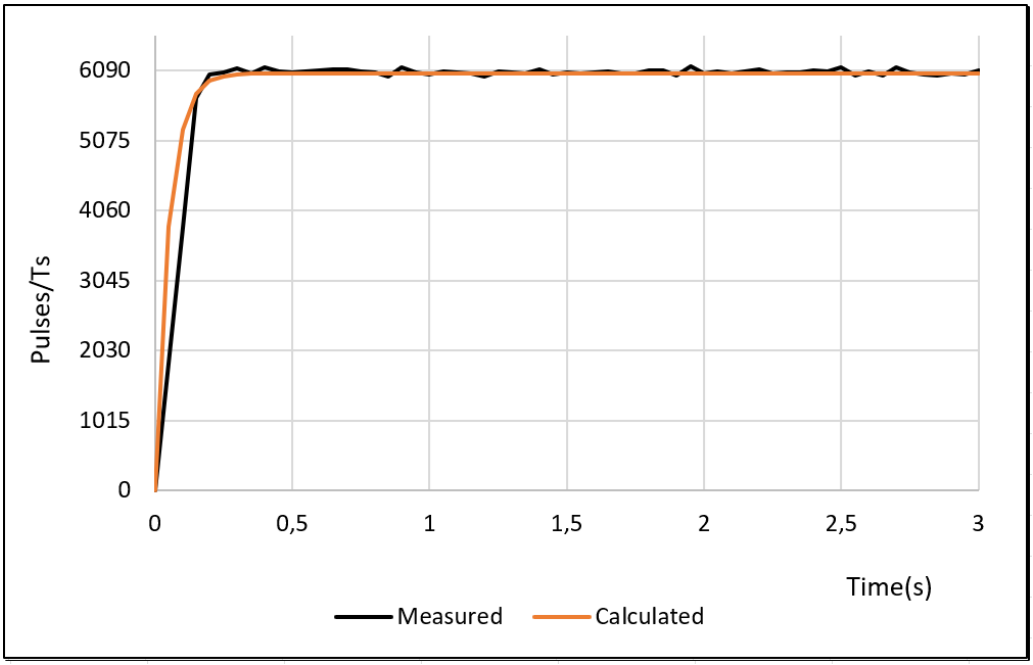
τ: Time constant;

A: Static gain.

Thus,

$$y(t) = 6050(1 - e^{\frac{-t}{0.05}}) \tag{13}$$

Graph 3 – Motor BLDC open-loop response curve measured vs calculated - T_s = 50 ms.



Source: Own authorship (2022)

Therefore, the mathematical model of the BLDC motor or its plant transfer function can be given by Equation 14.

$$G(s) = \frac{6050}{0.05s + 1} \tag{14}$$

The Equation 14 was obtained from a step input of 11.1 volts, because of it, must divide the equation by the step, getting the Equation 15.

$$G(s) = \frac{6050}{0.555s + 11.1} \tag{15}$$

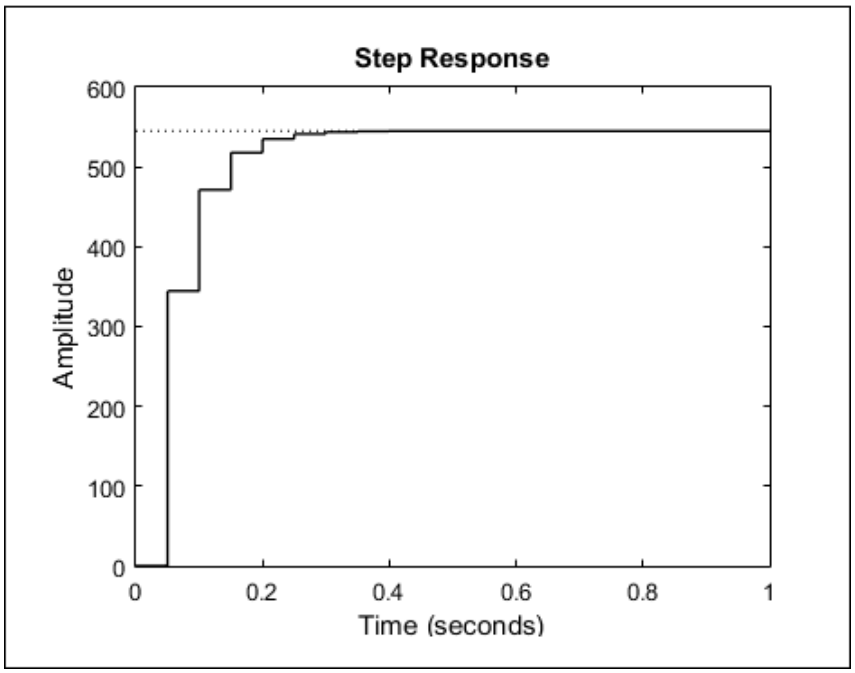
3.2 Plant discretization

As the system control is embedded in a microcontroller which is composed of a discrete control system, it is necessary to carry out the controller design in the discrete domain (Z-Plane), where the model is obtained through the discretization method Zero-Order Hold (ZOH), which is the most suitable model for digital systems.

The software Matlab was used to perform the discretization of the plant, resulting in Equation 16, which is represented by Graph 4 for a unit step response.

$$P(z) = \frac{344.5}{z - 0.3679} \tag{16}$$

Graph 4 – Open-loop discrete plant unit step response.



Source: Own authorship (2022)

3.3 Controllers design

3.3.1 Proportional-integral controller design (PI).

In order to control the speed of the BLDC motor, it was decided to design a PI controller without the need to use a derivative gain (K_d) because it is a first-order system. Thus, with the integral gain (K_i), it is possible to correct the stationary error and, with the proportional gain (K_p), obtain an improvement in the response speed.

In this way, the performance parameters adopted for the development of the controller were the percentage overshoot (PO) of at most 1.3% and a stabilization time (T_e) of 0.5 seconds. With this information, it is possible to obtain the damping ratio (ζ) as represented by Equation 17.

$$\zeta = \frac{log(\frac{100}{PO})}{\sqrt{\pi^2 + log(\frac{100}{PO})^2}} = 0.5147 \tag{17}$$

In this way, it is possible to calculate the natural frequency (ω_n) according to Equation 18 and the damped natural frequency (ω_d) represented in Equation 19.

$$\omega_n = \frac{4}{Te\zeta} = 15.5425 \tag{18}$$

$$\omega_d = \omega_n\sqrt{1-\zeta^2} = 13.3255 \tag{19}$$

As the values of the damping coefficient, the natural frequency of the system, and the natural frequency were obtained, it is possible to calculate and find the pole in the S-domain according to Equation 20.

$$s = -\zeta\omega_n \pm j\omega_d = -8 + j13.3255 \tag{20}$$

Transforming Equation 20 to the Z-domain, we have Equation 21.

$$z = e^{sT_s} = 0.5270 + j0.4143 \tag{21}$$

For the stationary error to be null, a pole was adopted at z=1 and a zero at α as represented in Equation 22, where k represents the controller gain.

$$K(z) = k\frac{z-\alpha}{z-1} \tag{22}$$

Consequently, the plant angle (θ_p) is obtained by replacing Equation 21 result in Equation 22. Knowing that the controller angle (θ_k) added to θ_p should result in -π, it is possible to obtain (θ_k), as shown in Equation 23.

$$\theta_k = -\pi - \theta_p = -1.9374 \tag{23}$$

It is known that the controller numerator angle (θ_n) is the result of the sum of the controller angle and the controller denominator angle (NISE; SILVA, 2002). In this way, keeping the controller denominator fixed at (z-1), we can calculate α, as presented in Equation 24.

$$\alpha = -\frac{z.tan(\theta_n)}{tan(\theta_n)} = -0.2594 \tag{24}$$

With the value of α, it remains to calculate the value of k to obtain the controller, according to Equation 25.

$$k = \frac{1}{|P(z)|.|\frac{z-\alpha}{z-1}|} = 0.0009113 \tag{25}$$

Substituting the Equation 24 and Equation 25 in Equation 22, we obtain the controller (Equation 26).

$$K(z) = \frac{0.0009113z + 0.0002364}{z-1} \tag{26}$$

3.3.2 Fuzzy-PI hybrid controller design

As already mentioned, the development of the Fuzzy system was chosen because it is the simplest method to perform adaptive control. However, it is empirical, requiring the designer or specialist to have full knowledge of the operation of the plant to be controlled. Thus, for it to be implemented, some steps need to be carried out: the definition of input and output variables, delimitation of the universe of discourse for each variable, and the definition of the rule base.

3.3.2.1 Definition of input and output variables

To establish the Fuzzy control system, two input variables were used. The first input was the error sign (E) which is the difference between the chosen reference and the output signal, and the second input is the error derivative (de) which indicates the proximity of the output signal with the reference, helping fine-tune the controller. For the output of the Fuzzy control system, the gains of the PI controller are called K_p and K_i.

3.3.2.2 Delimitation of the universe of discourse for each variable

At this stage, the fuzzification process begins, where numerical variables are transformed into linguistic variables. Thus, through tests in simulations, the universe of discourse for each input and output variable, as well as their pertinence functions, were delimited.

As shown in Figure 10, it was chosen for the error inputs and the error derivative to use five membership functions of the triangular type because it is the most accessible topology to manipulate and also because it is the most common in many applications of Fuzzy controllers. The membership functions are named NP, NE, ZO, PO, and PP, meaning negative plus, negative, zero, positive, and positive plus, respectively. The universe of discourse of the variable E varies from -5000 to 5000, and the universe of discourse of the variable de varies from -1200 to 1200.

According to Figure 11, four membership functions of the triangular type were used for the output variables: Z, S, M, and B, meaning zero, small, medium, and big, respectively.

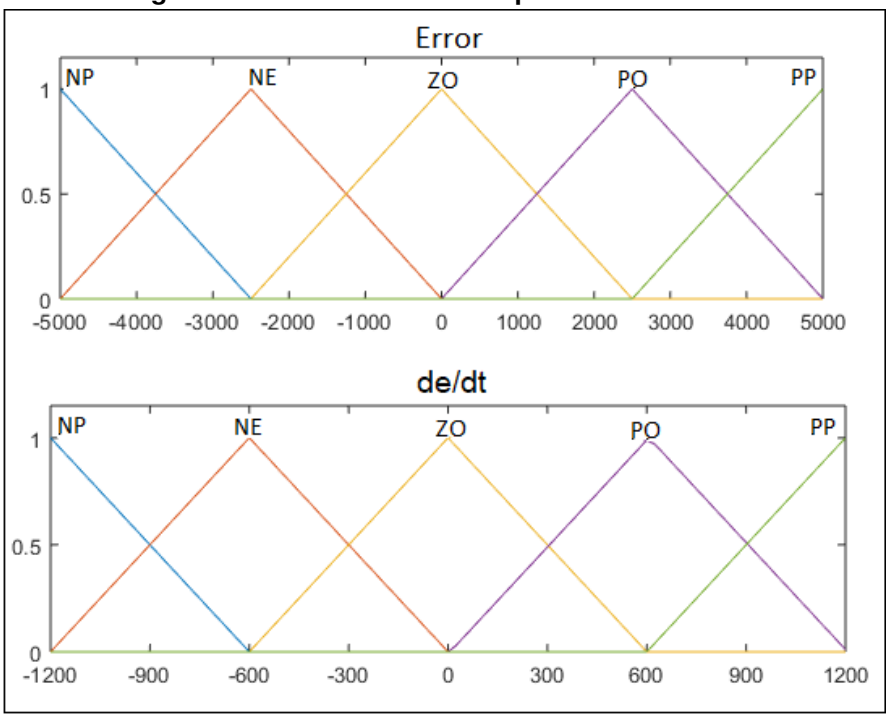
The universe of discourse of the variable K_P is from 0 to 3, and K_i is from 0 to 7.

