UNIVERSITE KASDI MERBAH OUARGLA

Faculté des Sciences Appliquées

Département de Génie Electrique

Mémoire

MASTER ACADEMIQUE

Domaine : Sciences et technologies

Filière : Génie électrique

Spécialité : Electrotechnique Industrielle

Présenté par :

MOULAI ELHADJ Mohammed Tahar DEBBA Choaib

Thème :

A Jaya-Optimized Fuzzy PI Speed Controller for a BLDC motor

Soumis au jury composé de :

Mr	Bouakaz Ouahid	MAA	Président	UKM Ouargla
Mr	Benmakhlouf Abdeslam	MCA	Encadreur/rapporteur	UKM Ouargla
Mr	Taibi Djamel	MAA	Examinateur	UKM Ouargla

Année universitaire 2024/2025

Acknowledgements

Praise be to Allah, by whose grace all good deeds are accomplished. With His guidance and support, we were able to complete this humble work, which represents the culmination of years of study and research.

We would like to express our deepest gratitude and sincere appreciation to our esteemed supervisor, Mr. Abdeslam Benmakhlouf, for his valuable guidance, insightful advice, and continuous support throughout all stages of this thesis. He has been a true example of professionalism and scientific generosity, and we are truly grateful to him.

We also extend our heartfelt thanks to Mr. Mohammed Lakhdar Louazene, Head of the Department of Electrical Engineering, for his constant support and for providing the proper conditions that enabled us to complete our academic journey successfully.

We would also like to acknowledge all our professors for their dedication and for sharing their knowledge throughout our years of study. Our sincere thanks go as well to our families for their unwavering support, and to everyone who contributed, directly or indirectly, to the realization of this work.

Dedication

To those who instilled in us a love for knowledge and guided us from our earliest days, to those whose prayers were the light that illuminated our path through every stage.
To those to whom we owe every success and every step forward in our journey,
To our dear parents, we dedicate this work as the fruit of your sacrifice, patience, and unwavering support.
To our esteemed professors, who generously shared their knowledge and guidance. To our late colleague Ramon Yahya, we will not forget you.
To everyone who believed in us and stood by us with words of encouragement or meaningful support, to our fellow students, with whom we shared the challenges and efforts along the way, to all of you we dedicate this work as a token of our deep gratitude and appreciation.

List of Notations and Acronyms

$\mu_A(x)$**:** membership degree of x to the fuzzy set A.

$\mathbf{R_s}$**:** Stator Resistance

$\mathbf{T_e}$**:** Electromagnetic Torque

J: Moment of inertia of the motor or load ($kg \cdot m^2$).

ϕ_f: The magnet flux (Wb).

λ_{abcs}: Flux Linkages in a.b.c phases (Wb).

IAE: Integral of Absolute Error

$\mathbf{k_i}$**:** Integral Gain

$\mathbf{k_p}$**:** Proportional Gain

BLDC: Brushless Direct Current motor.

EMI: electromagnetic interference

FOC: Field-Oriented Control

PID: Proportional Integral Derivative

FLC: Fuzzy Logic Control

FIS: Fuzzy Inference System

EMF: Electro Motive Force

IGBT: Insulated Gate Bipolar Transistor

List of Figures

Chapter I: Fuzzy Logic Control

Chapter II: Jaya Algorithm

Chapter III: A Fuzzy PI Speed Controller for a BLDC Motor

List of Tables

Chapter III: A Fuzzy PI Speed Controller for a BLDC Motor

Abstract
Brushless DC (BLDC) motors with permanent magnets are a prominent technology in modern industrial applications, particularly favored for low- and medium-power scenarios due to their high efficiency, low maintenance, and high-speed capabilities. This project focuses on designing and developing an advanced control system to enhance the dynamic response of BLDC motors by improving performance in terms of speed, torque, and current consumption.

The proposed methodology involves the implementation of a hybrid Fuzzy-PID controller, which is adept at managing system nonlinearities and parameter variations to ensure precise and stable control under diverse operating conditions. To further elevate the controller's performance, the Jaya algorithm, a simple yet effective population-based metaheuristic, is utilized to optimize the Fuzzy-PID parameters.

Computer simulation results validate the proposed system, demonstrating significant improvements in rise time, overshoot, and steady-state error when compared to conventional PID and standalone fuzzy logic controllers. This research underscores the importance of integrating artificial intelligence and optimization techniques into modern control systems, contributing to the development of highly efficient, reliable, and stable speed regulation strategies for BLDC motors in industrial and real-time applications.

Keywords: Brushless DC motor (BLDC), Speed Control, Fuzzy-PID Controller, Jaya Algorithm, Optimization, Metaheuristic Algorithm.

Résumé

Les moteurs à courant continu sans balais (BLDC) à aimants permanents constituent une technologie de premier plan pour les applications industrielles modernes. Ils sont particulièrement appréciés dans les contextes de faible et moyenne puissance pour leurs caractéristiques remarquables, telles qu'une haute efficacité, un entretien réduit et la capacité de fonctionner à des vitesses élevées. Ce projet a pour objectif de concevoir un système de commande performant afin d'améliorer la réponse dynamique du moteur BLDC en optimisant la vitesse, le couple et la consommation de courant.

La méthodologie adoptée repose sur l'implémentation d'un régulateur PI basé sur la logique floue qui est reconnue pour sa capacité à gérer les non-linéarités et les variations de paramètres du système, assurant un contrôle précis et stable. Pour optimiser davantage ce régulateur, l'algorithme métaheuristique Jaya a été employé pour le réglage automatique de ses paramètres, une méthode qui se distingue par sa simplicité et son efficacité.

Les résultats de simulation informatique confirment les performances du système proposé, montrant une amélioration significative du temps de réponse, une réduction du dépassement et de l'erreur en régime permanent par rapport aux régulateurs PID classiques ou à logique floue seuls. Cette étude met en lumière l'importance de l'intégration des techniques d'intelligence artificielle et d'optimisation dans les systèmes de commande modernes, contribuant ainsi au développement de stratégies plus efficaces pour la régulation de vitesse des moteurs BLDC, garantissant fiabilité et stabilité dans les applications industrielles et temps réel.

Mots-clés : Moteur BLDC, Commande de vitesse, Régulateur PID flou, Algorithme Jaya, Optimisation, Algorithme Métaheuristique.

الملخص

تُعدّ محركات التيار المستمر عديمة الفُرش (BLDC) المزودة بمغناطيس دائم من أبرز المحركات الحديثة المستخدمة في التطبيقات الصناعية، حيث أصبحت تنافس العديد من أنواع المحركات الأخرى، وخصوصًا في التطبيقات ذات القدرة المنخفضة إلى المتوسطة. ويُعزى ذلك إلى خصائصها المتميزة، مثل الكفاءة العالية، انخفاض الحاجة إلى الصيانة، الحجم الصغير، وغياب الفُرش الكربونية، بالإضافة إلى قدرتها على العمل بسرعات أعلى من المحركات التقليدية.

يهدف هذا المشروع إلى تصميم وتطوير نظام تحكم فعّال لتحسين استجابة محرك BLDC، وذلك من خلال تحسين الأداء في ما يتعلق بالسرعة والعزم واستهلاك التيار. تعتمد منهجية البحث على تطبيق وحدة تحكم هجينة من نوع PID تعتمد على المنطق الضبابي (Fuzzy-PID)، والتي تتميز بقدرتها على التعامل مع اللاخطية وتغيرات المعلمات في المحرك، ما يوفر استجابة أكثر دقة وثباتًا في بيئات التشغيل المختلفة.

لتحسين أداء وحدة التحكم بشكل أكبر، تم استخدام خوارزمية "جايا" (Jaya) لتوليف المعاملات المثلى لنظام التحكم. وتُعد هذه الخوارزمية من الخوارزميات الميتاهيوريستية المعتمدة على التجمعات، وتمتاز ببساطتها وفعاليتها في الوصول إلى حلول دقيقة دون الحاجة إلى إعدادات معقدة.

تم تقييم أداء النظام المقترح من خلال المحاكاة الحاسوبية، حيث أظهرت النتائج تحسنًا ملموسًا في زمن الاستجابة، وتقليل نسبة التجاوز، وخطأ الحالة المستقرة مقارنة بأنظمة التحكم التقليدية من نوع PID أو تلك المعتمدة فقط على المنطق الضبابي.

تُبرز هذه الدراسة أهمية دمج تقنيات الذكاء الاصطناعي والتحسين في أنظمة التحكم الحديثة، وتُسهم في تطوير استراتيجيات أكثر كفاءة للتحكم في سرعة محركات BLDC، بما يدعم موثوقية واستقرار الأنظمة في التطبيقات الصناعية والواقعية.

الكلمات المفتاحية: محرك تيار مستمر عديم الفُرش(BLDC) ، التحكم في السرعة، متحكمFuzzy-PID ، خوارزمية جايا، أمثلية، خوارزمية ميتاهيوريستية.

Table of Contents

Chapter II: Jaya Algorithm

Chapter III: A Fuzzy PI Speed Controller for a BLDC Motor

Introduction

A Brushless DC (BLDC) motor is a type of electric motor that has gained significant popularity due to its efficiency, reliability, and compact design. Unlike traditional brushed motors, BLDC motors rely on electronic controllers to operate, eliminating the need for physical brushes. This design choice leads to several advantages, including reduced friction, wear, and maintenance requirements, making them ideal for a wide range of applications. BLDC motors are known for their precise control and high torque-to-weight ratio. Consequently, they are commonly used in industries such as robotics, automotive (especially in electric vehicles), aerospace, home appliances, and even medical devices. Their ability to deliver smooth and efficient performance makes them a vital component in modern technological advancements.

The performance of a BLDC motor is critically dependent on the sophistication of its control system. While BLDC motors offer numerous benefits, they also present challenges such as limited fault tolerance capability, increased electromagnetic interference (EMI), increased acoustic noise, and increased torque ripples. All these challenges can be addressed by the control system, and strategies like fault-tolerant control, EMI control, and acoustic noise control enhance the motor's feasibility. Torque ripples can be reduced by improving the motor's design, but the control algorithm also contributes significantly [1].

Several control methodologies are employed to achieve precise speed control, with three various methods being commonly used [2]:

- **Trapezoidal Control:** This is the simplest method for powering a BLDC motor by energizing each phase in sequence. In this method, coils are energized in either a high or low state, or they can be left floating. While broadly applicable, this technique is often not as effective as more advanced methods and can produce audible noise.
- **Sinusoidal Control:** This method energizes each BLDC coil using variable duty-cycle PWM techniques to simulate analog outputs. This approach allows for a much

smoother transition between states by using a lookup table to determine the correct signal. The coils are often energized in a saddle pattern rather than a pure sinusoidal output.

- **Field-Oriented Control (FOC):** This method works similarly to sinusoidal control but also considers the motor's changing winding currents when calculating voltage inputs. FOC is the most efficient way to drive a BLDC motor, as it can produce constant torque and speeds with low acoustic noise.

Among the various control strategies, the PID control system is widely used in industrial control. The success of the PID controller is attributed to its good performance and functional simplicity, which allows engineers to operate it in a simple and straightforward manner. However, PID controllers have several limitations: the tuning is carried out offline without considering changes in the plant or disturbances, they are not adaptive, they require precise system modeling, and they often struggle to maintain performance in systems with nonlinearity, parameter variation, or external disturbances.

To overcome these limitations, the fuzzy PID controller has emerged as a dynamic and productive area of research, particularly for industrial processes that lack quantitative data for input-output relations. Fuzzy logic controllers have the advantage of being able to handle nonlinear systems and use human operator knowledge. Furthermore, they are cheaper to develop and cover a wider range of operating conditions. This thesis proposes the application of a fuzzy PI controller, optimized using the Jaya algorithm, to enhance the speed control of a BLDC motor and benchmarks its performance against a conventional PI controller also optimized with the same algorithm.

Thesis Organization

The research presented in this thesis is organized into three chapters. The work carried out in each chapter is summarized as follows:

- **Chapter 1:** This chapter delves into the fundamental principles and architecture of fuzzy logic control. We will explore the key components of a Fuzzy Inference System,

including membership functions, the fuzzification process, the structure and formulation of the rule base, various fuzzy inference mechanisms, and finally, defuzzification methods.

- **Chapter 2:** This chapter provides a comprehensive overview of the Jaya optimization algorithm.
- **Chapter 3:** This chapter focuses on the design of a speed controller for the BLDC motor using both a conventional PI controller and a fuzzy PI controller. The parameters for both controllers are optimized using the Jaya algorithm. Simulation results are presented to validate the proposed algorithms and demonstrate the performance improvements achieved.

Chapter I:

Fuzzy Logic Control

I.1. Introduction

Fuzzy Logic Control (FLC) has emerged as a powerful and versatile approach for managing complex systems, particularly those where mathematical models are difficult to derive or where human expertise plays a significant role. Unlike traditional control systems that rely on precise mathematical models and binary logic (true or false, 0 or 1), FLC embraces the concept of "degrees of truth," allowing it to handle imprecise, vague, or uncertain information – characteristics inherent in many real-world scenarios. This approach, rooted in Lotfi Zadeh's seminal work on fuzzy sets in 1965, provides a framework for mimicking human-like reasoning and decision-making in control applications [3].

The core idea behind FLC is to represent system behavior and control strategies using linguistic variables and fuzzy IF-THEN rules. These linguistic variables, such as "temperature is high" or "speed is low," are defined by fuzzy sets, which are characterized by membership functions. Membership functions quantify the degree to which a crisp (numerical) value belongs to a fuzzy set, ranging from 0 (no membership) to 1 (full membership). This allows for a more nuanced representation of system states and control actions compared to conventional control methods.