3.3.2.3 Rule base definition

The rule base strongly influences the control system's behavior, directly influencing performance parameters, such as response time and percentage overshoot. Therefore, in order for the rule base to bring the desired behavior to the control system, some points were taken into account, which are:

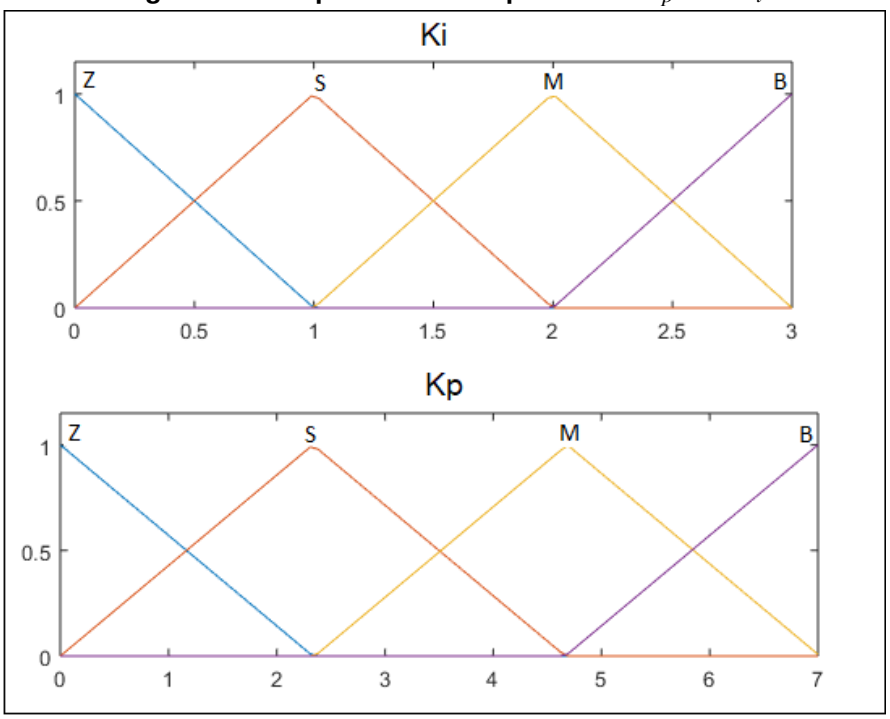
- For a high error value, there must be a high proportional gain, which causes the

Figure 10 – Entries membership function E and de.



Source: Own authorship (2022)

Figure 11 – Output membership function K_p and K_i.



Source: Own authorship (2022)

necessary correction, approaching the pre-established reference value; and a moderate value of the integral gain to avoid a high percentage of overshoot and not have a significant impact on the response time;

- For the situation where the error and derivative values are close to zero, the proportional and integral gain values must be close to zero, maintaining the stability of the system;

Thus, Figure 12 represents the rules established for K_p and K_i respectively.

Figure 12 – Rule base for K_p and K_i.

KP					
E \ de	NP	NE	ZO	PO	PP
NP	B	B	B	B	M
NE	M	B	S	S	S
ZO	M	B	Z	S	B
PO	S	S	S	S	S
PP	M	B	B	M	B

KI					
E \ de	NP	NE	ZO	PO	PP
NP	Z	Z	Z	Z	Z
NE	M	M	M	M	M
ZO	B	B	Z	B	B
PO	S	M	M	M	M
PP	Z	P	B	B	B

Source: Own authorship (2022)

3.4 Optimization of the Fuzzy-PI hybrid controller by PSO

As previously reported, the realization of a Fuzzy controller is done empirically. The particle swarm optimization was applied to avoid hours of work wasted in trial and error tests, finding the best parameters and obtaining better results from this controller.

The PSO, in this case, was used offline, where the values were obtained based on the simulations performed by the Simulink software.

To correctly generate the PSO, the following steps were applied:

1. Generate the initial population of 40 particles with 40 iterations, each particle having its position and velocity generated randomly;
2. Calculating the fitness function, in other words, check how far the particles are from the target/ setpoint;
3. Finding the best position of each particle at each iteration (*pBest*);
4. Finding the best particle in the population at the moment (*gBest*);
5. Updating the velocity of each particle, based on *pBest* and *gBest* obtained;

6. Determining the position of each particle at a given instant;
7. At the end of the 40 iterations, save the *gBest* obtained, compare the 35 simulations that will be performed, and use the best result for the implementation.

This algorithm was developed with the aim of finding the best result, in other words minimizing the steady-state error, for a given setpoint, which in this case is 2900 pulses per sampling period.

As the results obtained by the empirical Fuzzy controller were considerably good, it was decided to apply some restrictions with a focus on improving what has been developed so far.

In this case, the optimization algorithm was limited to generating 25 rules, the maximum range of the inputs E and de between -4000 to 4000 and the range of the outputs K_p between 0 and 7 and K_i between 0 and 15.

It is worth noting that the nomenclature of membership functions remained the same as those used in the Fuzzy controller for better comparison.

During the development process of the algorithm, several simulations were performed, and it was noted that with a number greater than 40 particles and also a number greater than 40 iterations, no significant improvement was obtained, in addition more interactions and particles would require a longer processing time. Thus, both numbers, 40 particles and 40 iterations were considered.

Also, the fifth step is the moment when Equation 6 is applied. In this case, the values for the parameters c_1 and c_2 were empirically chosen as 1 and 2.5, respectively. Therefore, a static inertia coefficient (ω) of 0.5 was used in the tests.

With the simulations, the rule base for each variable had a considerable change, about 87% of the rules were modified as shown by the highlighted blocks in Figure 13. As shown in Figure 14, the membership functions continued to be of the triangular type, but the universe of discourse for each variable changed.

Now E ranges from -3500 to 3500 and de ranges from -800 to 800, according to Figure 14, while K_p ranges from 0 to 3.5 and K_i varies from 0 to 10 as shown in Figure 15.

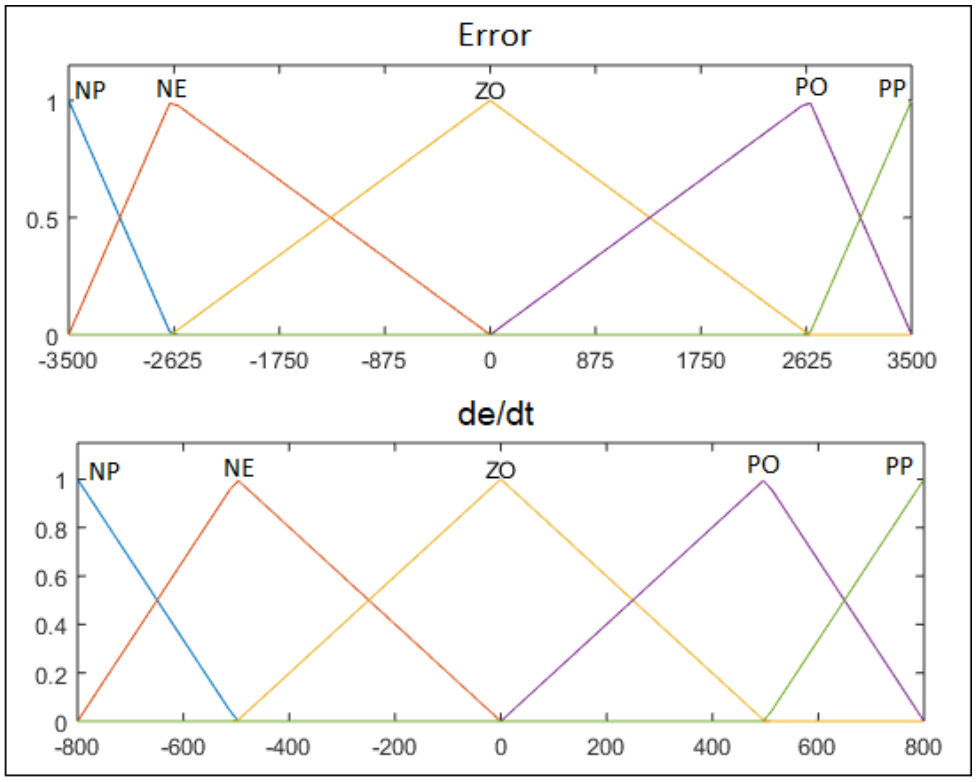
Figure 13 – Rule base for K_p and K_i obtained by the PSO.

KP					
de / E	NP	NE	ZO	PO	PP
NP	B	Z	B	Z	M
NE	B	B	B	Z	B
ZO	S	Z	B	B	M
PO	Z	Z	Z	Z	Z
PP	M	Z	Z	M	Z

KI					
de / E	NP	NE	ZO	PO	PP
NP	B	Z	B	Z	B
NE	B	B	Z	Z	B
ZO	B	Z	B	S	B
PO	Z	B	M	Z	B
PP	B	Z	B	Z	B

Source: Own authorship (2022)

Figure 14 – Entries membership function E e de obtained by the PSO.



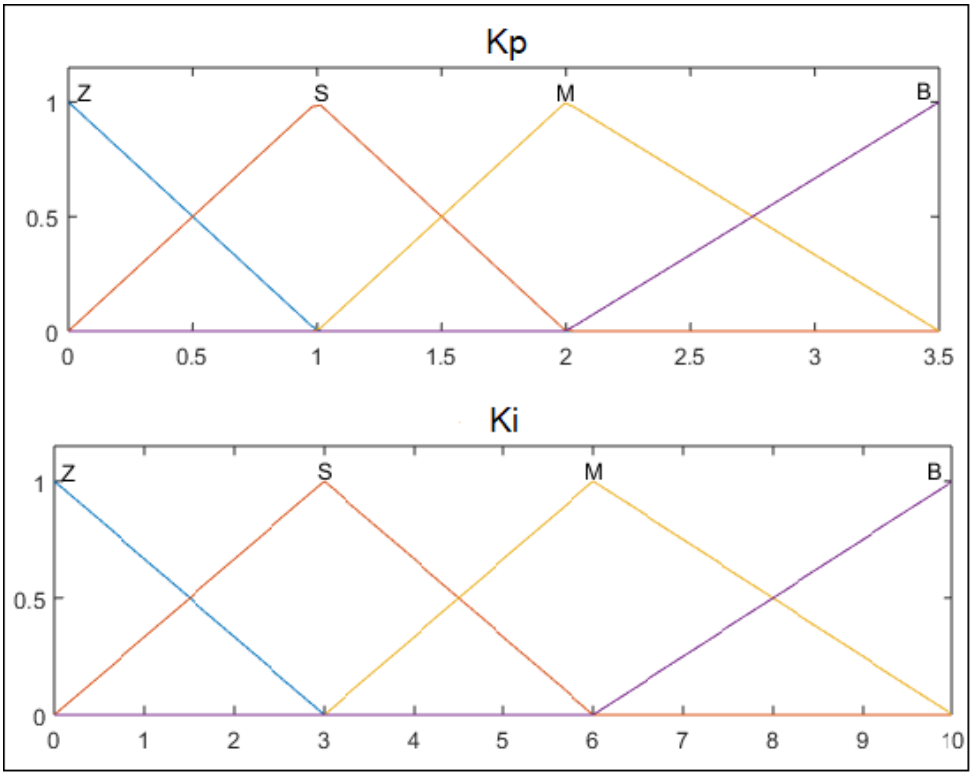
Source: Own authorship (2022)

3.5 Optimization of the GAPID controller by PSO

As discussed in Chapter 2, the determination of the 9 parameters, K_{p_1}, K_{i_1}, K_{d_1}, K_{p_0}, K_{i_0}, K_{d_0}, q_p, q_i and q_d, is not an easy task to perform empirically. Due to the complexity of the problem, it was decided to use the PSO to obtain them.

Before starting the optimization algorithm, some crucial factors were considered, such as the number of variables to be obtained and how the adaptive controller would be applied. Because the plant to be controlled has characteristics of a first-degree system, the parameters linked to the proportional gain were discarded, thus reducing the problem

Figure 15 – Output membership function K_p e K_i obtained by the PSO.

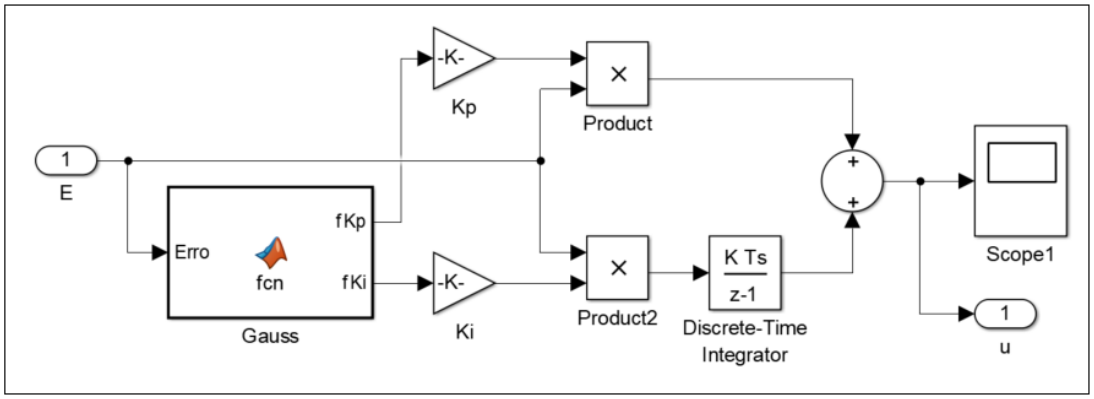


Source: Own authorship (2022)

to 6 variables. With that, the controller will be called GAPI.

Instead of using the specialized solution, which consists of the direct application of the Gaussian function in the plant, it was decided to link the set of parameters to the gains of the linear PI controller and, in this way, take advantage of the exact project requirements but seeking better performance, as shown in Figure 16.

Figure 16 – GAPI Block in Simulink.



Source: Own authorship (2022)

For the optimization algorithm (PSO), the same steps presented in section 3.4 were followed, ten simulations were compared, and the best result was chosen among

as explained in topic 3.4.

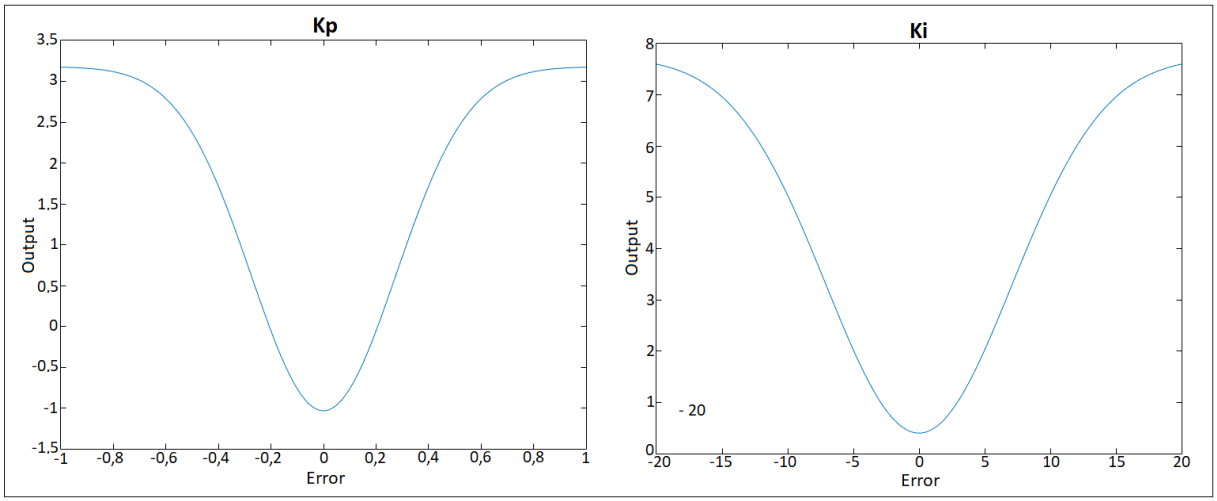
According to Table 3 and Graph 5, it is noted that, contrary to the expected scenario, K_p with an upward curve and K_i with a downward curve. Both gains had their curves turned downwards.

Table 3 – GAPI parameters.

Parameter	Value
K_{p_1}	3.17901
K_{i_1}	7.73366
K_{p_0}	-1.03353
K_{i_0}	0.39996
q_p	6.59485
q_i	5.16543

Source:Own Authorship (2022).

Graph 5 – GAPI Gaussian curves obtained by PSO.



Source: Own authorship (2022)

3.6 Controller implementation

As mentioned at the beginning of this chapter, the ESP32 microcontroller was used to control the motor speed through the signals obtained by the encoder. In this case, it was opted to use the Arduino IDE to program the controller due to the familiarity with the language and the practicality of monitoring the results obtained. In this way, the following steps were performed:

1. Obtaining a reliable pulse count approaching reality using 50ms interrupts;
2. Describing the discrete-time system by the difference equation;
3. Using the value resulting from the control effort to generate a PWM pulse;

3.6.1 Difference Equation

For the microcontroller to be able to interpret the controllers to be implemented, it is necessary to describe the discrete-time system by a difference equation model.

The Equation 27 can be written as follows:

$$U(z) \cdot (z-1) = E \cdot 0.0009113 \cdot (z + 0.0002364) \tag{27}$$

Thus, performing the inverse Z transform of Equation 27, $u_{(k)}$ is obtained, which is described by Equation 28.

$$u_{(k)} = u_{(k-1)} + 0.0009113 \cdot e_{(k)} + 0.0002364 \cdot e_{(k-1)} \tag{28}$$

Where:

u_k represents the output signal;

$u_{(k-1)}$ represents the past output signal;

e_k represents the current input error signal;

$e_{(k-1)}$ represents the past input error signal.

It is worth noting that Equation 28 represents the differential equation of the PI controller, as the hybrid Fuzzy-PI controller is nothing more than the multiplication of the outputs resulting from the Fuzzy logic by the gains of the PI controller. Therefore, the differential equation used for implementing this controller and the controller optimized by the PSO can be given by Equation 29

$$u_{(k)} = u_{(k-1)} + (0.0009113 \cdot e_{(k)} \cdot A) + (0.0002364 \cdot e_{(k-1)} \cdot B) \quad (29)$$

Where:

- A represents K_p output value obtained through fuzzy logic;
- B represents K_i output value obtained through fuzzy logic.

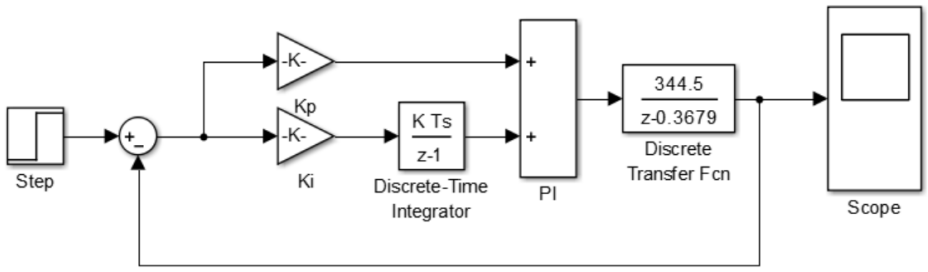
As the PWM pulse to be sent to the motor will be multiplied by $u_{(k)}$, the control effort was normalized to vary between 0 and 1 continuously. In other words, when $u_{(k)}$ saturates, as in the first moment where the motor needs to come out of inertia and $e_{(k)}$ s maximum, the value that could go from 1 becomes 1 without bursting the maximum value of the PWM emitted by the microcontroller which is 255 (8 bits).

4 RESULTS AND DISCUSSION

This chapter presents the simulation and experimental applications results obtained, where the simulations for all four controllers were performed in closed-loop for a setpoint of 2900 pulses/Ts using the Simulink software, which in all situations the load was not considered, as shown in Figure 17, Figure 18 and Figure 19, these simulations resulted in the curves Graph 6.The tests with load were performed on a test bench where a felt disk was applied directly to the rotor of the motor at pre-established time intervals.

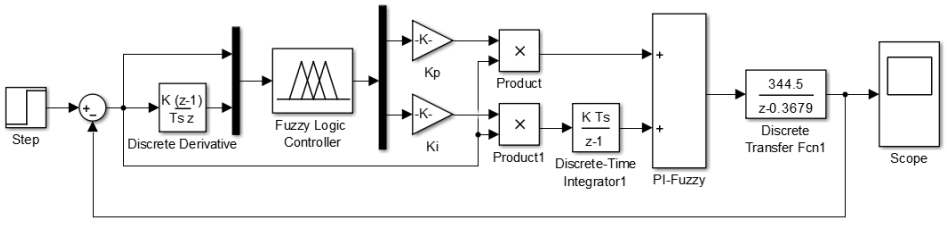
The best results obtained for each section in all the tables that will be presented in the next topics are highlighted and underlined for better analysis.

Figure 17 – PI controller simulation in Simulink.



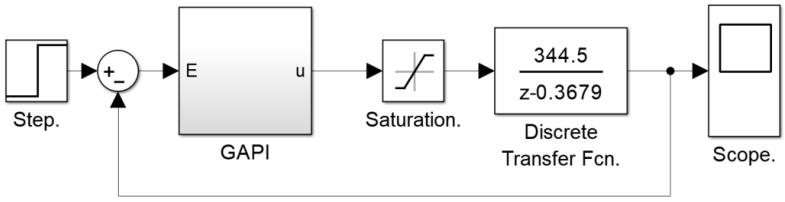
Source: Own authorship (2022)

Figure 18 – Fuzzy-PI Hybrid controller simulation in Simulink.