The appeal of FLC lies in its intuitive nature, its ability to incorporate expert knowledge directly into the control design, and its robustness in the face of system nonlinearities and uncertainties. It has found successful applications across a wide spectrum of fields, including consumer electronics like washing machines, air conditioners, automotive systems like anti-lock braking systems, cruise control, industrial process control including chemical processes, robotics, and decision support systems [4].

This chapter delves into the fundamental principles and architecture of fuzzy logic control. We will explore the key components of a Fuzzy Inference System (FIS), which forms the heart of any FLC. This includes a detailed examination of membership functions, the fuzzification process (converting crisp inputs into fuzzy sets), the structure and formulation of the rule base (the knowledge core of the FLC), various fuzzy inference mechanisms (how rules are fired and combined), and finally, defuzzification methods (transforming the fuzzy output

back into a crisp control signal). Through illustrations, equations, and clear explanations, this chapter aims to provide a comprehensive understanding of how fuzzy logic controllers are designed and how they operate to achieve intelligent control.

I.2. The Structure of a Fuzzy Inference System

A Fuzzy Inference System (FIS) is the computational framework that uses fuzzy set theory and fuzzy logic to map input variables to output variables. It forms the core of a fuzzy logic controller. The general structure of a FIS, as illustrated in Figure I.1, consists of four main components: the fuzzification interface, the knowledge base (which includes the rule base), the inference engine (or inference mechanism), and the defuzzification interface.

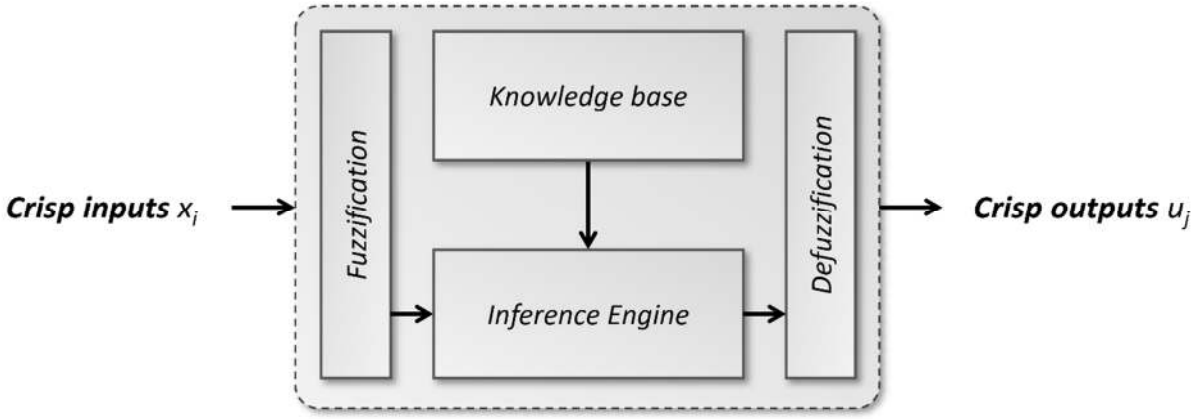

Figure I.1: General Structure of a Fuzzy Inference System

I.2.1. Membership Functions

A cornerstone of fuzzy set theory is the concept of a membership function $\mu_A(x)$. Unlike classical set theory where an element either belongs or does not belong to a set, fuzzy set theory allows for degrees of membership.[5] A membership function maps each element of a universe of discourse X to a membership degree (or membership value) in the interval [0,1]. If $\mu_A(x) = 1$, then x fully belongs to the fuzzy set A. If $\mu_A(x) = 0$, then x does not belong to A. If $0 < \mu_A(x) < 1$,then x has a partial membership in A.

Linguistic variables, such as "temperature," can take on linguistic values, like "cold," "warm," or "hot," which are represented by fuzzy sets. These fuzzy sets are, in turn, defined by their respective membership functions. The choice of membership function shape and

parameters is crucial as it influences the performance of the fuzzy logic controller. Common types of membership functions include:

I.2.1.1. Triangular Membership Function

Defined by three parameters (a, b, c), where 'a' and 'c' are the feet of the triangle and 'b' is its peak. It is computationally efficient and widely used. The formula for a triangular membership function is:

$$\mu(x; a, b, c) = \max\left(0, \min\left(\frac{x-a}{b-a}, \frac{c-x}{c-b}\right)\right) \quad (1.1)$$

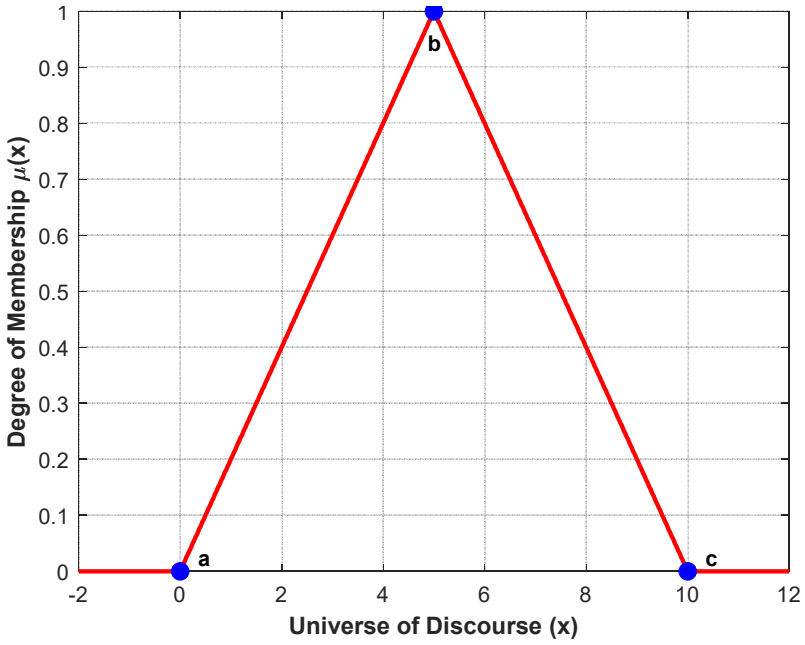

Figure I.2: Triangular Membership Function

I.2.1.2. Trapezoidal Membership Function

Defined by four parameters (a, b, c, d), where 'a' and 'd' are the feet and 'b' and 'c' define the shoulder (flat top). It is useful for representing a range of values with full membership. The formula for a trapezoidal membership function is:

$$\mu(x; a, b, c, d) = \max\left(0, \min\left(\frac{x-a}{b-a}, 1, \frac{d-x}{d-c}\right)\right) \quad (1.2)$$

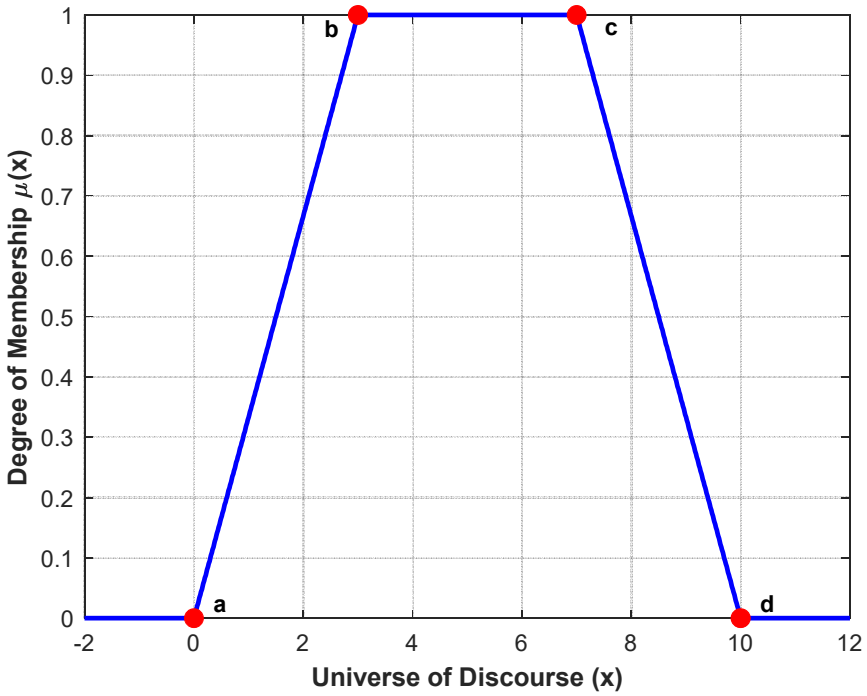

Figure I.3: Trapezoidal Membership Function

I.2.1.3. Gaussian Membership Function

Defined by two parameters: the center (c) and the width σ. It is smooth and non-zero everywhere, offering a gradual transition between degrees of membership. The formula for a Gaussian membership function is:

$$\boldsymbol{\mu(x; c, \sigma) = e^{-\frac{(x-c)^2}{2\sigma^2}}} \qquad \boldsymbol{(1.3)}$$

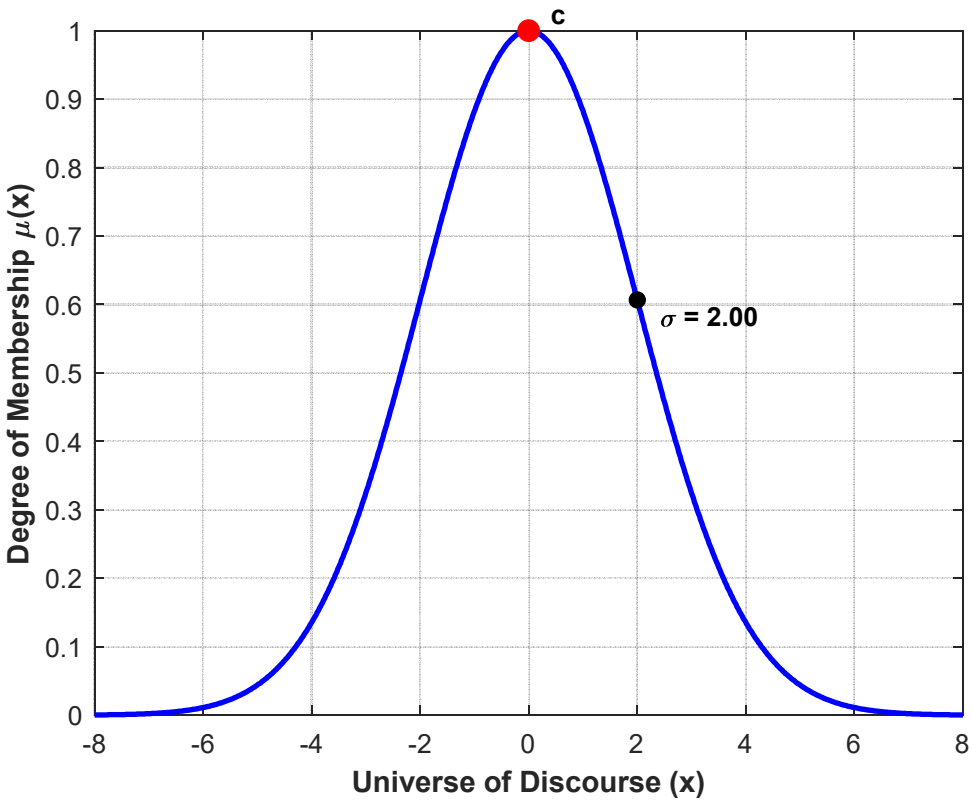

Figure I.4: Gaussian Membership Function

I.2.1.4. Generalized Bell Membership Function

Defined by three parameters (a, b, c), offering more flexibility in shape (slope and flatness of the peak) compared to the Gaussian function. The formula for a Generalized Bell membership function is:

$$\boldsymbol{\mu(x; a, b, c) = \frac{1}{1 + \left|\frac{x - c}{a}\right|^{2b}}} \qquad \boldsymbol{(1.4)}$$