Source: Own authorship (2022)

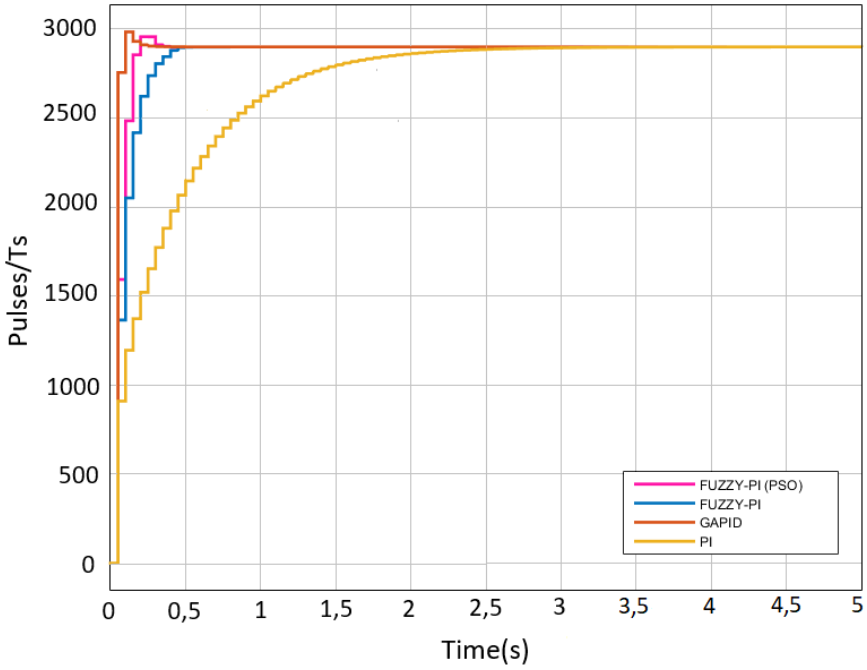
Figure 19 – GAPI controller simulation in Simulink.



Source: Own authorship (2022)

Table 4 and Graph 6 show that the GAPI controller obtained the best results in the rise time and settling time, about 1.64 seconds faster than the PI controller, but

Graph 6 – Curve of the four controllers simulated without load for a reference of 2900 pulses/Ts.



Source: Own authorship (2022)

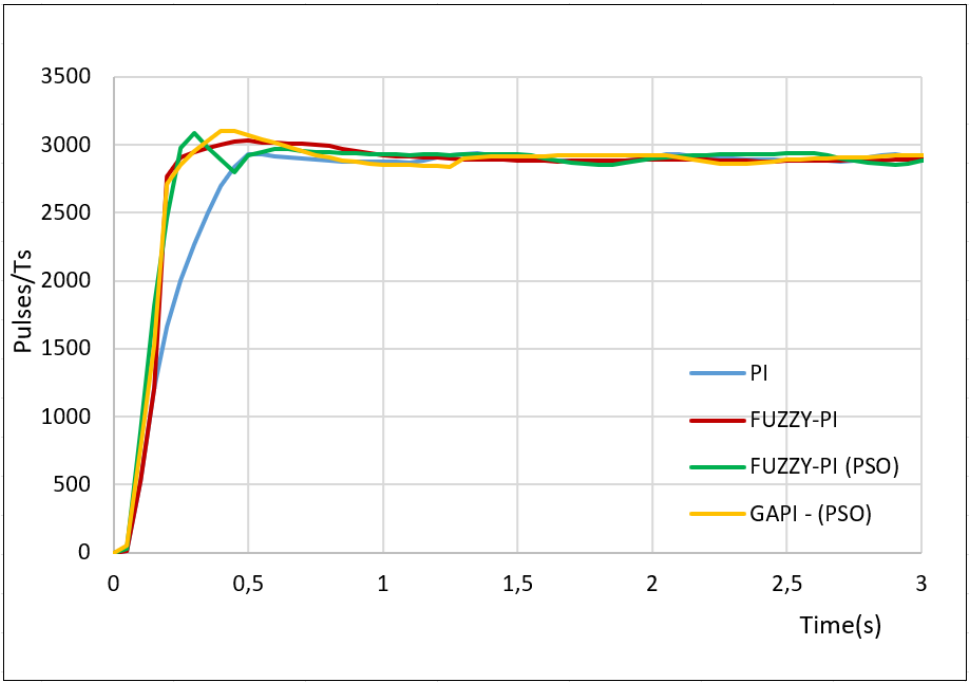
controller showed a fast response time with an overshoot of 0 %, and its PSO-optimized version showed a slight improvement, but with an overshoot of 1.9 %.

Table 4 – Simulation Results.

Section	PI	Fuzzy-PI	Fuzzy-PI (PSO)	GAPI
Rise Time (s)	0.9587	0.1864	0.1078	**0.0421**
Settling Time (s)	1.7738	0.3463	0.1482	**0.1246**
Settling Min.	2624	2622	**2855**	2756
Settling Max.	2900	2900	2957	2984
Overshoot (%)	**0**	**0**	1.9663	2.9017
Peak	2900	2900	2957	**2984**
Peak Time (s)	10	10	0.2	**0.1**

Source:Own Authorship (2022).

Graph 7 – Curve of the four controllers implemented without load for a reference of 2900 pulses/Ts.



Source: Own authorship (2022)

As much as the designer tries to carry out simulations taking into account as many variables as possible to get closer to reality, it is still possible to forget one of them, which is evident when we compare the results obtained in the bench tests without load.

To avoid a discrepancy between the simulated and obtained results, using saturators in the controller output simulations was taken as a countermeasure, as illustrated in Figure 19, thus limiting the voltage applied to the motor to 11.1 V.

With the data obtained, it is possible to notice that the PI controller was adequately designed, achieving a better performance than the simulated one with a rise time of 0.3 seconds and a similar percentage of overshoot of 1.3 %.

Another interesting fact is that in the implementation, the PSO-optimized Fuzzy-PI hybrid controller could not perform better than all the controllers as shown in the simulation, having a 2% higher overshoot percentage and a 0.02 seconds slower rise time than the Fuzzy-PI hybrid controller.

The Fuzzy-PI hybrid controller only gained the stabilization time, and the GAPI controller presented the highest overshoot, 7 %, according to Table 5 and Graph 7.

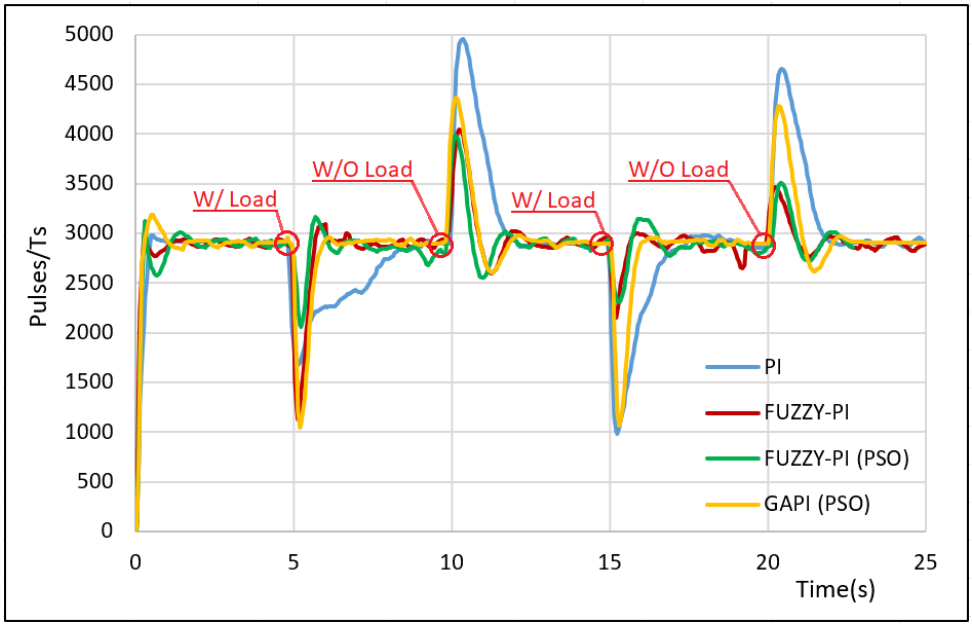
Table 5 – Implementation Results without load.

Section	PI	Fuzzy-PI	Fuzzy-PI (PSO)	GAPI
Rise Time (s)	0.3003	**0.1180**	0.1490	0.1288
Settling Time (s)	**0.4532**	0.8844	0.7	1.2545
Settling Min.	2698	2765	**2802**	2713
Settling Max.	**2940**	3030	3086	3103
Overshoot (%)	**1.3793**	4.4828	6.4138	7.000
Peak	2940	3030	3086	3103
Peak Time (s)	1.35	0.50	**0.30**	0.40

Source:Own Authorship (2022).

A load disturbance was inserted to investigate the robustness of each control technique used, where a felt disk was applied directly to the rotor of the BLDC motor and repeatedly removed every 5 seconds. In other words, the load is placed in 5 and 15 seconds and withdrawn at 10 and 20 seconds, as seen in Graph 8.

Graph 8 – Curve of the four controllers implemented with load for a reference of 2900 pulses/Ts.



Source: Own authorship (2022)

According to the graph presented and the measured data in Table 6, it can be noted that the Fuzzy-PI hybrid controller optimized by the PSO has a faster response to load disturbances, managing to maintain the lowest percentage of overshoot among the four analyzed controllers. In addition, it is also worth mentioning the robustness of the PI controller, which despite having a slow response to the load variation, still manages to adapt and reach the established reference, having a shorter stabilization time than the other three controllers.

the adaptive controllers based on fuzzy logic, it fulfilled its function by being practical and leading to a significant improvement for the PI controller with an accommodation time 2 seconds faster than the Fuzzy-PI hybrid controller optimized by PSO and a 20 % reduction in overshoot.

Table 6 – Implementation Results with load.

Section	PI	Fuzzy-PI	Fuzzy-PI (PSO)	GAPI
Rise Time (s)	0.2970	0.1391	0.1512	**0.1289**
Settling Time (s)	**21.8275**	24.71	24.4115	22.3991
Settling Min.	984	1131	**2057**	1047
Settling Max.	4957	4042	**3983**	4369
Overshoot (%)	70.9310	39.3793	**37.3448**	50.5513
Peak	4957	4042	**3983**	4369
Peak Time (s)	10.35	10.25	**10.15**	**10.15**

Source:Own Authorship (2022).

5 CONCLUSIONS

The necessity to optimize processes is increasingly present due to the complexity of systems and the advancement of technology. For this, new control techniques need to be developed and applied.

For the proposed research, the control technique widely used in the industry, the proportional-integral controller, was not enough to meet the project's needs precisely because it does not behave linearly when there are load variations.

Thus, one of the ways to mitigate this problem would be the application of adaptive techniques such as Fuzzy and Gaussian. However, for such techniques to be used correctly in the controllers, the designer must have complete knowledge of the control system. As it is an entirely empirical control, the values used in the simulation and implementation may not be ideal.

In this way, using an optimization algorithm was the best way to get reliable results. As the PSO creates a universe of possible results and, in a few interactions, can obtain the best values to project. It was achievable to find the parameters necessary for a Fuzzy system, such as the membership functions and their respective degrees, as well as the organization of the rule base and the bottom and upper limits much needed for the GAPI controller.

Based on the results obtained, it became clear that the Fuzzy-PI Hybrid controller, when properly designed, proves to be a powerful tool. Due to its construction, it is possible to perform precise control with a high level of fine adjustment. On the other hand, it requires high processing power compared to the GAPI controller, which can make this technique unattractive for specific projects, mainly when focusing on developing a low-cost product. For example, the ESP32 microcontroller can be replaced by an Arduino Nano, as it has enough processing power to run the GAPI controller algorithm and obtain good results.

Despite not presenting the best results, the GAPI controller fulfilled its role by supplying some evident deficiencies in the PI controller for the test proposal of this study.

In this work, it was possible to conclude that the union of these control and optimization techniques resulted in a controller that could supply the PI controller's low response time for load variations, reduce the complexity, and eliminate the empiricism of the applied adaptive techniques.

Although the results were positive for the motor speed control, an important point for this type of control was not considered, which was the efficiency. As mentioned at the beginning of this paper, efficiency is the main focus of the electric vehicle industry, when the BLDC motor is used in this application, several variables must be taken into account, such as the electric current.

It is proposed for future research, the improvement of Fuzzy and GAPI controllers using the current demanded by the motor as a second control variable, aiming at optimization, greater efficiency and a longer battery usage time, as well as the analysis of the particularities of this electrical magnitude in a BLDC motor.

REFERENCES

ARIS, RSN Adiimah Raja *et al.* Enhancement of variable speed brushless dc motor using neural network. **Indian J. of Science and Technology**, v. 9, n. 14, p. 1–9, 2016.

BORGES, Fábio Galvão *et al.* Metaheuristics-Based Optimization of a Robust GAPID Adaptive Control Applied to a DC Motor-Driven Rotating Beam with Variable Load. **Sensors**, v. 22, n. 16, p. 6094, 2022. Publisher: MDPI.

CHHLONH, Chhith; KIM, Bunthern, *et al.* Generation of efficient interprocedural analyzers with PAG. **2021 International Symposium on Electrical and Electronics Engineering (ISEE)**, p. 166–171, 2021.

CHHLONH, Chhith; RIAWAN, Dedet Candra; SURYOATMOJO, Heri. Modeling and simulation of independent speed steering control for front in-wheel in EV using BLDC motor in MATLAB GUI. **2019 International Seminar on Intelligent Technology and Its Applications (ISITIA)**, IEEE, p. 270–275, 2019.

CHOI, Kang-Min *et al.* Active control for seismic response reduction using modal-fuzzy approach. **International Journal of Solids and Structures**, v. 42, n. 16, p. 4779–4794, 2005. Publisher: Elsevier.

CHOPRA, Vikram; SINGLA, Sunil K.; DEWAN, Lillie. Comparative analysis of tuning a PID controller using intelligent methods. **ACTA Polytechnica hungarica**, v. 11, n. 8, p. 235–249, 2014.

DE AZEVEDO, Fernando Mendes; BRASIL, Lourdes Mattos; OLIVEIRA, Roberto Célio Limão de. **Redes neurais com aplicações em controle e em sistemas especialistas**. [*S. l.*]: Visual Books, 2000.

DEL TORO, Vincent. **Fundamentos de máquinas elétricas**. [*S. l.*]: Prentice-Hall do Brasil, 1994.

DOPPELBAUER, Martin. The invention of the electric motor 1800-1854. **Philosophical Magazine**, v. 59, 1822.

EBERHART, Russell C; SHI, Yuhui; KENNEDY, James. **Swarm intelligence**. [*S. l.*]: Elsevier, 2001.

FENG, G. *et al.* A model reference adaptive control algorithm for fuzzy dynamic systems. **Proceedings of the 4th World Congress on Intelligent Control and Automation (Cat. No. 02EX527)**, IEEE, v. 4, p. 3242–3246, 2002.

FITGERALD, AE; KINGSLEY JR, Charles; UMANS, Stephen D. **Máquinas Elétricas. 6ª edição**. [*S. l.*]: Porto Alegre–RS: Editora Bookman, 2008.

FRANKLIN, Gene F; POWELL, J David; EMAMI-NAEINI, Abbas. **Sistemas de controle para engenharia**. [*S. l.*]: Bookman Editora, 2013.

GARCIA-GONZALO, Esperanza; FERNANDEZ-MARTINEZ, Juan Luis. A brief historical review of particle swarm optimization (PSO). **Journal of Bioinformatics and Intelligent Control**, v. 1, n. 1, p. 3–16, 2012. Publisher: American Scientific Publishers.

GEETHA, V.; THANGAVEL, S. Performance analysis of direct torque controlled BLDC motor using fuzzy logic. **International Journal of Power Electronics and Drive Systems**, v. 7, n. 1, p. 144, 2016. Publisher: IAES Institute of Advanced Engineering and Science.

GHANY, MA Abdel; SHAMSELDIN, Mohamed A.; GHANY, AM Abdel. A novel fuzzy self tuning technique of single neuron PID controller for brushless DC motor. **2017 nineteenth international middle east power systems conference (MEPCON)**, IEEE, p. 1453–1458, 2017.

GOSWAMI, Rakesh; JOSHI, Dheeraj. Performance review of fuzzy logic based controllers employed in brushless DC motor. **Procedia computer science**, v. 132, p. 623–631, 2018. Publisher: Elsevier.

HOLLAND, Jeffrey R. **An High Priest of Good Things to Come**. [*S. l.: s. n.*], 1999. Available from: https://www.churchofjesuschrist.org/study/general-conference/1999/10/an-high-priest-of-good-things-to-come.

IZO, Alexandre. **Rota Mundial de Carros Elétricos Cresce 55% No Primeiro Semestre de 2018**. [*S. l.: s. n.*], 2018. Available from: https://revistaautoesporte.globo.com/Noticias/noticia/2018/08/frota-mundial-de-carros-eletricos-cresce-55-em-um-ano.html.

JAVORSKI ECKERT, Jony *et al.* Energy storage and control optimization for an electric vehicle. **International Journal of Energy Research**, v. 42, n. 11, p. 3506–3523, 2018. Publisher: Wiley Online Library.

JÚNIOR, Francisco Guerra Fernandes *et al.* Implementação de controladores PID utilizando lógica fuzzy e instrumentação industrial. **VII Simpósio Brasileiro Automação Inteligente, São Luiz, Brazil**, p. 27–30, 2005.

KENNEDY, James; EBERHART, Russell. Particle swarm optimization. **Proceedings of ICNN'95-international conference on neural networks**, IEEE, v. 4, p. 1942–1948, 1995.

KOSOW, I.I. **Maquinas eletricas e transformadores**. [*S. l.*]: Globo, 1993. ISBN 9788525002303.

KUMAR, Brajesh; SWAIN, Subrat Kumar; NEOGI, Nirbhar. Controller design for closed loop speed control of BLDC motor. **International Journal on Electrical Engineering and Informatics**, v. 9, n. 1, p. 146, 2017.

KUMAR, Neelam Sanjeev; CHANDRASEKARAN, Gokul, *et al.* A Novel Design Methodology and Numerical Simulation of BLDC Motor for Power Loss Reduction. **Applied Sciences**, v. 12, n. 20, p. 10596, 2022. Publisher: MDPI.

KUMPANYA, Danupon; THAIPARNAT, Sattarpoom; PUANGDOWNREONG, Deacha. Parameter identification of BLDC motor model via metaheuristic optimization techniques. **Procedia Manufacturing**, v. 4, p. 322–327, 2015. Publisher: Elsevier.

LI, Jun *et al.* Chaotic Characteristic Analysis of Brushless DC Motor with Vibration Load Disturbance. **Journal of Engineering Science & Technology Review**, v. 11, n. 6, 2018.

MATULKA, Rebecca. **The History of the Electric Car.-US Department of ENERGY**. [*S. l.: s. n.*], 2014. Available from: https://www.energy.gov/articles/history-electric-car.

NARMADA, R.; AROUNASSALAME, M. Design and performance evaluation of Fractional ORDER controller for Brushless DC Motor. **International Journal on Electrical Engineering and Informatics**, v. 6, n. 3, p. 606, 2014. Publisher: School of Electrical Engineering and Informatics, Bandung Institute of ...

NISE, Norman S; SILVA, Fernando Ribeiro da. **Engenharia de sistemas de controle**. [*S. l.*]: LTC, 2002. v. 3.

OGATA, K. **Engenharia de controle moderno**. [*S. l.*]: PRENTICE HALL BRASIL, 2011. ISBN 9788576058106.

PAGANI, Regina Negri; KOVALESKI, João Luiz; RESENDE, Luis Mauricio. Methodi Ordinatio: a proposed methodology to select and rank relevant scientific papers encompassing the impact factor, number of citation, and year of publication. **Scientometrics**, v. 105, p. 2109–2135, 2015. Publisher: Springer.

PUCHTA, Erickson Diogo Pereira *et al.* Swarm-inspired algorithms to optimize a nonlinear gaussian adaptive PID controller. **Energies**, v. 14, n. 12, p. 3385, 2021. Publisher: MDPI.

EL-SAMAHY, Adel A.; SHAMSELDIN, Mohamed A. Brushless DC motor tracking control using self-tuning fuzzy PID control and model reference adaptive control. **Ain Shams Engineering Journal**, v. 9, n. 3, p. 341–352, 2018. Publisher: Elsevier.

SHI, Jun *et al.* Optimizing BLDC motor drive performance using particle swarm algorithm-tuned fuzzy logic controller. **SN Applied Sciences**, v. 4, n. 11, p. 293, 2022. Publisher: Springer.

SILVA, Guillermo J.; DATTA, Aniruddha; BHATTACHARYYA, Shankar P. New results on the synthesis of PID controllers. **IEEE transactions on automatic control**, v. 47, n. 2, p. 241–252, 2002. Publisher: IEEE.

SIMÕES, Marcelo Godoy; SHAW, Ian S. **Controle e modelagem fuzzy**. [*S. l.*]: Editora Blucher, 2007.

SUTARNA, Nana; PURWANTI, BS Rahayu; SUHADHA, Lingga. The Fuzzy PID Controller Performance in BLDC Motor Rotor Speed Variable. **2022 9th International**

Conference on Electrical Engineering, Computer Science and Informatics (EECSI), IEEE, p. 321–326, 2022.

VARGHESE, Albert John; ROY, Rejo; THIRUNAVUKKARASU, S. Optimized speed control for BLDC motor. **International Journal of Innovative Research in Science, Engineering and Technology**, v. 3, p. 1019–30, S1 2014.

WANG, L.-X. Stable adaptive fuzzy control of nonlinear systems. **IEEE Transactions on fuzzy systems**, v. 1, n. 2, p. 146–155, 1993. Publisher: IEEE.

XIA, Chang-liang. **Permanent magnet brushless DC motor drives and controls**. [*S. l.*]: John Wiley & Sons, 2012.

APPENDIX A — CODES USED FOR SIMULATIONS AND IMPLEMENTATION

In this section, MatLab Software was used in order to perform the necessary calculations to obtain the discretized plant and the development of the PI controller in which the root locus method was applied.

In the same way, the algorithm for the particle swarm optimization of the Fuzzy-PI hybrid controller and GAPI controller were developed.