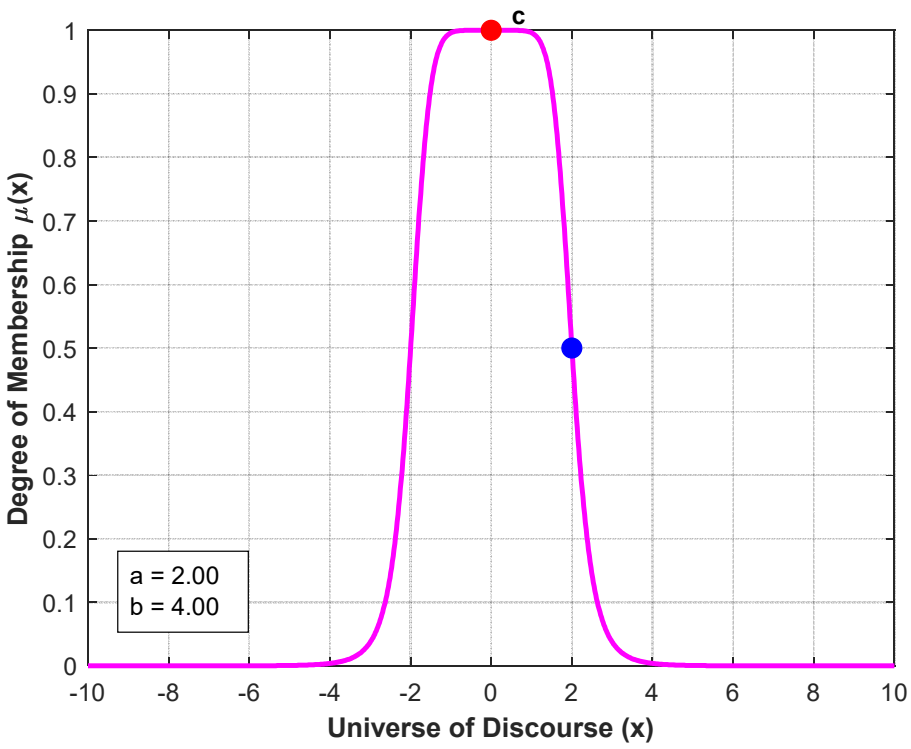

Figure I.5: Generalized Bell Membership Function

I.2.1.5. Sigmoidal Membership Function

Defined by two parameters (a, c), where 'a' controls the steepness and 'c' is the inflection point. It is useful for representing concepts like "large" or "small" where the membership transitions monotonically. The formula for a sigmoidal membership function is:

$$\mu(x; a, c) = \frac{1}{1 + e^{-a(x-c)}} \qquad (1.5)$$

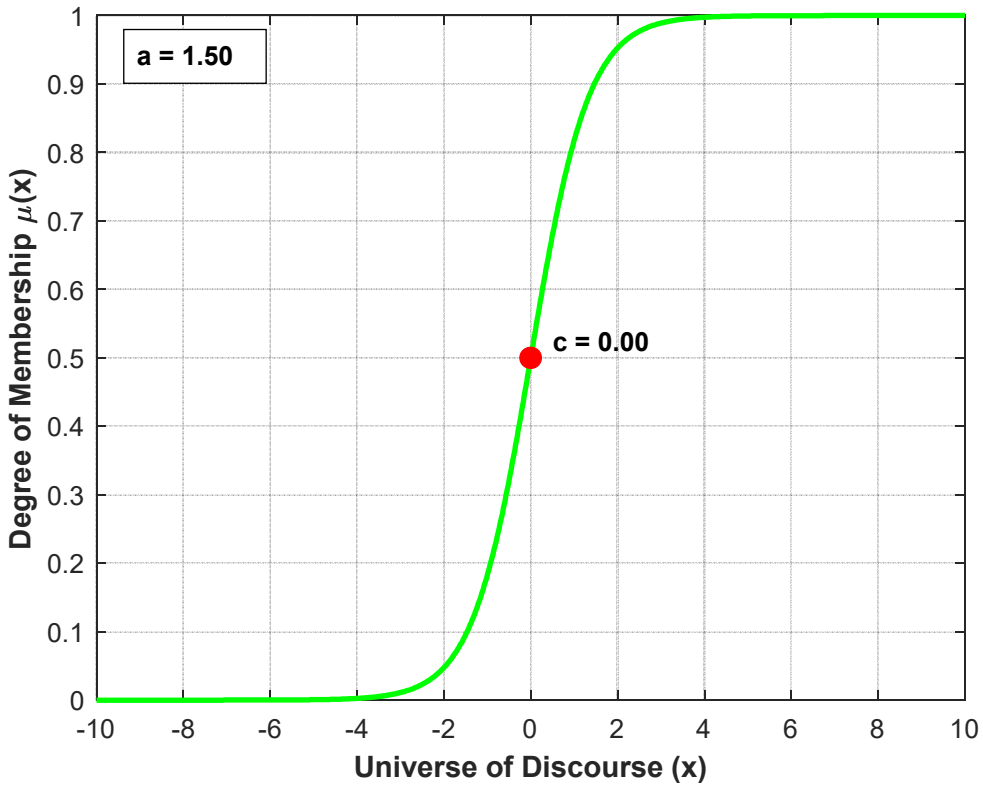

Figure I.6: Sigmoidal Membership Function

The selection of the number of membership functions for a variable and their shapes depends on the specific application, the desired control precision, and the available expert knowledge. Typically, between 3 and 7 membership functions are used for each variable.

I.2.2. Fuzzification

Fuzzification is the process of converting a crisp (numerical) input value from the real world into a fuzzy value, which involves determining the degree of membership of this crisp input to the relevant fuzzy sets. In essence, it maps a point-wise input value to a distribution of membership values across the defined linguistic terms.

For a given crisp input x_0, the fuzzifier calculates its membership degree for each fuzzy set associated with that input variable. For example, if the input variable is "temperature" and its current crisp value is $23°C$, the fuzzification process might determine that this temperature has a membership degree of 0.7 to the fuzzy set "Warm" and 0.3 to the fuzzy set "Cool," and

0.0 to "Hot" and "Cold," based on the defined membership functions for these linguistic terms.

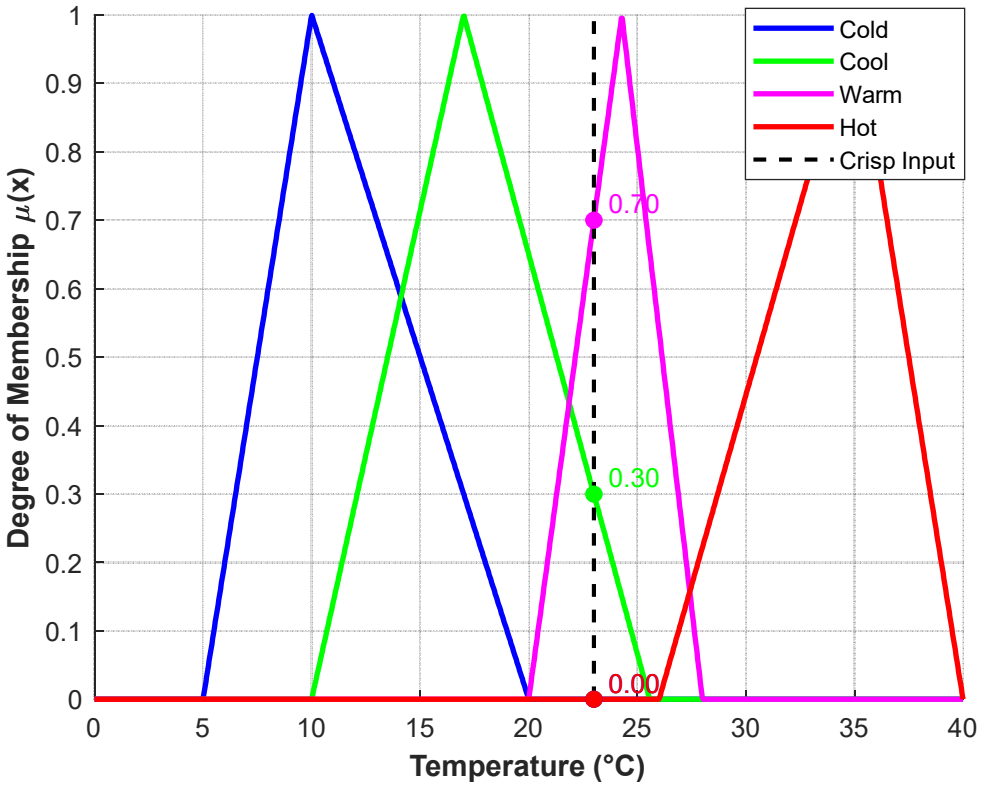

Figure I.7: Fuzzification Process

Mathematically, if x_0 is a crisp input value for a linguistic variable X, and $A_1, A_2, \ldots, A_n$ are the fuzzy sets representing the linguistic terms of X, then the fuzzified output for x_0 is the set of membership degrees: $\mu A1(x0), \mu A2(x0), \ldots, \mu An(x0)$ This set of membership degrees then serves as the input to the inference engine.

I.2.3. The Rule Base

The rule base, also known as the knowledge base in conjunction with the database of membership functions, is the heart of the fuzzy logic controller. It contains a set of fuzzy IF-THEN rules that encapsulate the expert knowledge or control strategies required to manage the system.[6] These rules define the relationship between the input fuzzy sets and the output fuzzy sets.

The general form of a fuzzy IF-THEN rule is:

IF (antecedent condition 1) & (antecedent condition 2) ... **THEN** (consequent action)

Where:

- The **antecedent** (the "IF" part) consists of one or more conditions involving linguistic variables and their fuzzy sets. These conditions describe the state of the system.
- The **consequent** (the "THEN" part) specifies the control action to be taken, also typically represented by a linguistic variable and its fuzzy set.
- The & between antecedent conditions is usually a fuzzy logical operator, such as **AND** (representing fuzzy intersection, often implemented using the MIN operator) or **OR** (representing fuzzy union, often implemented using the MAX operator).

I.2.3.1. Form of a Rule

A typical rule with two antecedents and one consequent would look like:

Rule R_i: $IF\ x_1\ is\ A_i\ AND\ x_2\ is\ B_i\ THEN\ y\ is\ C_i$

Where:

- x_1 and x_2 are input linguistic variables (like "temperature" and "humidity").
- A_i and B_i are fuzzy sets representing linguistic values for x_1 and x_2 respectively (like "High" and "Normal").
- y is the output linguistic variable (for example "fan speed").
- C_i is a fuzzy set representing the linguistic value for y (for example "Fast").

The collection of these rules forms the rule base. For example, a simple FLC for an air conditioning system might have rules like:

- IF Temperature is Hot AND Humidity is High THEN Fan Speed is Very Fast.
- IF Temperature is Warm AND Humidity is Normal THEN Fan Speed is Medium.

- IF Temperature is Cold THEN Fan Speed is Off.

The design of the rule base is a critical step and often involves:

- **Expert Knowledge Elicitation:** Interviewing domain experts to extract their control strategies.
- **System Modeling:** Analyzing the input-output behavior of the system.
- **Learning from Data:** Using machine learning techniques to automatically generate or tune rules from experimental data.[7]

The completeness and consistency of the rule base significantly impact the controller's performance.

I.2.4. Inference Mechanisms

The inference engine, or inference mechanism, is the component of the FIS that applies the fuzzy rules from the rule base to the fuzzified inputs to produce a fuzzy output. It essentially simulates human decision-making based on the provided rules. This process involves two main steps:

1. **Aggregation of Antecedents:** For each rule, the degrees of membership of the fuzzified inputs in the antecedent fuzzy sets are combined using fuzzy logical operators (typically AND or OR) to determine the overall "firing strength" or "truth value" of the rule's antecedent.
    - For an **AND** connective (e.g., IF x_1 is A AND x_2 is B), the firing strength (alpha) is often calculated using the minimum (MIN) operator:

$$\boldsymbol{\alpha = \min\left(\mu_A(x_1), \mu_B(x_2)\right)} \qquad \boldsymbol{(1.6)}$$

Another common T-norm operator for AND is the product:

$$\boldsymbol{\alpha = \mu_A(x_1) \cdot \mu_B(x_2)} \qquad \boldsymbol{(1.7)}$$

- For an **OR** connective (e.g., IF x_1 is A OR x_2 is B), the firing strength (alpha) is often calculated using the maximum (MAX) operator:

$$\boldsymbol{\alpha = \max\left(\mu_A(x_1), \mu_B(x_2)\right)} \quad \textbf{(1.8)}$$

Another common T-conorm (S-norm) operator for OR is the probabilistic sum:

$$\boldsymbol{\alpha = \mu_A(x_1) + \mu_B(x_2) - \mu_A(x_1) \cdot \mu_B(x_2)} \quad \textbf{(1.9)}$$

2. **Implication (Applying Firing Strength to Consequent):** The firing strength of each rule is then used to determine the shape of the fuzzy set for the output variable in that rule's consequent. This is known as the implication step. The most common implication methods are MIN (clipping) and PRODUCT (scaling).