PI Controller

```
numP=[6050];
denP=[0.05 1];
P=tf(numP,denP);
P=P/11.1;
Ts=0.05;
Pd=c2d(P,Ts);
step(Pd)
hold on
PSS = 1.3;
qsi = log10(100/PSS)/(sqrt((pi^2)+log10(100/PSS)^2));}
Te= 0.5;
wn = 4/(Te*qsi)
sigma = -qsi*wn
wd = wn*sqrt(1-(qsi$^$2))
s= sigma + j*wd
z = exp(s*Ts)
rp = freqresp (Pd,z)
mp = abs(rp)
tetap = angle$^$s(rp)
tetak = -pi - tetap
tetad=angle(z-1)
tetan=tetak+tetad
tan(tetan);
alpha=(imag(z)-(real(z)*ans))/-ans
mk=abs((z-alpha)/(z-1))
k=1/(mk*mp)
numk=k*[1 -alpha]
denk=[1 -1]
K=tf(numk,denk,Ts)
G=K*Pd;
T=feedback(G,1);
kp1=alpha*k;
ki1=k-kp1;
step(T);
```

Particle swarm optimization algorithm - Fuzzy

```
% chama o controlador PI
controlador();
% estabelecimento dos parâmetros de inicialização
NUM_PARTICULAS = 40;
NUM_GERACOES = 35;
nIteracoes = 1;
best_fit = 0;
fit_max=0; vetor_fit_max=0;

% Inicialização dos limites para as funções de Pertinência
```

```
ub1 = [0 1.5 2.5 3.5 600 2650 1300 4000 3 6 15];
lb1 = [0 1 2 3 300 2150 800 3500 2.5 5 7];
numRegras=25;
ub=ub1; lb=lb1;
for i=1:numRegras
    % limites para kp
    ub(i+length(ub1)) = 4;
    lb(i+length(lb1)) = 1;
    % limites para ki
    ub(numRegras+i+length(ub1)) = 4;
    lb(numRegras+i+length(lb1)) = 1;
end
[~, numVar] = size(ub);
numFuncoes = 11; %regras SC
numRegras = numVar - numFuncoes;

% Executa o PSO
for n=1:nIteracoes
disp(' ----------------  BPSO   ---------------- ');

% Inicia o enxame com vetor velocidade zero
    swarm = [];
    for i=1:NUM_PARTICULAS
        swarm = [swarm, Particula()];
        for j=1:length(lb)
            swarm(i).vel(j)=0;
        end
    end

% Gera posição aleatória
    for i=1:length(swarm)
        swarm(i)=swarm(i).inicPosicao(lb,ub,numRegras);
    end

%Inicia a partícula gBest
   gBest = Particula();

  classdef Particula
    properties
        pos = [];
        vel = [];
        fit = 0;
        pbestpos;
        pbest = 0;

    end
    methods
        function self = inicPosicao(self,lb,ub,numRegras)
            for i = 1:length(lb)
     self.pos = [self.pos, rand*(ub(1,i)-lb(1,i))+lb(1,i)];
            end
            for k=1:numRegras
while(self.pos(k)==0 && self.pos(k+numRegras)==0)
        self.pos(k) = randi([lb(1,k),ub(1,k)]);
        self.pos(k+numRegras)=
        randi([lb(1,k+numRegras),ub(1,k+numRegras)]);
                end
            end
        end
```

```
        function self = calculaFit(self)
            self.fit = avaliacao(self.pos);
            if self.fit > self.pbest
                self.pbest = self.fit;
                self.pbestpos = self.pos;
            end
        end
        function self = calcVel(self,gBest,i,NUM_GERACOES)
        w=0.5;
        c1=1;
        c2=2.5;
for i=1:length(self.vel)
self.vel(i) = w*self.vel(i) + c1*rand*(self.pbestpos(i)
- self.pos(i)) +
c2*rand*(gBest.pos(i) - self.pos(i));
            end
        end
        function self = calcPos(self,lb,ub,numRegras)
            self.pos = round(self.pos+self.vel);

%Checagem dos Limites
            contLimites=0;
            for c=1:length(lb)
                if self.pos(c) > ub(c)
                    self.pos(c) = ub(c);
                    contLimites=contLimites+1;
                elseif self.pos(c) < lb(c)
                    self.pos(c) = lb(c);
                    contLimites=contLimites+1;
                end
            end
    fprintf('Bounds: %d out of %d.\n',contLimites,length(lb));
            for k=1:numRegras
    while(self.pos(k)==0 && self.pos(k+numRegras)==0)
    self.pos(k) = randi([lb(1,k),ub(1,k)]);
    self.pos(k+numRegras) = randi([lb(1,k+numRegras),
    ub(1,k+numRegras)]);
                    disp('Regra nula por calcPos');
                end
            end
        end
        function self = mut(self,lb,ub,numRegras)
            self.pos(i) = randi([lb(1,k),ub(1,k)]);
            for k1=1:numRegras
    while(self.pos(k1)==0 && self.pos(k1+numRegras)==0)
    self.pos(k1) = randi([lb(1,k1),ub(1,k1)]);
    self.pos(k1+numRegras) =  randi([lb(1,k1+numRegras),
    ub(1,k1+numRegras)]);
                    disp('Regra nula por mutacao');
                end
            end
        end
    end
  end

%Realização dos Cálculos
    i = 1;
    while i <= NUM_GERACOES
```

```
%Calculo fitness
        for j=1:NUM_PARTICULAS
        fprintf('Avaliacao %d de %d\n',j,NUM_PARTICULAS);
            swarm(j)=swarm(j).calculaFit();

% Atualiza o gbest
            if(swarm(j).fit > gBest.fit)
                gBest = swarm(j);
            end
        end
        hold off
        fprintf('\n');
        %plot - melhor fitness
        vetor_fit_max(i)=gBest.fit;
        subplot(2,2,[1,2])
        plot(vetor_fit_max)
        axis([0 inf 0 max(vetor_fit_max)])

% Busca local da melhor velocidade e posição
        for j=1:NUM_PARTICULAS
            swarm(j)=swarm(j).calcVel(gBest,i,NUM_GERACOES);
        end

        for j=1:NUM_PARTICULAS
            swarm(j)=swarm(j).calcPos(lb,ub,numRegras);
        end

        fprintf('Ger: %d \nMelhor Fit: %f\n\n',i,gBest.fit);
        if gBest.fit == 1
            break
        end
        i=i+1;
    end
    disp('-- FIM --');

    best(n,:) = gBest.pos;
    dispersaoG(1,n) = i;
    dispersaoR(1,n) = gBest.fit;
    if gBest.fit > best_fit
        fit_idx = n;
        vetor_fit_max1 = vetor_fit_max;
        best_fit = gBest.fit;
    end
save('log.mat','best','fit_idx','
dispersaoG','dispersaoR','vetor_fit_max1');
end

% Respostas
load log.mat

%Atualiza o arquivo .fis do controlador Fuzzy
atualizacaoFuzzy(best(fit_idx,:));
   function atualizacaoFuzzy(x1)
   global a

% Configuração do Fuzzy
x2=x1(1,1:11);
a=newfis('fuzzy');
```

```
a=addvar(a,'input','erro',[-x2(8) x2(8)]);
a=addmf(a,'input',1,'NG','trimf',[-x2(8) -x2(8) -x2(6)]);
a=addmf(a,'input',1,'NP','trimf',[-x2(8) -x2(6) x2(1)]);
a=addmf(a,'input',1,'ZO','trimf',[-x2(6) x2(1) x2(6)]);
a=addmf(a,'input',1,'PP','trimf',[x2(1) x2(6) x2(8)]);
a=addmf(a,'input',1,'PG','trimf',[x2(6) x2(8) x2(8)]);
;
a=addvar(a,'input','derro',[-x2(7) x2(7)]);
a=addmf(a,'input',2,'NG','trimf',[-x2(7) -x2(7) -x2(5)]);
a=addmf(a,'input',2,'NP','trimf',[-x2(7) -x2(5) x2(1)]);
a=addmf(a,'input',2,'ZO','trimf',[-x2(5) x2(1) x2(5)]);
a=addmf(a,'input',2,'PP','trimf',[x2(1) x2(5) x2(7)]);
a=addmf(a,'input',2,'PG','trimf',[x2(5) x2(7) x2(7)]);

a=addvar(a,'output','Kp',[x2(1) x2(4)]);
a=addmf(a,'output',1,'Z','trimf',[x2(1) x2(1) x2(2)]);
a=addmf(a,'output',1,'S','trimf',[x2(1) x2(2) x2(3)]);
a=addmf(a,'output',1,'M','trimf',[x2(2) x2(3) x2(4)]);
a=addmf(a,'output',1,'B','trimf',[x2(3) x2(4) x2(4)]);

a=addvar(a,'output','Ki',[x2(1) x2(11)]);
a=addmf(a,'output',2,'Z','trimf',[x2(1) x2(1) x2(9)]);
a=addmf(a,'output',2,'S','trimf',[x2(1) x2(9) x2(10)]);
a=addmf(a,'output',2,'M','trimf',[x2(9) x2(10) x2(11)]);
a=addmf(a,'output',2,'B','trimf',[x2(10) x2(11) x2(11)]);

ruleList=[ ...
1 1 round(x1(12)) round(x1(37)) 1 1
1 2 round(x1(13)) round(x1(38)) 1 1
1 3 round(x1(14)) round(x1(39)) 1 1
1 4 round(x1(15)) round(x1(40)) 1 1
1 5 round(x1(16)) round(x1(41)) 1 1
2 1 round(x1(17)) round(x1(42)) 1 1
2 2 round(x1(18)) round(x1(43)) 1 1
2 3 round(x1(19)) round(x1(44)) 1 1
2 4 round(x1(20)) round(x1(45)) 1 1
2 5 round(x1(21)) round(x1(46)) 1 1
3 1 round(x1(22)) round(x1(47)) 1 1
3 2 round(x1(23)) round(x1(48)) 1 1
3 3 round(x1(24)) round(x1(49)) 1 1
3 4 round(x1(25)) round(x1(50)) 1 1
3 5 round(x1(26)) round(x1(51)) 1 1
4 1 round(x1(27)) round(x1(52)) 1 1
4 2 round(x1(28)) round(x1(53)) 1 1
4 3 round(x1(29)) round(x1(54)) 1 1
4 4 round(x1(30)) round(x1(55)) 1 1
4 5 round(x1(31)) round(x1(56)) 1 1
5 1 round(x1(32)) round(x1(57)) 1 1
5 2 round(x1(33)) round(x1(58)) 1 1
5 3 round(x1(34)) round(x1(59)) 1 1
5 4 round(x1(35)) round(x1(60)) 1 1
5 5 round(x1(36)) round(x1(61)) 1 1
];
a = addrule(a,ruleList);
writefis(a,'fuzzy');
end

fuzzy fuzzy.fis
```

```
%simula o controlador Fuzzy
sim('Controle_Fuzzy_PID_Motor23');
```

Particle swarm optimization algorithm - GAPI

```
Sav_Gbests = zeros(1,7);
%% Problem Definition
FitFunction = @(X) MotorFitGAPID_Carlos(X);
nVar = 6;               %Number of genes of a particle
VarSize = [1 nVar];     %Matrix size of Particle
VarMin = 0.00000001;            %Lower bound
VarMax = 0.01;             %Upper bound

%% Parameters of the PSO
MaxIt =180;    %Max Iterations
nPop = 50;      %Population Size (swarm size)
%w = 0.6;          %Inertia coefficient
wmax = 0.99;    %Max inertial weight value
wmin = 0.6;     %Min inertial weight value
wdamp = 0.99;   %Damping Ratio of Inertia Coefficient
c1 = 1.5;         %Personal Acceleration COefficient
c2 = 2.5;         %Social Acceleration COefficient

MaxVelocity = 100*(VarMax - VarMin);
MinVelocity = -MaxVelocity;

simi = 1;

%% Initialization
empty_particle.Position = [];
empty_particle.Velocity = [];
empty_particle.Fit = [];
empty_particle.Best.Position = [];
empty_particle.Best.Fit = [];
X = zeros(nPop, nVar);

%Create population array:
particle = repmat(empty_particle, nPop, 1);

%Initialize Global Best:
GlobalBest.Fit = 0;
GlobalBest.Position = 0;
for i=1:nPop

    %Generate initial random solutions:
    particle(i).Position= unifrnd(VarMin, VarMax, VarSize);

    %Initialize Velocity at zero:
    particle(i).Velocity = zeros(VarSize);

    %Evaluate the Fit:
    X(i,:) = particle(i).Position;
   kp1 = X(i,1);
        ki1 = X(i,2);
        %kd1 = X(i,3);
        kp0 = X(i,3);
        ki0 = X(i,4);
        qp =  X(i,5);
```

```
        qi =  X(i,6);
%         qd =  X(i,8);
    TempX = X(i,:);

    %------------------------------------------------------
    FitSum = 0;
    AuxFit = FitFunction(TempX);
    FitSum = FitSum + AuxFit;
    FitFinal = FitSum;
    particle(i).Fit = FitFinal;
    %------------------------------------------------------

    %Update Personal best:
    particle(i).Best.Position = particle(i).Position;
    particle(i).Best.Fit = particle(i).Fit;

    %Update Global best:
    if particle(i).Best.Fit > GlobalBest.Fit
        GlobalBest = particle(i).Best;
        GlobalBest.Fit = particle(i).Best.Fit;
        GlobalBest.Position = particle(i).Best.Position;
    end

    %Array with the best Fit at each iteration:
    BestFits = zeros(MaxIt, 1);
    ite = zeros(MaxIt, 1);
    partFit = zeros(MaxIt, 1);
    WorstFits = zeros(MaxIt, 1);
    AvgFits = zeros(MaxIt, 1);
    MutCount = 0;

end


%% Main loop
for it=1:MaxIt
TempFit = GlobalBest.Fit;

    for i=1:nPop


        w = wmax - it.*((wmax-wmin)/MaxIt);
        %Velocity update:
particle(i).Velocity = w*particle(i).Velocity + %
c1*rand(VarSize).*(particle(i).Best.Position - particle(i).Position)+ %
c2*rand(VarSize).*(GlobalBest.Position - particle(i).Position);

        %Update Position:
        particle(i).Position = particle(i).Position + particle(i).Velocity;

        %Evaluation:
        X(i,:) = particle(i).Position;
        kp1 = X(i,1);
        ki1 = X(i,2);
        %kd1 = X(i,3);
        kp0 = X(i,3);
        ki0 = X(i,4);
        qp =  X(i,5);
        qi =  X(i,6);
```

```
        %qd =  X(i,8);
        TempX = X(i,:);
        %--------------------------------------------------------
        FitSum = 0;
            AuxFit = FitFunction(TempX);
            FitSum = FitSum + AuxFit;

        FitFinal = FitSum;
        particle(i).Fit = FitFinal;
        %--------------------------------------------------------

        %Update Personal Best:
        if particle(i).Fit > particle(i).Best.Fit

            particle(i).Best.Position = particle(i).Position;
            particle(i).Best.Fit = particle(i).Fit;

            %Update Global Best:
            if particle(i).Best.Fit > GlobalBest.Fit
                disp(['Gbest Fitness before =  ' num2str(GlobalBest.Fit)]);
                GlobalBest = particle(i).Best;
                GlobalBest.Fit = particle(i).Best.Fit;
                disp(['Gbest Fitness after =  ' num2str(GlobalBest.Fit)]);
            end

        end

    end

    %Store the Best Fit Value
    ite(it) = it;
    WorstFits(it) = min([particle.Fit]);
    AvgFits(it) = median([particle.Fit]);
    BestFits(it) = GlobalBest.Fit;
    partFit(it) = max([particle.Fit]);

    %Display Iteration Information
    disp(['Iteration ' num2str(it) ';     Best Fit = ' num2str(BestFits(it)) ';      %
Best of Ite = ' num2str(partFit(it))]);
    disp(['Gbest =  ' num2str(GlobalBest.Position)]);
    disp(['Gbest repetition count =  ' num2str(MutCount)]);

end


%% Results
figure
plot(ite,BestFits,ite,WorstFits,ite,AvgFits);
xlabel('Iterations');
ylabel('Fitness');
legend('Best','Worst','Average','Location','northoutside','Orientation','horizontal');
grid on;

     Sav_Gbests(1,1:6)  = GlobalBest.Position;
     Sav_Gbests(1,7)    = GlobalBest.Fit;

T = table(Sav_Gbests);
writetable(T,'PSO_Gbests_List_99.xls');
```

Implementation code of controllers in ESP32 microcontroller

For the implementation of the code to the microcontroller and for familiarity with the language, the Arduino IDE was used. In this section are the codes used for all controllers present in this paper.

Open loop control

This code was used for the purpose of obtaining the system transfer function.

```
#include <Wire.h>
hw_timer_t * timer = NULL;
int pulseCount;
void setup() {
  Serial.begin(115200);
  attachInterrupt(digitalPinToInterrupt(15), counter, RISING);
  timer = timerBegin(0, 1, true);
  timerAttachInterrupt(timer, calc, true);
  timerAlarmWrite(timer, 2000000, true);
  timerAlarmEnable(timer);
}void calc() {
 Serial.println(pulseCount);
 pulseCount = 0;
}void counter() {
  pulseCount++;
}void loop() {}
```

PID Controller

```
#include <HardwareSerial.h>
#include <ESP32Servo.h>
hw_timer_t * timer = NULL;
int ref = 2900;
int vel,pwmreal;
double uk=0, uk1=0, ek=0, ek1=0, pwm = 0, erro;
Servo ESC;

void setup() {
Serial.begin(115200);
Serial2.begin(115200, SERIAL_8N1, 16, 17);
ESC.attach(18);
delay(5000);
start();
delay(3000);
}
void start(){
  ESC.write(180);
  delay(3000);
  ESC.write(40);
}
void loop() {
   if (Serial2.available()>0){
    vel = Serial2.parseInt();
    }
  erro = ref - vel;
  ek = erro/255;
  Ts=50ms
      uk= uk1+ (0.0009113*ek) + (0.0002364*ek1);
```

```
      pwmreal=uk;
    uk1=uk;
    ek1=ek;

  if(uk>1) uk=1;
  if(uk<0) uk=0;

  pwm=180*uk;
  ESC.write(pwm);
  Serial.println(vel);
  vel = 0;
}
```