- **MIN (Clipping):** The output fuzzy set of the consequent is "clipped" at the level of the rule's firing strength. If C is the fuzzy set of the consequent and alpha is the firing strength, the implied fuzzy set C' has a membership function:

$$\boldsymbol{\mu_{C'}(y) = \min\left(\alpha, \mu_C(y)\right)} \quad \textbf{(1.10)}$$

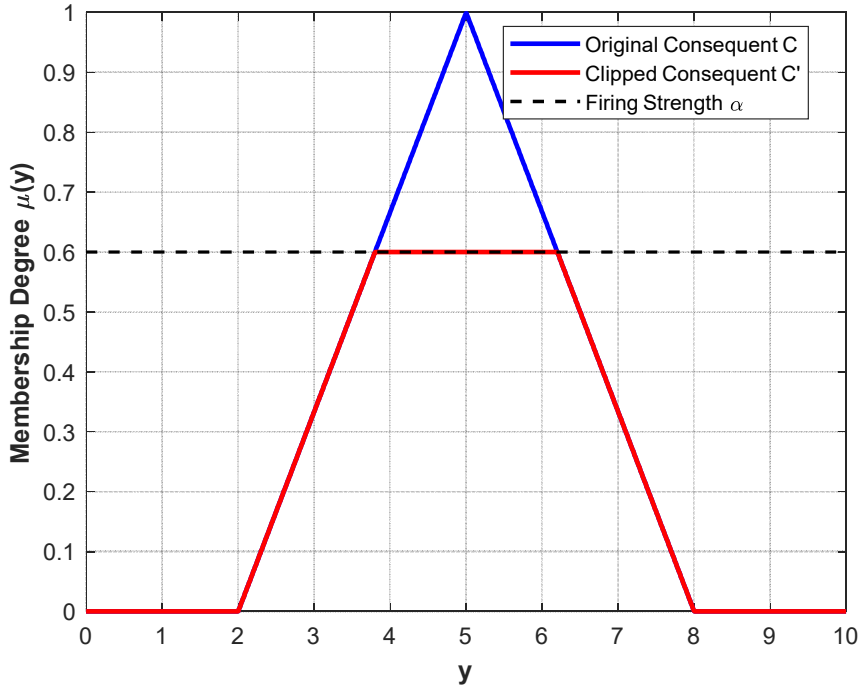

Figure I.8: MIN Implication (Clipping)

- **PRODUCT (Scaling):** The output fuzzy set of the consequent is scaled by the rule's firing strength.

$$\mu_{C'}(y) = \alpha \cdot \mu_C(y) \tag{1.11}$$

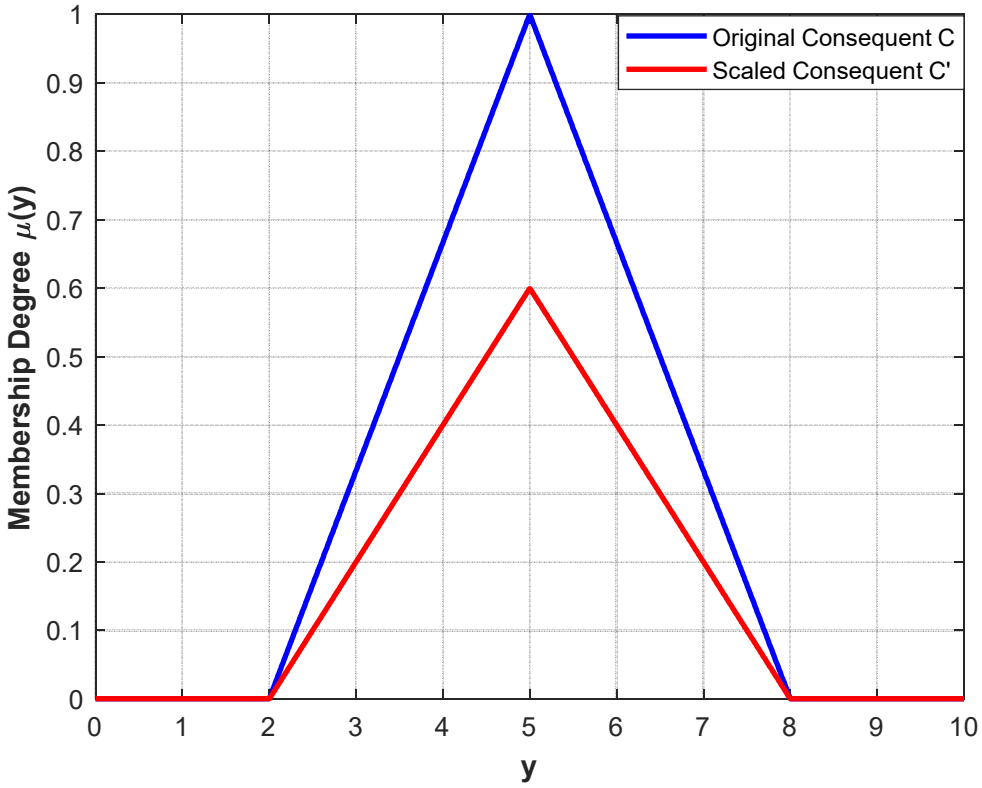

Figure I.9: PRODUCT Implication (Scaling)

There are several types of fuzzy inference mechanisms, with the most common being Mamdani, Sugeno (also known as Takagi-Sugeno-Kang or TSK), and Larsen.

I.2.4.1. Mamdani Fuzzy Inference

This is one of the earliest and most widely used inference methods [8]. It expects the output membership functions to be fuzzy sets. After the implication step for each rule, the resulting fuzzy sets (one for each rule that fires with a non-zero strength) are aggregated to form a single fuzzy set for each output variable. This aggregation typically uses the MAX operator, which takes the pointwise maximum of all the implied fuzzy sets.

$$\mu_{final_output}(y) = \max_{i=1,\dots,N} \left(\mu_{C_i'}(y) \right) \tag{1.12}$$

where N is the number of rules. The final output of the Mamdani inference process is a fuzzy set, which then needs to be defuzzified to get a crisp output.

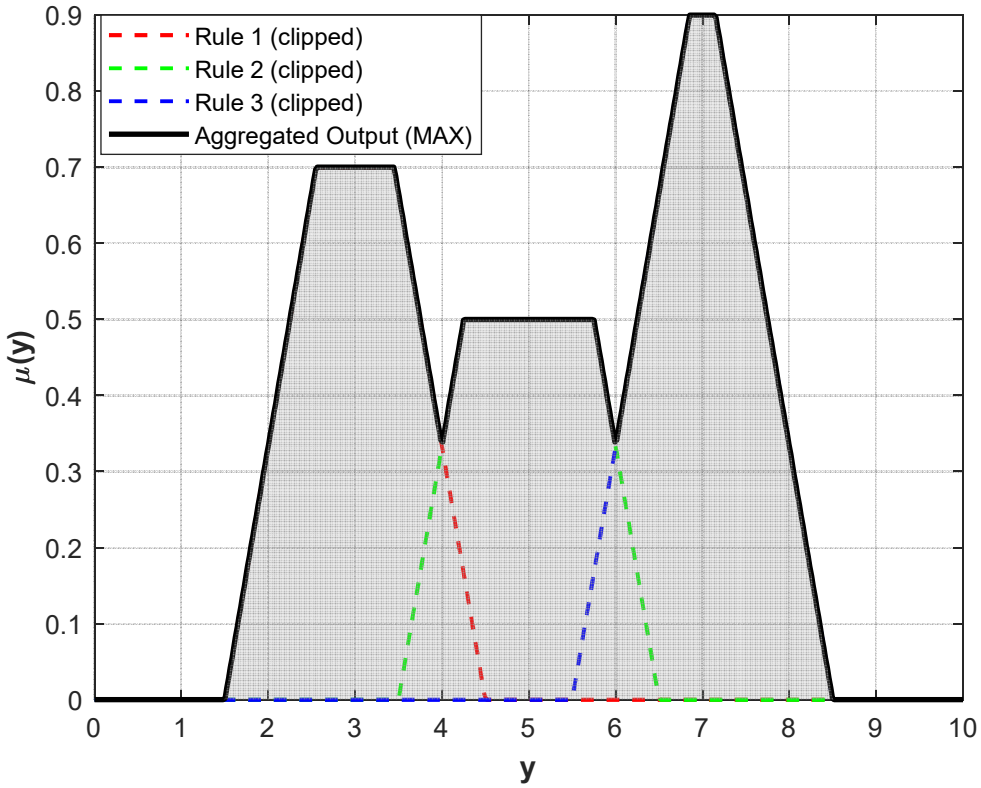

Figure I.10: Mamdani Inference Process

I.2.4.2. Sugeno Fuzzy Inference (TSK Model): Takagi & Sugeno, 1985)

In the Sugeno model, the consequent of each rule is a mathematical function (typically linear or constant) of the input variables, rather than a fuzzy set [9]. For a first-order Sugeno model, a rule has the form: IF x_1 is A AND x_2 is B THEN $y = px_1 + qx_2 + r$ Where (p,q,r) are constants. For a zero-order Sugeno model, the consequent is simply a constant: IF $x_1$is A AND $x_2$is B THEN y=k The inference process is similar in calculating the firing strength α_i for each rule R_i. However, the output of each rule (Y_i) is a crisp value calculated from its consequent function. The overall crisp output y^* is then typically computed as a weighted average of the individual rule outputs, where the weights are the firing strengths of the rules:

$$\boldsymbol{y^* = \frac{\sum_{i=1}^{N}.\alpha_i y_i}{\sum_{i=1}^{N}.\alpha_i}} \qquad (\mathbf{1.13})$$

The Sugeno model is highly efficient computationally and excels when used with optimization and adaptive techniques. It's especially good at creating smooth control surfaces. Since its output is inherently a precise, or "crisp," value (or can be directly calculated as one), the defuzzification step is often integrated directly into the inference process or handled by a simplified method like a weighted average.

I.2.4.3. Larsen Fuzzy Inference

The Larsen method is similar to Mamdani in that the output of each rule is a fuzzy set. However, it uses the product operator · for both the AND connective in the antecedent and for the implication process (scaling the consequent fuzzy set). The aggregation of the rule outputs can also be done using various operators, though sum (algebraic sum) or MAX are common. If product is used for implication, the implied fuzzy set for rule i is:

$$\boldsymbol{\mu_{C'i}(y) = \alpha_i \cdot \mu C_i(y)} \qquad \boldsymbol{(1.14)}$$

Larsen's method, particularly with product implication, often results in fuzzy output sets that preserve more information about the individual rule contributions.

The choice of inference mechanism depends on the application, the nature of the system being controlled, computational constraints, and the desired interpretability of the rules versus the precision of the output.

I.2.5. Defuzzification Methods

The output of the fuzzy inference engine (especially in Mamdani-type systems) is a fuzzy set, representing the combined result of all fired rules. However, most real-world control systems require a single, crisp numerical value as the control action (e.g., a specific voltage, speed, or angle). Defuzzification is the process of converting this aggregated fuzzy output set into a precise, crisp value.

There are several methods for defuzzification, each with its own properties and computational complexity. Some of the most common methods include:

I.2.5.1. Centroid of Area (COA) / Center of Gravity (COG)

This is the most widely used and physically appealing method. It calculates the crisp output as the x-coordinate of the center of gravity of the aggregated output fuzzy set.[10] For a continuous aggregated output fuzzy set $\mu_A(y)$, the COA is calculated as:

$$y_{COA} = \frac{\int y \cdot \mu_A(y), dy}{\int \mu_A(y), dy} \quad (1.15)$$

For a discrete universe of discourse with q quantization levels y_j:

$$y_{COA} = \frac{\sum_{j=1}^{q}. y_j \cdot \mu_A(y_j)}{\sum_{j=1}^{q}. \mu_A(y_j)} \quad (1.16)$$

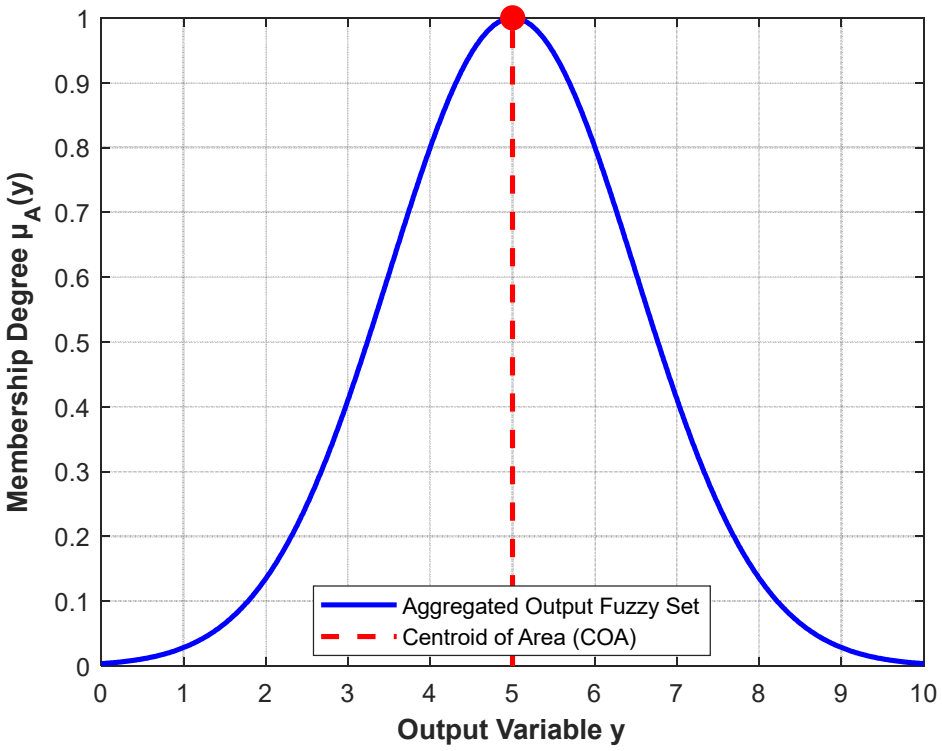

Figure I.11: Centroid of Area Defuzzification

COA provides a smooth output and considers the entire shape of the fuzzy set.

I.2.5.2. Bisector of Area (BOA)

This method finds the x-coordinate y_{BOA} that divides the area under the aggregated output fuzzy set into two equal halves.

$$\int_{-\infty}^{y_{BOA}} .\mu_A(y), dy = \int_{y_{BOA}}^{\infty} .\mu_A(y), dy \qquad (1.17)$$

BOA is generally more complex to compute than COA.

I.2.5.3. Mean of Maximum (MOM)

This method considers only the part of the output fuzzy set that has the highest membership degree (the "plateau"). The crisp output is the average of the y-values at which the membership function reaches its maximum. If there is a unique maximum, MOM is that value. If there is a plateau, it is the mean of the values on that plateau. Let

$M = y \mid mu_A(y)$ text is the maximum degree of membership.

$$y_{MOM} = \frac{\int_M . y, dy}{\int_M ., dy} \qquad (1.18)$$

For a discrete set of points where the maximum is achieved, it's the average of these points.

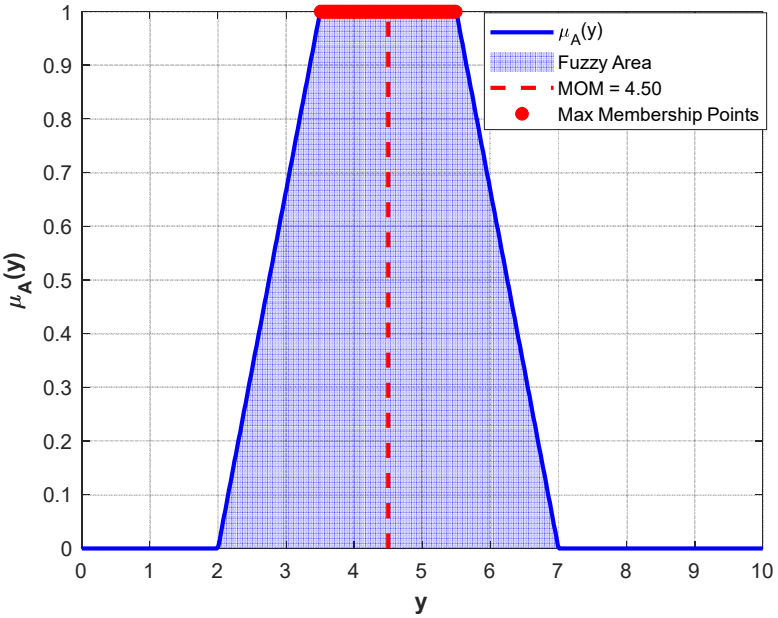

Figure I.12: Mean of Maximum Defuzzification

MOM is computationally simpler than COA but can lead to abrupt changes in output.

I.2.5.4. Smallest of Maximum (SOM)

If the maximum membership value is achieved over a range of output values (a plateau), SOM selects the smallest output value from this range.

$$y_{SOM} = \boldsymbol{min} y | \mu_A(y) = \textbf{height}(\mu_A) \quad (1.19)$$

Where height (μ_A) is the maximum membership value.

I.2.5.5. Largest of Maximum (LOM)

Similar to SOM, but LOM selects the largest output value from the range where the maximum membership is achieved.

$$y_{LOM} = \boldsymbol{max} y | \mu_A(y) = \textbf{height}(\mu_A) \quad (1.20)$$

The choice of defuzzification method can significantly affect the controller's performance, particularly its smoothness and responsiveness. COA is often preferred for its continuity and averaging properties, while MOM, SOM, and LOM are computationally faster but may not always produce the most suitable control action for dynamic systems. For Sugeno-type FIS, where the output is already a crisp value (or a weighted average of crisp values), a separate defuzzification stage in the sense of these methods is not typically required.

I.3. Conclusion

Fuzzy Logic Control provides a robust and intuitive framework for designing control systems, especially when dealing with complex, non-linear systems or when human expert knowledge is a primary source of control strategy. This chapter has detailed the fundamental architecture of a Fuzzy Inference System, which is the engine behind any FLC.

We began by understanding the crucial role of **membership functions** in defining linguistic variables and quantifying the degree to which a crisp value belongs to a fuzzy set. The choice of shape (triangular, trapezoidal, Gaussian, etc.) and parameters of these functions is a key design consideration, directly impacting the system's behavior.

The **fuzzification** process, which translates crisp sensor readings into fuzzy inputs understandable by the inference engine, was then explained. This step is essential for bridging the gap between the real world and the fuzzy domain.

The heart of the FLC, the **rule base**, was presented as a collection of IF-THEN rules that encapsulate the control logic. The structure of these rules, often derived from expert knowledge or system data, dictates how input conditions are mapped to output control actions. The use of linguistic variables makes these rules easily interpretable.

We then explored different **inference mechanisms**, primarily Mamdani, Sugeno, and Larsen, which determine how the fuzzy rules are fired and their outputs combined. The Mamdani method, with its fuzzy consequents and aggregation, offers high interpretability, while the Sugeno model, with its functional consequents, is computationally efficient and well-suited for integration with adaptive techniques.

Finally, the **defuzzification** stage, necessary for converting the aggregated fuzzy output from the inference engine back into a crisp, actionable control signal, was discussed. Various methods, including Centroid of Area (COA), Bisector of Area (BOA), and Mean of Maximum (MOM), each offer different trade-offs between computational load, smoothness, and responsiveness.

In summary, the power of FLC lies in its ability to systematically translate vague human reasoning and heuristic knowledge into an operational control strategy.[11] The careful design and tuning of its component's membership functions, rule base, inference mechanism, and defuzzification method are critical for achieving desired control performance. As will be demonstrated in subsequent chapters (or as can be seen in numerous applications), FLC provides a valuable alternative and complement to traditional control techniques, offering solutions where conventional methods may struggle. The flexibility and model-free nature (or reduced reliance on precise models) of fuzzy logic continue to drive its application in an ever-expanding range of engineering and scientific problems.[12]

Chapter II:

Jaya Algorithm

II.1 Introduction

All the evolutionary and swarm intelligence-based algorithms are probabilistic algorithms and require common controlling parameters like population size, number of generations, elite size, etc. Besides the common control parameters, different algorithms require their own algorithm-specific control parameters. For example, GA uses mutation probability, crossover probability, selection operator; PSO uses inertia weight, social and cognitive parameters; ABC uses number of onlooker bees, employed bees, scout bees and limit; HS algorithm uses harmony memory consideration rate, pitch adjusting rate, and the number of improvisations. Similarly, the other algorithms such as ES, EP, DE, SFL, ACO, FF, CSO, AIA, GSA, BBO, FPA, ALO, IWO, etc. need the tuning of respective algorithm-specific parameters. The proper tuning of the algorithm-specific parameters is a very crucial factor which affects the performance of the above-mentioned algorithms. The improper tuning of algorithm-specific parameters either increases the computational effort or yields the local optimal solution. Considering this fact, Rao et al. (2011) introduced the teaching-learning-based optimization (TLBO) algorithm which does not require any algorithm-specific parameters. The TLBO algorithm requires only common controlling parameters like population size and number of generations for its working. The TLBO algorithm has gained wide acceptance among the optimization researchers.

II.2 Definition

The Jaya algorithm is a population-based optimization technique introduced by R. Venkata Rao in 2016. Its core principle is to iteratively move candidate solutions toward the best solution and away from the worst solution within the search space. The algorithm is known for its simplicity, as it requires no algorithm-specific parameters, making it easy to implement and use across various optimization problems. Its name, "Jaya," is derived from the Sanskrit word meaning "victory," emphasizing the algorithm's goal of achieving optimal outcomes effectively.

II.3 Demonstration of the working of Jaya algorithm

The Jaya algorithm operates by iteratively improving a population of candidate solutions to optimize a given problem. Here's a simplified demonstration of its working:

1. **Initialization**:
   - Define the problem and its constraints.
   - Generate an initial population of candidate solutions randomly.
2. **Evaluation**:
   - Calculate the fitness of each candidate solution based on the objective function.
3. **Update Process**:
   - Identify the best and worst solutions in the current population.
   - Update each candidate solution to move closer to the best solution and away from the worst solution using the formula:

$$A_{\text{new}} = A_{\text{current}} + r_1 \cdot (A_{\text{best}} - | A_{\text{current}} |) - r_2 \cdot (A_{\text{worst}} - | A_{\text{current}} |)$$

   Where r_1 and r_2 are random numbers between 0 and 1.
4. **Iteration**:
   - Repeat the evaluation and update process for a predefined number of iterations or until a stopping criterion is met.
5. **Output**:
   - The best solution obtained after all iterations is considered the optimal solution.

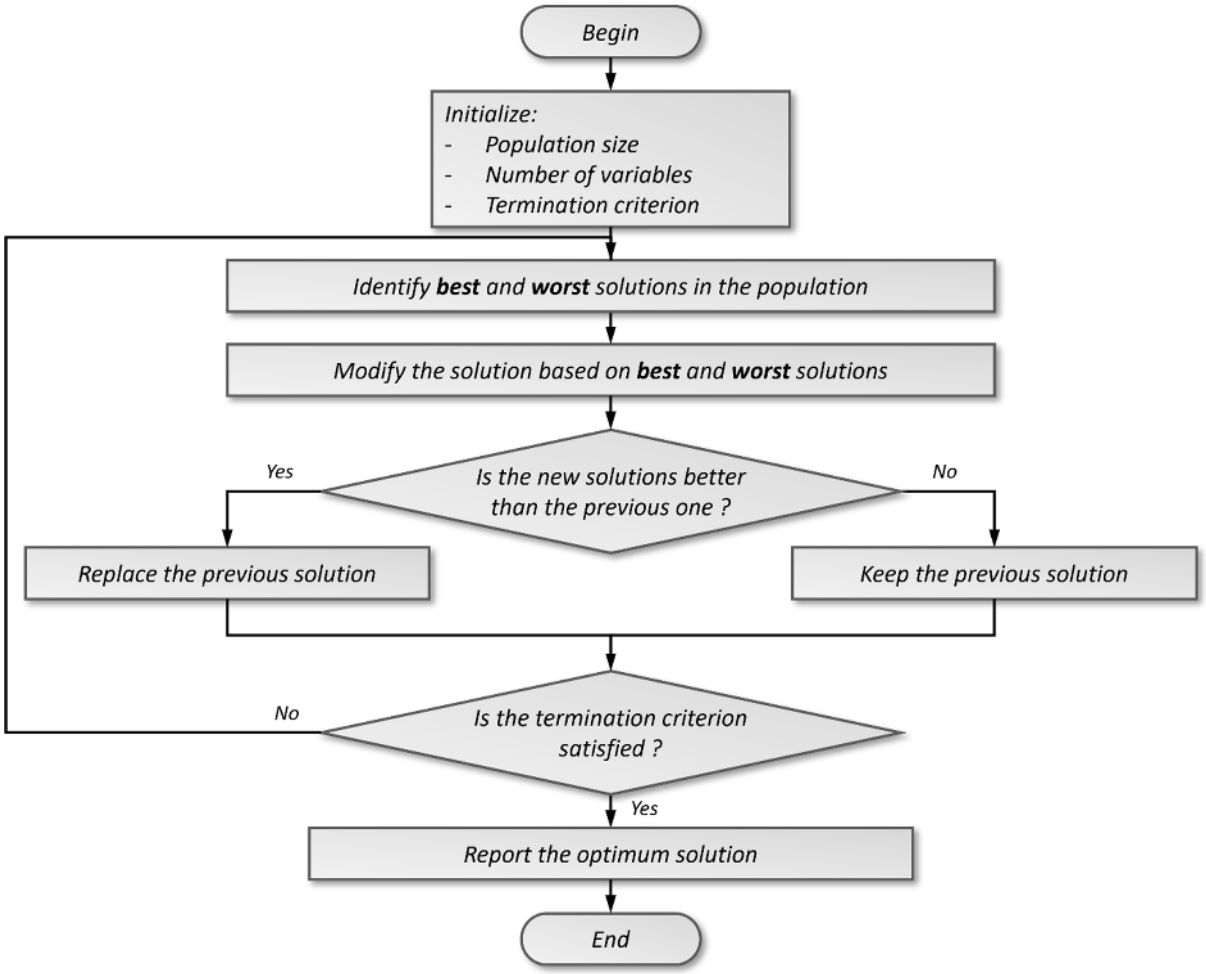

Figure II.1: Flowchart of the Jaya algorithm

II.4 Matlab code for Jaya algorithm:

```
function [bestSolution, bestFitness, convergenceCurve] =
jayaAlgorithm(objectiveFunction, lowerBound, upperBound, populationSize,
maxIterations)
% JAYAALGORITHM Implements the Jaya optimization algorithm.%
% [bestSolution, bestFitness, convergenceCurve] = jayaAlgorithm(
% objectiveFunction, lowerBound, upperBound, populationSize, maxIterations)%
% Inputs:
% objectiveFunction - Function handle for the objective function to be minimized.
% It should accept a row vector representing a solution and
% return a scalar fitness value.
% lowerBound -Row vector specifying the lower bound for each dimension of the
% solution space.
% upperBound - Row vector specifying the upper bound for each dimension
% of the solution space.
% populationSize - Integer representing the number of solutions in the population.
% maxIterations - Integer representing the maximum number of iterations.
% Outputs:
% bestSolution - Row vector representing the best solution found.
% bestFitness - Scalar value representing the fitness of the best solution.
% convergenceCurve - Column vector storing the best fitness value at each
% iteration.
```

```
% 1. Initialization
    dimension = length(lowerBound);
    population = lowerBound + rand(populationSize, dimension) .* (upperBound -
lowerBound);
    fitness = zeros(populationSize, 1);
    for i = 1:populationSize
        fitness(i) = objectiveFunction(population(i, :));
    end
    [bestFitness, bestIndex] = min(fitness);
    bestSolution = population(bestIndex, :);
    worstFitness = max(fitness);
    worstIndex = find(fitness == worstFitness, 1); % Handle cases with multiple
worst solutions.
    convergenceCurve = zeros(maxIterations, 1);
    convergenceCurve(1) = bestFitness;
    % 2. Iterations
    for iteration = 2:maxIterations
        newPopulation = zeros(populationSize, dimension);
        for i = 1:populationSize
            % Generate random numbers r1 and r2
            r1 = rand(1, dimension);
            r2 = rand(1, dimension);
            % Update the solution based on the Jaya principle
            newSolution = population(i, :) + r1 .* (bestSolution -
abs(population(i, :))) - r2 .* (population(i, :) - abs(population(worstIndex,
:)));
            % Handle boundary constraints
            newSolution = max(newSolution, lowerBound);
            newSolution = min(newSolution, upperBound);
            newPopulation(i, :) = newSolution;
        end
        % Evaluate the fitness of the new population
        newFitness = zeros(populationSize, 1);
        for i = 1:populationSize
            newFitness(i) = objectiveFunction(newPopulation(i, :));
        end
        % Selection: Keep the better solution
        for i = 1:populationSize
            if newFitness(i) < fitness(i)
                population(i, :) = newPopulation(i, :);
                fitness(i) = newFitness(i);
            end
        end
        % Update the best and worst solutions
        [currentBestFitness, currentBestIndex] = min(fitness);
        if currentBestFitness < bestFitness
            bestFitness = currentBestFitness;
            bestSolution = population(currentBestIndex, :);
        end
        currentWorstFitness = max(fitness);
        currentWorstIndex = find(fitness == currentWorstFitness, 1);
        worstFitness = currentWorstFitness;
        worstIndex = currentWorstIndex;

        convergenceCurve(iteration) = bestFitness;
    end
end
```

II.5 Key Features of the Jaya Algorithm:

- **Parameter-Free:** The algorithm does not require algorithm-specific parameters like mutation rates, crossover rates, or inertia weights, making it straightforward to implement.
- **Simplicity:** Its structure is easy to understand and use, which allows for quick adaptation to different optimization problems.
- **Flexibility:** Jaya is applicable to both constrained and unconstrained optimization problems and can be tailored to various real-world scenarios.
- **Deterministic Update Rule:** The candidate solutions are updated based on their proximity to the best and worst solutions, ensuring a focused search in the solution space.
- **Reduced Computational Cost:** As it avoids the need for parameter tuning, it is computationally efficient and requires fewer resources compared to algorithms with extensive parameter requirements.
- **Scalability:** It works effectively across problems of varying complexity and dimensions, demonstrating robustness in different contexts.