Fuzzy-PID hybrid controller and its PSO-optimized version

The code below was used for the implementation of the hybrid Fuzzy-PID controller, for its implementation only the rules and the values of the membership functions were changed.

```
#define FIS_TYPE double
#define FIS_RESOLUSION 101
#define FIS_MIN -3.4028235E+6
#define FIS_MAX 3.4028235E+6
typedef FIS_TYPE(*_FIS_MF)(FIS_TYPE, FIS_TYPE*);
typedef FIS_TYPE(*_FIS_ARR_OP)(FIS_TYPE, FIS_TYPE);
typedef FIS_TYPE(*_FIS_ARR)(FIS_TYPE*, double, _FIS_ARR_OP);

#include <ESP32Servo.h>
#include <HardwareSerial.h>

int ref = 2900;
double pwm, vel, Velo;
int vel1=0;
double uk=0, uk1=0, ek=0, ek1=0, ek2=0, kp, ki,KP,KI;
Servo ESC;
double erro = 0;
double derro = 0;
float Ts = 0.05;

FIS_TYPE g_fisInput[2];
FIS_TYPE g_fisOutput[2];


void setup() {
Serial.begin(115200);
Serial2.begin(115200, SERIAL_8N1, 16, 17);
ESC.attach(18);
delay(3000);
start();
delay(5000);
}

void start(){
ESC.write(180);
delay(3000);
ESC.write(40);
delay(3000);
```

```
}

void fuzzy()
{
    g_fisInput[0] = erro;
    g_fisInput[1] = derro;

    g_fisOutput[0] = 0;
    g_fisOutput[1] = 0;

    fis_evaluate();

    kp = g_fisOutput[0];
    ki = g_fisOutput[1];
}

FIS_TYPE fis_trimf(FIS_TYPE x, FIS_TYPE* p)
{
      FIS_TYPE a = p[0], b = p[1], c = p[2];
      FIS_TYPE t1 = (x - a) / (b - a);
      FIS_TYPE t2 = (c - x) / (c - b);
      if ((a == b) && (b == c)) return (FIS_TYPE) (x == a);
      if (a == b) return (FIS_TYPE) (t2*(b <= x)*(x <= c));
      if (b == c) return (FIS_TYPE) (t1*(a <= x)*(x <= b));
      t1 = min(t1, t2);
      double yy=0;
      return (FIS_TYPE) max(t1, yy);
}

FIS_TYPE fis_min(FIS_TYPE a, FIS_TYPE b)
{
    return min(a, b);
}

FIS_TYPE fis_max(FIS_TYPE a, FIS_TYPE b)
{
    return max(a, b);
}

FIS_TYPE fis_array_operation(FIS_TYPE *array,
int size, _FIS_ARR_OP pfnOp)
{
    int i;
    FIS_TYPE ret = 0;
    if (size == 0) return ret;
    if (size == 1) return array[0];
    ret = array[0];
    for (i = 1; i < size; i++)
    {
        ret = (*pfnOp)(ret, array[i]);
    }
    return ret;
}
_FIS_MF fis_gMF[] =
{
    fis_trimf
};

int fis_gIMFCount[] = { 5, 5 };
```

```
int fis_gOMFCount[] = { 4, 4 };

FIS_TYPE fis_gMFI0Coeff1[] = { -5000, -5000, -2500 };
FIS_TYPE fis_gMFI0Coeff2[] = { -5000, -2500, 0 };
FIS_TYPE fis_gMFI0Coeff3[] = { -2500, 0, 2500 };
FIS_TYPE fis_gMFI0Coeff4[] = { 0, 2500, 5000 };
FIS_TYPE fis_gMFI0Coeff5[] = { 2500, 5000, 5000 };
FIS_TYPE* fis_gMFI0Coeff[] = { fis_gMFI0Coeff1, fis_gMFI0Coeff2,
fis_gMFI0Coeff3, fis_gMFI0Coeff4, fis_gMFI0Coeff5 };

FIS_TYPE fis_gMFI1Coeff1[] = { -1200, -1200, -600 };
FIS_TYPE fis_gMFI1Coeff2[] = { -1200, -600, 0 };
FIS_TYPE fis_gMFI1Coeff3[] = { -600, 0, 600 };
FIS_TYPE fis_gMFI1Coeff4[] = { 0, 600, 1200 };
FIS_TYPE fis_gMFI1Coeff5[] = { 600, 1200, 1200 };
FIS_TYPE* fis_gMFI1Coeff[] = { fis_gMFI1Coeff1, fis_gMFI1Coeff2,
fis_gMFI1Coeff3, fis_gMFI1Coeff4, fis_gMFI1Coeff5 };

FIS_TYPE** fis_gMFICoeff[] = { fis_gMFI0Coeff, fis_gMFI1Coeff };

FIS_TYPE fis_gMFO0Coeff1[] = { 0, 0, 1 };
FIS_TYPE fis_gMFO0Coeff2[] = { 0, 1, 2 };
FIS_TYPE fis_gMFO0Coeff3[] = { 1, 2, 3 };
FIS_TYPE fis_gMFO0Coeff4[] = { 2, 3, 3 };
FIS_TYPE* fis_gMFO0Coeff[] = { fis_gMFO0Coeff1, fis_gMFO0Coeff2,
fis_gMFO0Coeff3, fis_gMFO0Coeff4 };

FIS_TYPE fis_gMFO1Coeff1[] = { 0, 0, 2.4 };
FIS_TYPE fis_gMFO1Coeff2[] = { 0, 2.4, 4.7 };
FIS_TYPE fis_gMFO1Coeff3[] = { 2.4, 4.7, 7 };
FIS_TYPE fis_gMFO1Coeff4[] = { 4.7, 7, 7 };
FIS_TYPE* fis_gMFO1Coeff[] = { fis_gMFO1Coeff1, fis_gMFO1Coeff2,
fis_gMFO1Coeff3, fis_gMFO1Coeff4 };

FIS_TYPE** fis_gMFOCoeff[] = { fis_gMFO0Coeff, fis_gMFO1Coeff };

int fis_gMFI0[] = { 0, 0, 0, 0, 0 };
int fis_gMFI1[] = { 0, 0, 0, 0, 0 };
int* fis_gMFI[] = { fis_gMFI0, fis_gMFI1};
int fis_gMFO0[] = { 0, 0, 0, 0 };
int fis_gMFO1[] = { 0, 0, 0, 0 };
int* fis_gMFO[] = { fis_gMFO0, fis_gMFO1};
FIS_TYPE fis_gRWeight[] = { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 };
int fis_gRType[] = { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1 };

int fis_gRI0[] = { 1, 1 };
int fis_gRI1[] = { 1, 2 };
int fis_gRI2[] = { 1, 3 };
int fis_gRI3[] = { 1, 4 };
int fis_gRI4[] = { 1, 5 };
int fis_gRI5[] = { 2, 1 };
int fis_gRI6[] = { 2, 2 };
int fis_gRI7[] = { 2, 3 };
int fis_gRI8[] = { 2, 4 };
int fis_gRI9[] = { 2, 5 };
int fis_gRI10[] = { 3, 1 };
int fis_gRI11[] = { 3, 2 };
```

```
int fis_gRI12[] = { 3, 3 };
int fis_gRI13[] = { 3, 4 };
int fis_gRI14[] = { 3, 5 };
int fis_gRI15[] = { 4, 1 };
int fis_gRI16[] = { 4, 2 };
int fis_gRI17[] = { 4, 3 };
int fis_gRI18[] = { 4, 4 };
int fis_gRI19[] = { 4, 5 };
int fis_gRI20[] = { 5, 1 };
int fis_gRI21[] = { 5, 2 };
int fis_gRI22[] = { 5, 3 };
int fis_gRI23[] = { 5, 4 };
int fis_gRI24[] = { 5, 5 };
int* fis_gRI[] = { fis_gRI0, fis_gRI1,
fis_gRI2, fis_gRI3, fis_gRI4, fis_gRI5,
fis_gRI6, fis_gRI7, fis_gRI8, fis_gRI9,
fis_gRI10, fis_gRI11, fis_gRI12,
fis_gRI13, fis_gRI14, fis_gRI15,
fis_gRI16, fis_gRI17, fis_gRI18,
fis_gRI19, fis_gRI20, fis_gRI21,
fis_gRI22, fis_gRI23, fis_gRI24 };

int fis_gRO0[] = { 4, 1 };
int fis_gRO1[] = { 4, 1 };
int fis_gRO2[] = { 4, 1 };
int fis_gRO3[] = { 4, 1 };
int fis_gRO4[] = { 3, 1 };
int fis_gRO5[] = { 3, 3 };
int fis_gRO6[] = { 4, 3 };
int fis_gRO7[] = { 2, 3 };
int fis_gRO8[] = { 2, 3 };
int fis_gRO9[] = { 2, 3 };
int fis_gRO10[] = { 3, 4 };
int fis_gRO11[] = { 4, 4 };
int fis_gRO12[] = { 1, 1 };
int fis_gRO13[] = { 2, 4 };
int fis_gRO14[] = { 4, 4 };
int fis_gRO15[] = { 2, 2 };
int fis_gRO16[] = { 2, 3 };
int fis_gRO17[] = { 2, 3 };
int fis_gRO18[] = { 2, 3 };
int fis_gRO19[] = { 2, 3 };
int fis_gRO20[] = { 3, 1 };
int fis_gRO21[] = { 4, 2 };
int fis_gRO22[] = { 4, 4 };
int fis_gRO23[] = { 3, 4 };
int fis_gRO24[] = { 4, 4 };
int* fis_gRO[] = { fis_gRO0, fis_gRO1,
fis_gRO2, fis_gRO3, fis_gRO4, fis_gRO5,
fis_gRO6, fis_gRO7, fis_gRO8, fis_gRO9,
fis_gRO10, fis_gRO11, fis_gRO12,
fis_gRO13, fis_gRO14, fis_gRO15,
fis_gRO16, fis_gRO17, fis_gRO18,
fis_gRO19, fis_gRO20, fis_gRO21,
fis_gRO22, fis_gRO23, fis_gRO24 };

FIS_TYPE fis_gIMin[] = { -5000, -1200 };
FIS_TYPE fis_gIMax[] = { 5000, 1200 };
FIS_TYPE fis_gOMin[] = { 0, 0 };
```

```
FIS_TYPE fis_gOMax[] = { 3, 7 };

FIS_TYPE fis_MF_out(FIS_TYPE** fuzzyRuleSet, FIS_TYPE x, int o)
{
    FIS_TYPE mfOut;
    int r;
    for (r = 0; r < 25; ++r)
    {
        int index = fis_gRO[r][o];
        if (index > 0)
        {
            index = index - 1;
mfOut = (fis_gMF[fis_gMFO[o][index]])(x,fis_gMFOCoeff[o][index]);
        }
        else if (index < 0)
        {
            index = -index - 1;
mfOut = 1 - (fis_gMF[fis_gMFO[o][index]])
(x,fis_gMFOCoeff[o][index]);
        }
        else
        {
            mfOut = 0;
        }
        fuzzyRuleSet[0][r] = fis_min(mfOut, fuzzyRuleSet[1][r]);
    }
    return fis_array_operation(fuzzyRuleSet[0], 25, fis_max);
}
FIS_TYPE fis_defuzz_centroid(FIS_TYPE** fuzzyRuleSet, int o)
{
FIS_TYPE step = (fis_gOMax[o] - fis_gOMin[o])/ (FIS_RESOLUSION - 1);
    FIS_TYPE area = 0;
    FIS_TYPE momentum = 0;
    FIS_TYPE dist, slice;
    int i;

    for (i = 0; i < FIS_RESOLUSION; ++i){
        dist = fis_gOMin[o] + (step * i);
        slice = step * fis_MF_out(fuzzyRuleSet, dist, o);
        area += slice;
        momentum += slice*dist;
    }

return ((area == 0) ? ((fis_gOMax[o] + fis_gOMin[o])/ 2) :
(momentum / area));}
void fis_evaluate()
{
    FIS_TYPE fuzzyInput0[] = { 0, 0, 0, 0, 0 };
    FIS_TYPE fuzzyInput1[] = { 0, 0, 0, 0, 0 };
    FIS_TYPE* fuzzyInput[2] = { fuzzyInput0, fuzzyInput1, };
    FIS_TYPE fuzzyOutput0[] = { 0, 0, 0, 0 };
    FIS_TYPE fuzzyOutput1[] = { 0, 0, 0, 0 };
    FIS_TYPE* fuzzyOutput[2] = { fuzzyOutput0, fuzzyOutput1, };
    FIS_TYPE fuzzyRules[25] = { 0 };
    FIS_TYPE fuzzyFires[25] = { 0 };
    FIS_TYPE* fuzzyRuleSet[] = { fuzzyRules, fuzzyFires };
    FIS_TYPE sW = 0;
    int i, j, r, o;
    for (i = 0; i < 2; ++i)
```

```
    {
        for (j = 0; j < fis_gIMFCount[i]; ++j)
        {
    fuzzyInput[i][j] = (fis_gMF[fis_gMFI[i][j]])
    (g_fisInput[i], fis_gMFICoeff[i][j]);
        }
    }
    int index = 0;
    for (r = 0; r < 25; ++r)
    {
        if (fis_gRType[r] == 1)
        {
            fuzzyFires[r] = FIS_MAX;
            for (i = 0; i < 2; ++i)
            {
                index = fis_gRI[r][i];
                if (index > 0)
fuzzyFires[r] = fis_min(fuzzyFires[r], fuzzyInput[i]
[index - 1]);
                else if (index < 0)
fuzzyFires[r] = fis_min(fuzzyFires[r], 1 - fuzzyInput[i]
[-index - 1]);
                else
fuzzyFires[r] = fis_min(fuzzyFires[r], 1);
            }
        }
        else
        {
            fuzzyFires[r] = FIS_MIN;
            for (i = 0; i < 2; ++i)
            {
                index = fis_gRI[r][i];
                if (index > 0)
fuzzyFires[r] = fis_max(fuzzyFires[r], fuzzyInput[i]
[index - 1]);
                else if (index < 0)
fuzzyFires[r] = fis_max(fuzzyFires[r], 1 - fuzzyInput[i]
 [-index - 1]);
                else
fuzzyFires[r] = fis_max(fuzzyFires[r], 0);
            }
        }
        fuzzyFires[r] = fis_gRWeight[r] * fuzzyFires[r];
        sW += fuzzyFires[r];
    }
    if (sW == 0)
    {
        for (o = 0; o < 2; ++o)
        {
    g_fisOutput[o] = ((fis_gOMax[o] + fis_gOMin[o]) / 2);
        }
    }
    else
    {
        for (o = 0; o < 2; ++o)
        {
    g_fisOutput[o] = fis_defuzz_centroid(fuzzyRuleSet, o);
        }
    }
```

```
}
void loop() {
    if (Serial2.available()>0){
    vel = Serial2.parseInt();
 }
    erro = ref - vel;
    derro = (erro-ek1)/Ts;

    fuzzy();
    KP=kp*1;
    KI=ki*1;
    ek = erro/255;

    uk= uk1+ ((0.0009113*KP)*ek) + ((0.0002364*KI)*ek1);
    uk1=uk;
    ek1=ek;
    vel1=vel;

  if(uk>1) uk=1;
  if(uk<0) uk=0;

% PWM do ESC 30 valor MIN e 180 valor MAX

  pwm=180*uk;
  ESC.write(pwm);
  Serial.println(vel);
 vel = 0;
}
```

GAPID Controller

```
//para o ESP32
#include <HardwareSerial.h>
#include <ESP32Servo.h>
hw_timer_t * timer = NULL;
int ref = 2900,count=0;
int vel,pwmreal;
double fKi=0, fKp=0,fKd=0, uk=0, uk1=0, ek=0, ek1=0, pwm =0,erro;
double kp1= 3.179014015,
ki1=  7.733658167,
kp0=  -1.033526258,
ki0=  0.399961113,
qp= 6.5948459,
qi= 5.1654269;

Servo ESC;
void setup() {
Serial.begin(115200);
Serial2.begin(115200, SERIAL_8N1, 16, 17);
ESC.attach(18);
delay(5000);
start();
delay(3000);
}
void start(){//ESC.writeMicroseconds(2500);
  ESC.write(180);
  delay(3000);
 ESC.write(40);//ESC.writeMicroseconds(1500);
```

```
}
void loop() {
   if (Serial2.available()>0){
    vel = Serial2.parseInt();
    }
  erro = ref - vel;
  ek = erro/255;
  fKi = ki1 - (ki1 - ki0)*exp(-qi*pow(erro,2));
  fKp = kp1 - (kp1 - kp0)*exp(-qp*pow(erro,2));

      uk= uk1+ ((0.0009113*fKp)*ek) + ((0.0002364*fKi)*ek1);
      pwmreal=uk;
    uk1=uk;
    ek1=ek;

  if(uk>1){
  uk=1;}
  if(uk<0){
  uk=0;}
pwm=180*uk;
  ESC.write(pwm);
  vel = 0;
}
```