II.6 Applications:

The Jaya algorithm has been applied to a variety of real-world problems across different domains. Here are some examples:

- **Engineering Design Optimization**: Used for optimizing mechanical systems, such as gear design, structural components, and thermal systems, to improve performance and reduce costs2.
- **Energy Systems**: Applied in renewable energy optimization, including minimizing carbon emissions, reducing costs, and maximizing reliability in energy systems.
- **Manufacturing Processes**: Utilized for optimizing machining parameters, such as cutting speed and feed rate, to enhance productivity and product quality.

- **Scheduling and Logistics**: Employed in solving complex scheduling problems, such as job-shop scheduling and transportation logistics, to improve efficiency.
- **Machine Learning**: Used for hyperparameter tuning in machine learning models to achieve better predictive performance.

II.7 Conclusion:

The Jaya algorithm is a robust and efficient optimization technique known for its simplicity and versatility. By minimizing reliance on algorithm-specific parameters, it offers a user-friendly approach to solving complex optimization problems across various domains. Its iterative process of converging towards the best solution and avoiding the worst makes it a powerful tool in fields ranging from engineering to machine learning. As it continues to gain attention and application, the Jaya algorithm demonstrates its value as a reliable and adaptable method for achieving optimal outcomes.

Chapter III:

A Fuzzy PI Speed Controller for a BLDC Motor

III.1 Introduction

Brushless DC (BLDC) motors are increasingly vital in modern industries, offering superior performance and efficiency. To harness their full potential, a robust control drive is essential for precisely regulating their speed and torque [13]. Effective speed control of BLDC motors is crucial for achieving optimal performance, maximizing energy efficiency, and ensuring longevity across a wide range of applications. This control minimizes issues such as overheating, excessive wear, and energy losses by dynamically adapting to changing load conditions and operating environments.

Numerous methods are employed for BLDC motor speed control, with several prominent techniques commonly utilized:

- **Trapezoidal Commutation:** This technique is used to control three-phase permanent magnet BLDC motors by managing stator currents to achieve the desired speed and direction of rotation.
- **Sinusoidal Commutation:** In this motor control technique, BLDC motor stator windings are energized with smooth, sinusoidal currents instead of abrupt signals, resulting in more refined operation.
- **Field-Oriented Control (FOC):** An advanced control method for BLDC motors, FOC regulates the motor's magnetic field and current to achieve smooth, precise, and efficient operation.
- **Digital Voltage/Frequency Control:** A simpler, open-loop method often found in low-cost BLDC drives, which regulates speed by proportionally varying the DC-link voltage with the switching frequency. Although more common in induction motor drives, it is also applied in certain BLDC systems.
- **PID Controllers:** Proportional-Integral-Derivative (PID) controllers are widely adopted feedback mechanisms in industrial control due to their robustness and effectiveness.

This chapter details the design of a speed controller for the BLDC motor, specifically focusing on both a conventional PI controller and a Fuzzy PI controller. The parameters for both controllers are optimized using the Jaya algorithm, and simulation results are presented to validate the proposed approaches and demonstrate the achieved performance improvements.

III.2 The BLDC motor

A Brushless DC (BLDC) motor is an electric motor characterized by its high efficiency, reliability, and compact design. Unlike brushed DC motors, BLDC motors operate without physical brushes, relying on electronic commutation instead. This design eliminates friction, wear, and maintenance requirements, making them ideal for a wide range of applications [14]. BLDC motors are renowned for their precise control capabilities and high torque-to-weight ratio. Consequently, they are widely used in industries such as robotics, automotive (especially electric vehicles), aerospace, home appliances, and medical devices. Their ability to deliver smooth and efficient performance makes them a vital component in modern technological advancements.

The rotor of a typical BLDC motor is constructed from permanent magnets, while the stator contains the windings. The commutation of a BLDC motor is controlled electronically, unlike a brushed DC motor. Hall sensors provide crucial information to synchronize the stator armature excitation with the rotor position. The stator windings must be energized in sequence to rotate the BLDC motor effectively [15,16].

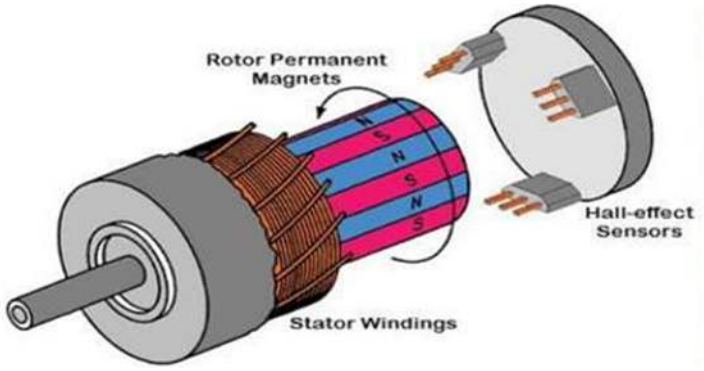

Figure III.1: BLDC motor.

III.2.1 Working principle and operations

The fundamental principle behind a BLDC motor is to adjust the energized phase windings based on the detected position of the rotor's permanent magnet, thereby ensuring continuous torque production. To achieve this, obtaining accurate information about the rotor magnet's position is crucial, a task typically accomplished using Hall effect sensors.

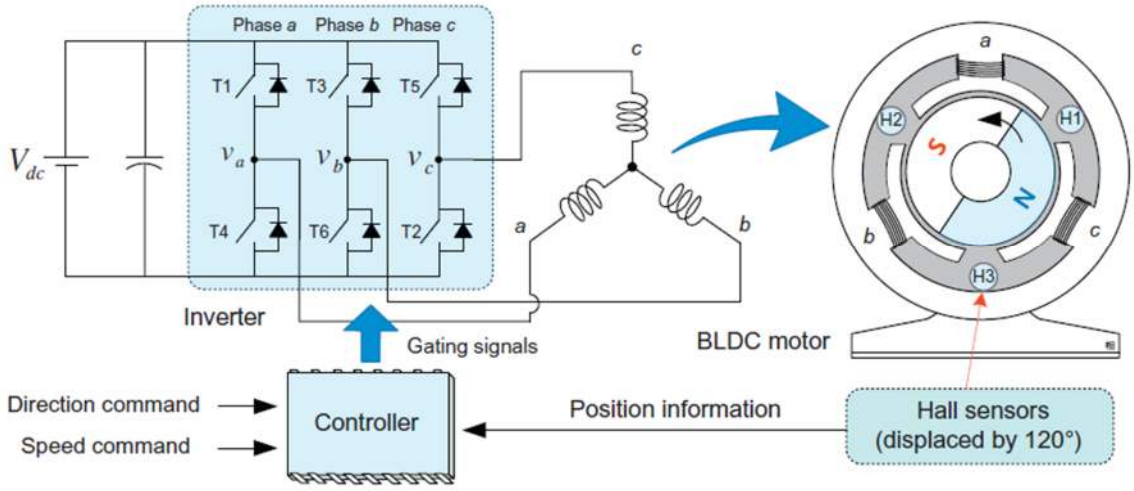

Figure III.2: BLDC motor drive system.

III.2.2 Mathematical model of the BLDC motor

In this section, we will derive the mathematical model of a three-phase BLDC motor. The model of a BLDC motor is similar to that of a PMSM due to their structural similarity [17].

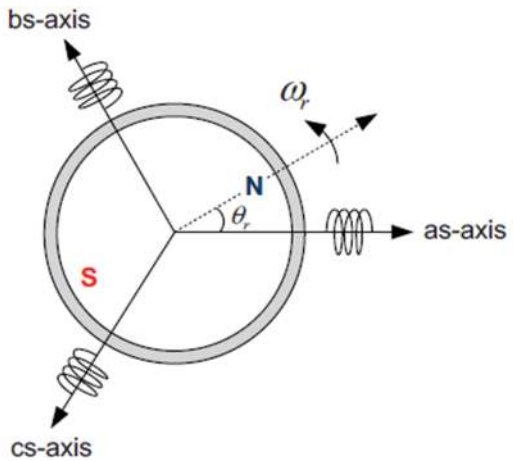

Figure III.3: Windings of a two-pole three-phase BLDC motor.

III.2.2.1 Voltage equations

Let us consider a two-pole three-phase BLDC motor as shown in Figure III.3 the voltage equation for the stator windings of a BLDC motor can be expressed as:

$$\boldsymbol{v_{abcs} = R_s i_{abcs} + \frac{d\lambda_{abcs}}{dt}} \qquad \textbf{(III. 1)}$$

Where the stator voltage $\boldsymbol{v_{abcs} = [v_{as} v_{bs} v_{cs}]^T}$, the stator current $\boldsymbol{i_{abcs} = [i_{as} i_{bs} i_{cs}]^T}$, the stator flux linkage $\boldsymbol{\lambda_{abcs} = [\lambda_{as} \lambda_{bs} \lambda_{cs}]^T}$, and the stator resistance

$$R_S = \begin{bmatrix} R_s & 0 & 0 \\ 0 & R_s & 0 \\ 0 & 0 & R_s \end{bmatrix}$$

The flux linkage $\boldsymbol{\lambda_{abcs}}$ of the stator windings consists of $\boldsymbol{\lambda_{abcs(s)}}$ due to the stator currents $\boldsymbol{i_{abcs}}$ and $\boldsymbol{\lambda_{abcs(f)}}$ due to the permanent magnet as:

$$\boldsymbol{\lambda_{abcs} = \lambda_{abcs(s)} + \lambda_{abcs(f)}} \qquad \textbf{(III. 2)}$$

Substituting Eq. **(III.2)** into Eq. **(III.1)** gives the following stator voltage equation.

$$\boldsymbol{v_{abcs} = R_s i_{abcs} + \frac{d\lambda_{abcs}}{dt} = R_s i_{abcs} + \frac{d\lambda_{abc(s)}}{dt} + e_{abcs}} \qquad \textbf{(III. 3)}$$

where the back-EMF due to the magnet flux is expressed as $\boldsymbol{e_{abcs} = d\lambda_{abc(f)}/dt}$

and is also given as:

$$\boldsymbol{e_{abcs} = \begin{bmatrix} e_{as} \\ e_{bs} \\ e_{ac} \end{bmatrix} = \omega_m \begin{bmatrix} \lambda_{asf} \\ \lambda_{bsf} \\ \lambda_{csf} \end{bmatrix} = \omega_m \lambda_f \begin{bmatrix} f(\theta_r) \\ f\left(\theta_r - \frac{2\pi}{3}\right) \\ f\left(\theta_r - \frac{2\pi}{3}\right) \end{bmatrix}} \qquad \textbf{(III. 4)}$$

where $\boldsymbol{\lambda_f = (N\phi_f)}$ is the amount of the magnet flux $\boldsymbol{\phi_f}$ linking N turns of the stator windings, $\boldsymbol{f(\theta_r)}$ is a unit function representing the waveform of the back-EMF and $\boldsymbol{\theta_r}$ is the rotor position.

The stator flux linkage $\boldsymbol{\lambda_{abcs(s)}}$ due to the stator currents is given by

$$\boldsymbol{\lambda_{abcs(s)} = L_s i_{abcs}} = \begin{bmatrix} \boldsymbol{L_{aa}} & \boldsymbol{L_{ab}} & \boldsymbol{L_{ac}} \\ \boldsymbol{L_{ba}} & \boldsymbol{L_{bb}} & \boldsymbol{L_{bc}} \\ \boldsymbol{L_{ca}} & \boldsymbol{L_{cb}} & \boldsymbol{L_{cc}} \end{bmatrix} \begin{bmatrix} \boldsymbol{i_{as}} \\ \boldsymbol{i_{bs}} \\ \boldsymbol{i_{cs}} \end{bmatrix} \quad \textbf{(III. 5)}$$

For symmetry three-phase windings, the self-inductances are all the same and the mutual-inductances are all the same as in the following

$$\boldsymbol{L_{aa} = L_{bb} = L_{cc} = L_s = L_{ls} + L_m} \quad \textbf{(III. 6)}$$

$$\boldsymbol{L_{ab} = L_{ac} = L_{ba} = L_{bc} = L_{ca} = L_{cb} = -\frac{1}{2} L_m = M} \quad \textbf{(III. 7)}$$

Where $\boldsymbol{L_{\alpha\beta} = (\lambda / i_\beta)}$ expresses the winding inductance, which is the ratio of the flux linkage **λ** of the winding α to the current $\boldsymbol{i_\beta}$ that produces the flux.

From equations **(III.5)** and **(III.7)**, the stator voltage equation is rewritten as

$$\boldsymbol{v_{abcs} = R_s i_{abcs} + L_{abcs} \frac{di_{abc(s)}}{dt} + e_{abcs}} \quad \textbf{(III. 8)}$$

$$\begin{bmatrix} \boldsymbol{v_{as}} \\ \boldsymbol{v_{bs}} \\ \boldsymbol{v_{cs}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{R_s} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{R_s} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{R_s} \end{bmatrix} \begin{bmatrix} \boldsymbol{i_{as}} \\ \boldsymbol{i_{bs}} \\ \boldsymbol{i_{cs}} \end{bmatrix} + \begin{bmatrix} \boldsymbol{L_s} & \boldsymbol{M} & \boldsymbol{M} \\ \boldsymbol{M} & \boldsymbol{L_s} & \boldsymbol{M} \\ \boldsymbol{M} & \boldsymbol{M} & \boldsymbol{L_s} \end{bmatrix} \frac{\boldsymbol{d}}{\boldsymbol{dt}} \begin{bmatrix} \boldsymbol{i_{as}} \\ \boldsymbol{i_{bs}} \\ \boldsymbol{i_{cs}} \end{bmatrix} + \begin{bmatrix} \boldsymbol{e_{as}} \\ \boldsymbol{e_{bs}} \\ \boldsymbol{e_{cs}} \end{bmatrix} \quad \textbf{(III. 9)}$$

here, since $\boldsymbol{i_{as} + i_{bs} + i_{cs} = 0}$, the mid-term of Eq. **(III.9)** is reduced as

$$\boldsymbol{\frac{d}{dt}[L_s i_{as} + M i_{bs} + M i_{cs}] = \frac{d}{dt}[L_s i_{as} - M i_{as}]} \textbf{ (III.10)}$$

Thus Eq. **(III.9)** becomes the following voltage equations.

$$\begin{bmatrix} \boldsymbol{v_{as}} \\ \boldsymbol{v_{bs}} \\ \boldsymbol{v_{cs}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{R_s} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{R_s} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{R_s} \end{bmatrix} \begin{bmatrix} \boldsymbol{i_{as}} \\ \boldsymbol{i_{bs}} \\ \boldsymbol{i_{cs}} \end{bmatrix} + \begin{bmatrix} \boldsymbol{L_s - M} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{L_s - M} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{L_s - M} \end{bmatrix} \frac{\boldsymbol{d}}{\boldsymbol{dt}} \begin{bmatrix} \boldsymbol{i_{as}} \\ \boldsymbol{i_{bs}} \\ \boldsymbol{i_{cs}} \end{bmatrix} + \begin{bmatrix} \boldsymbol{e_{as}} \\ \boldsymbol{e_{bs}} \\ \boldsymbol{e_{cs}} \end{bmatrix} \quad \textbf{(III. 11)}$$

Voltage Equations of a Brushless Direct Current Motor:

$$\boldsymbol{v_{as} = R_s i_{as} + (L_s - M) \frac{di_{as}}{dt} + e_{as}} \quad \textbf{(III. 12)}$$

$$\boldsymbol{v_{bs} = R_s i_{bs} + (L_s - M) \frac{di_{bs}}{dt} + e_{bs}} \quad \textbf{(III. 13)}$$

$$v_{cs} = R_s i_{cs} + (L_s - M)\frac{di_{cs}}{dt} + e_{cs} \quad (\mathbf{III.14})$$

The electromagnetic torque of three-phase BLDC motor can be described by the following four equations:

$$T_e = \frac{1}{\omega_r}(E_a i_a + E_b i_b + E_c i_c) \quad (\mathbf{III.15})$$

The generated back-EMF is trapezoidal in shape due to permanent magnet straddling on the rotor and its expression is given below:

$$E_a = K_e * \frac{d}{dt}(\theta) * \omega_r(t) \quad (\mathbf{III.16})$$

$$E_b = K_e * \frac{d}{dt}\left(\theta - \frac{2\pi}{3}\right) * \omega_r(t) \quad (\mathbf{III.17})$$

$$E_c = K_e * \frac{d}{dt}\left(\theta + \frac{2\pi}{3}\right) * \omega_r(t) \quad (\mathbf{III.18})$$

III.2.2.2 The mechanical Equations

The mechanical equation of the BLDC can be written as follows:

$$\mathrm{J}\frac{\mathrm{d}}{\mathrm{dt}}\omega_{\mathrm{m}} = \mathrm{T_e} - \mathrm{T_1} - f\omega_{\mathrm{m}} \quad (\mathbf{III.19})$$

III.3 The structure of the controller

The figure below shows the basic of the controller containing here we use the PI controller, and we replace it with Fuzzy PI to do our tests.

As shown in Figure III.4 a presenting of the controller containing, below we'll explain the roles of each controller containing:

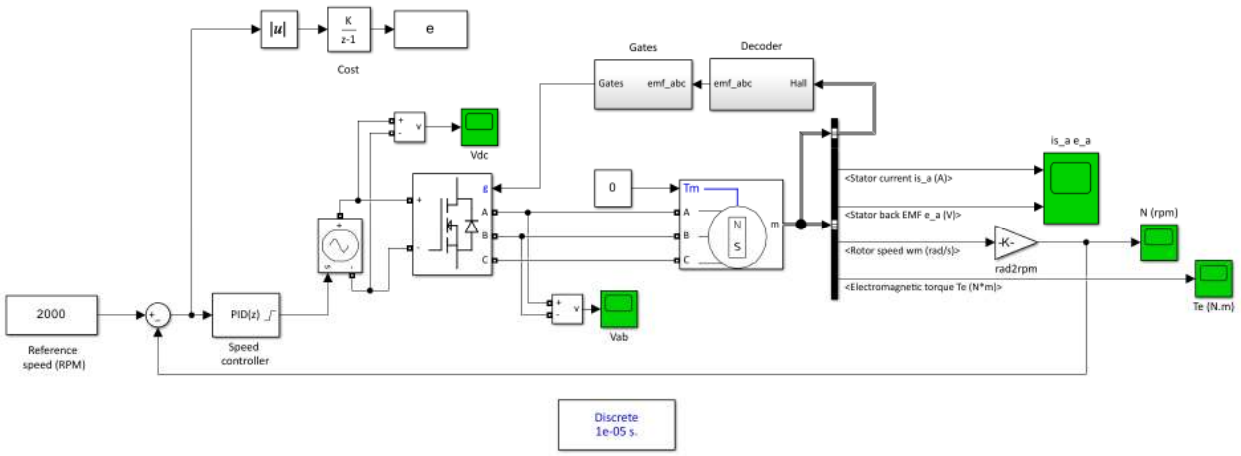

Figure III.4. The controller architecture.

III.3.1 Speed controller

The controller (a conventional PI or a fuzzy PI) uses two input the error e and its rate $\Delta\mathbf{e}$ to produce an output to regulate motor speed which is a control signal u by using the roles base to decide whether to output low, medium, or high.

$$e_{\omega} = \mathrm{RPM}_{\mathrm{Req}} - \mathrm{rpm} \qquad (\mathbf{III.20})$$

$$\Delta e_{\omega} = e_{\omega}(k) - e_{\omega}(k-1) \qquad (\mathbf{III.21})$$

The output of the controller is limited by a Saturation block. Its job is to clip ***u*** to a safe range by prevents runaway commands (for example if the output tries to go above what the inverter/DC bus can physically supply).

III.3.2 Controlled voltage source

The output of Saturation is the input of the Controlled Voltage Source. The role of it is to translate the scalar command u into a DC-link amplitude V_{dc} which feeding the inverter to generate the phase voltages.

III.3.3 Commutation logic

The Decoder generates switching signals based on rotor position which reserving from motor's (Hall or EMF) sensers to updates the gate pattern. The decoder outputs a 6-bit "gate" vector (e.g., [1 0; 0 1; 0 0] meaning "A+, A−, B+, B−, C+, C−" which mean that the rotor is in

the sector 1(0° – 60°)). There are six valid “states” in a 3-phase BLDC (every 60 electrical degrees), and each state energizes two of the three windings.

Those gate signals feed directly into the Universal Bridge so that the inverter always energizes the correct phases in the proper sequence.

III.3.4 Three-phase inverter

A “Universal Bridge” typically represents A three-phase inverter takes the DC-link voltage V_{dc} and, via six power switches (typically MOSFETs or IGBTs), synthesizes three AC output legs 120° apart by activating these switches in coordination with Hall-sensor feedback, the inverter directs current into the appropriate stator windings at the right times, producing the rotating field required by the BLDC motor.

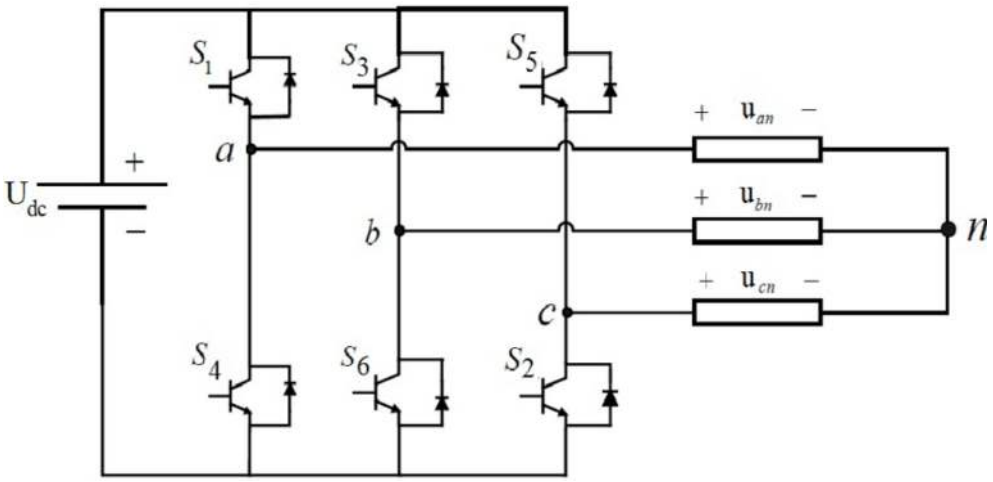

Figure III.5: Three phase inverter circuit.

III.4 The PID type fuzzy controller

Figure III.6 illustrates a typical closed-loop control system incorporating a standard ‘PID type FLC’ [18]. The control signal generated by the ‘PID type FLC’ is determined by:

$$\boldsymbol{u = \alpha U + \beta \int U dt} \qquad \textbf{(III.22)}$$

Where U is the output of the FLC bloc.

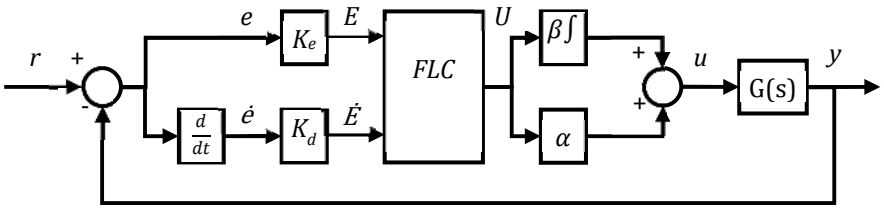

Figure III.6: Architecture of PID type FLC.

As demonstrated in [19], FLCs employing product-sum inference, center-of-gravity defuzzification, and triangular input 'membership functions' with a crisp output exhibit an input-output relationship governed by:

$$\boldsymbol{U = A + PE + D\dot{E}} \quad \textbf{(III.23)}$$

Where $E = K_e e$ and $\dot{E} = K_d \dot{e}$.

Consistent findings in [19] confirm that the minimum inference engine also produces the same input-output relationship. Combining equations (**III.22**) and (**III.23**) yields the controller output as follows:

$$\boldsymbol{u = \alpha A + \beta A t + \alpha K_e P e + \beta K_d D e + \beta K_e P \int e dt + \alpha K_d D \dot{e}} \quad \textbf{(III.24)}$$

As a result, the equivalent PID-type FLC control components are derived as follows:

Proportional gain: $\alpha K_e P \beta K_d D$

Integral gain: $\beta K_e P$

Derivative gain: $\alpha K_d D$.

III.4.1 The fuzzy inference system

There are many design factors in a fuzzy logic controller determining its structure, such as membership functions, input space partition by fuzzy rules, various types of fuzzy inference mechanisms, defuzzification schemes, etc. They may appear either highly nonlinear or approximately linear. Nevertheless, to perform proportional, integral and derivative control modes, the structure of a fuzzy logic controller has to be in some way analogous to a normal PID controller [20].

The **Figure III.7** below shows the membership functions of the error e and its rate **Δe**. Each input is normalized in the range [−1, 1], and the three triangular membership functions are labeled to Negative, Zore or Positive.

These functions define how the crisp inputs e and **Δe** are fuzzified**.**

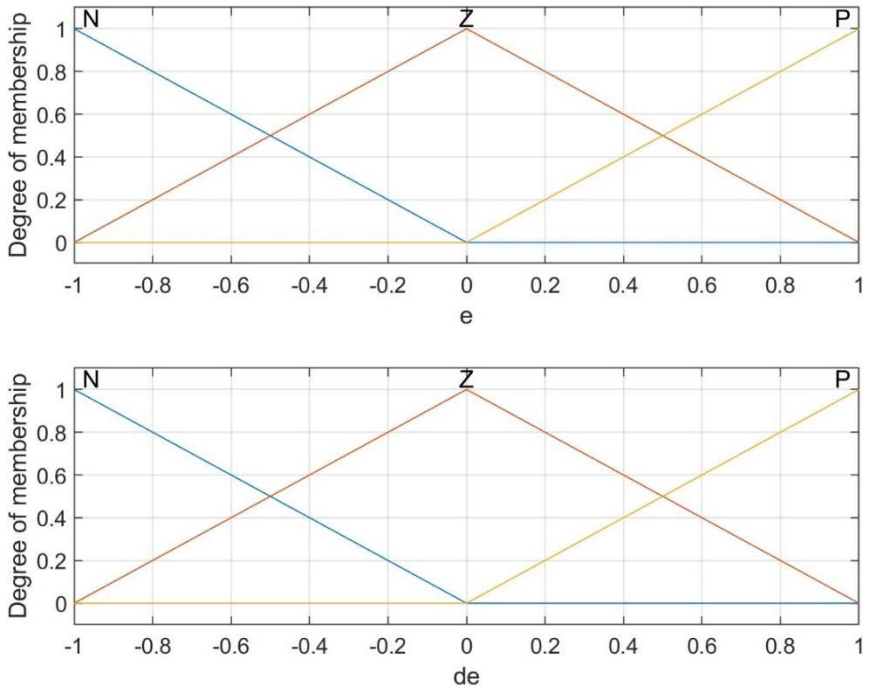

Figure III.7: The input membership functions.

The controller uses these input membership functions to activate the role base to decide what is the value of the output ***u***.

The table below represents the role base that the Fuzzy PI following and its output in each condition:

Table 3.1: Rule base for the FIS.

e \ Δe	**NE**	**ZE**	**PO**
NE	-1	-0.5	0
ZE	-0.5	0	0.5
PO	0	0.5	1

The **Figure III.8** represent the fuzzy PI controller surface it shows how the inputs error and its rate influence in the controller's output.

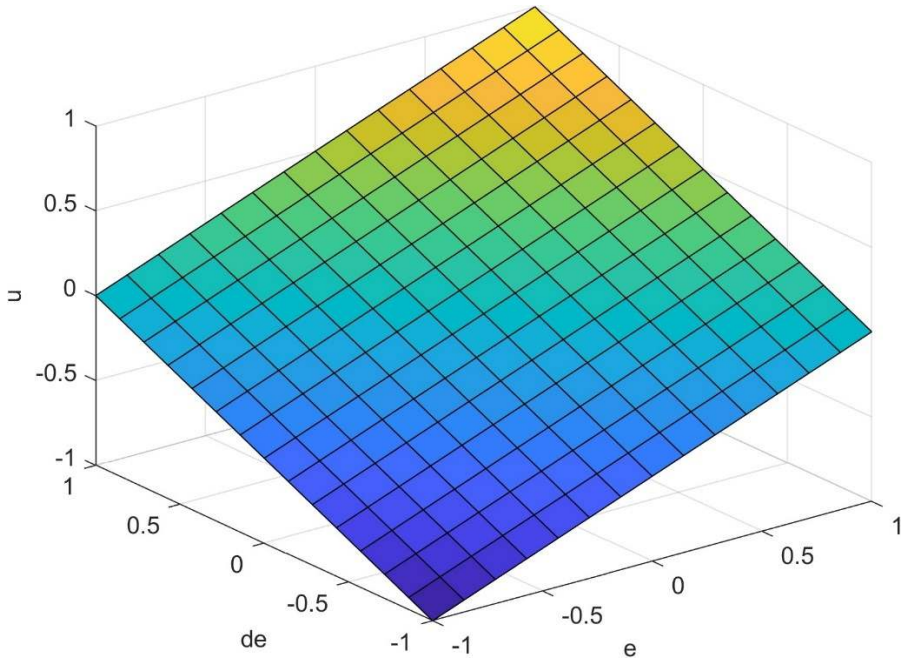

Figure III.8: Fuzzy control surface.

III.5 The optimization of the controller parameters

The cost function quantifies "how good" a particular set of controller parameters is. We use the Integrated Absolute Error (IAE) of the speed tracking error over the simulation horizon as the cost.

$$\boldsymbol{IAE = \int |e(t)| dt} \qquad \textbf{(III. 25)}$$

Jaya algorithm tries to minimize IAE. A smaller IAE means the controller setpoint was tracked more precisely over the entire simulation. Optimizing IAE balances fast rise time against overshoot.

III.5.1 Optimize the PI parameters

For optimizing the PI parameters, we use a Population size equal to 100 with a max iteration equal 2.

The lower Bound = [0.01 17] and upper Bound = [0.05 22] which mean that $k_p \epsilon [0.01\ 0.05]$ and $k_i \epsilon [17\ 22]$.

The execution time increases roughly linearly with (population Size × max Iteration).

The optimal gains are $\boldsymbol{k_p}$ **= 0.02** and $\boldsymbol{k_i} = \mathbf{19}$.

III.5.2 Optimization of the Fuzzy PI parameters

For optimizing the fuzzy PI parameters, we use a Population size equal to 100 with a max iteration equal 10.

The lower Bound = [3x10-3 0.01 140000] and upper Bound = [4x10-3 0.09 145000] which mean that $k_e \epsilon [3 * 10^{-3}\ 4 * 10^{-3}]$,$k_d \epsilon [0.01\ 0.09]$, $\beta \epsilon [140000\ 145000]$, with $\alpha = 0$.

Then the execution time increases roughly linearly with (population Size × max Iteration).

The optimal gains are $\mathbf{k_e} = \mathbf{3.33} * \mathbf{10^{-4}}, \mathbf{k_d} = \mathbf{0.0705}, \boldsymbol{\beta} = \mathbf{140000}$ and $\boldsymbol{\alpha} = \mathbf{0}$.

III.6 Simulation results

Show some simulation examples and discuss the results. The results should include difference reference signal with different load constraints. The results should also include motor parameters variation.

The parameters of BLDC motor are as follows:

3 phase motor.

Rated voltage V_{dc} =300 V,

Rated torque T_m = 0.8 Nm,

Rated speed r = 3000 rpm,

Stator phase resistance Rs = 2.875 Ω,

Stator phase inductance Ls = 8.5x10^{-3} H,

Voltage constant Kv = 86.6271,

Torque constant Kt = 0.7164.

III.6.1 Case 1: Reference Tracking (No Load)

This case presents a comparative speed response of a BLDC motor using two different control strategies a classical PI controller and a Fuzzy PI controller.

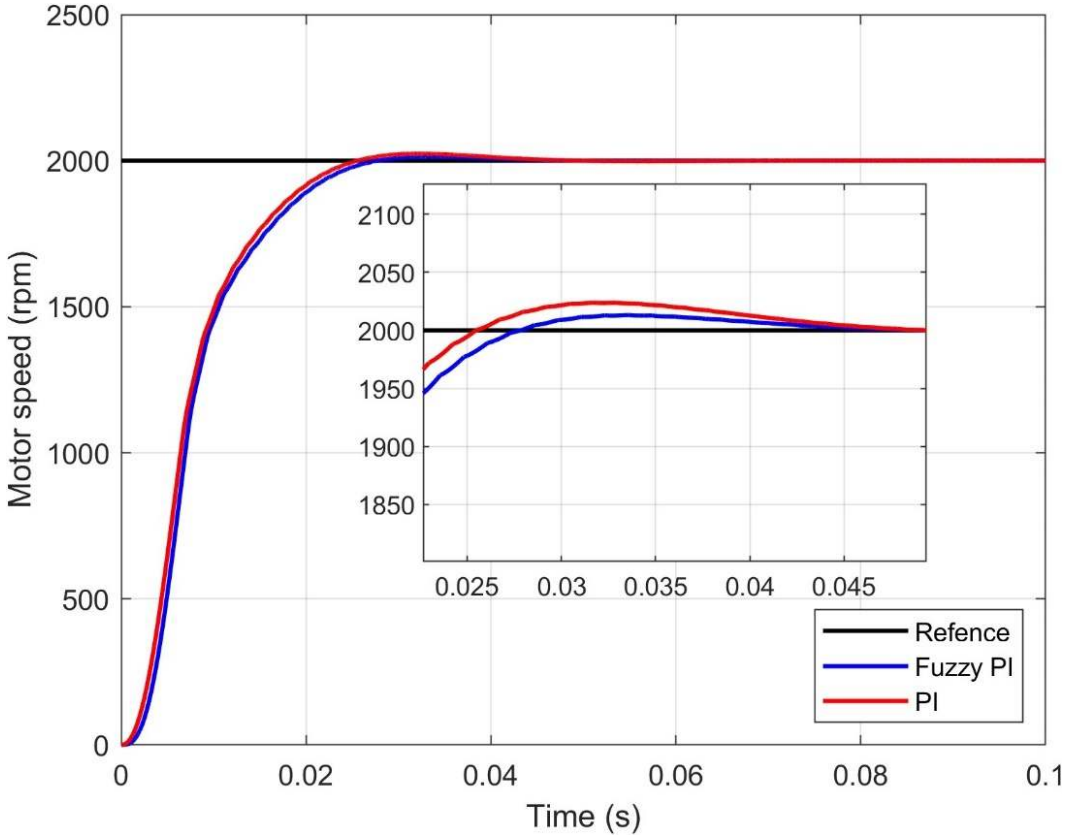

Figure III.9: Response of the BLDC speed motor case1.

Table 3.2: The Response Characteristics.

	Rise Time	**Overshoot**	**Settling Time**	**Absolute Error**
PI	0.0132	1.1967	0.0223	0
Fuzzy PI	0.0135	0.6596	0.0235	0

Figure.9 represent the speed response of a BLDC motor with two different speed controllers. The motor speed increase from 0rpm to the reference speed 2000rpm in 0.0235s for Fuzzy PI and 0.0223s for PI with a smaller overshoot in FPI than in PI, which corresponds to the transient regime. Then they are stable with a constant speed 2000rpm with 0 error for both, which corresponds to the steady-state regime.

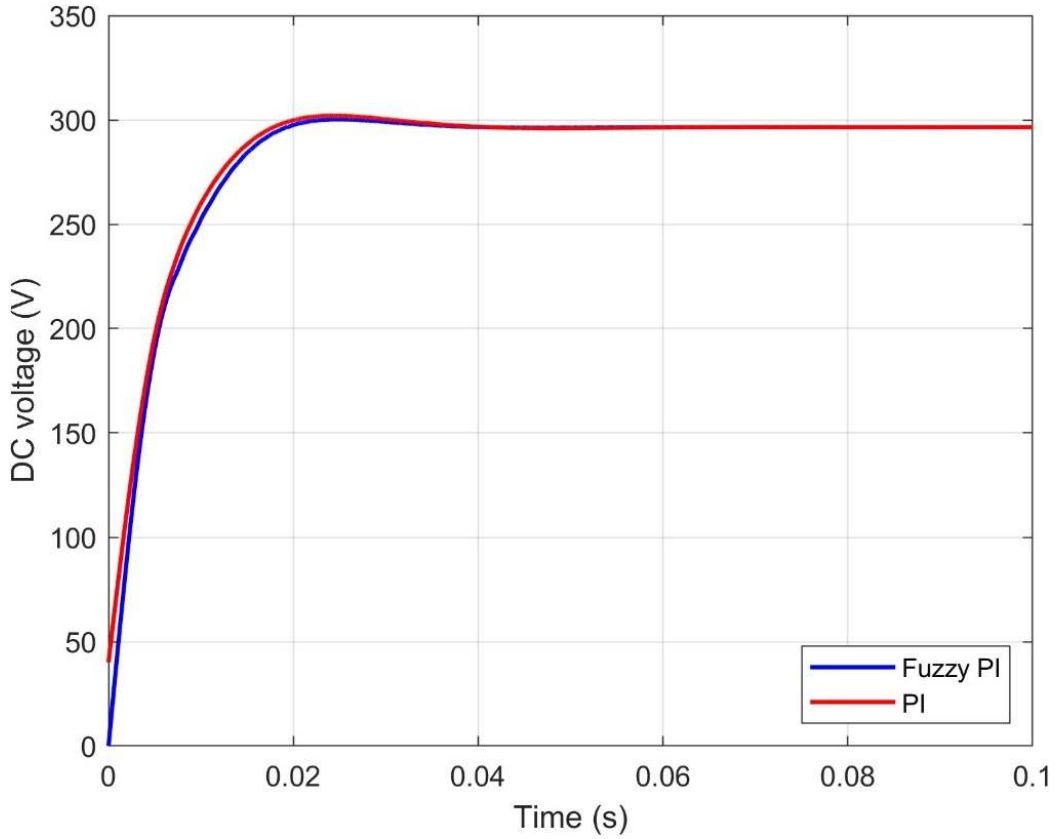

Figure III.10: DC voltage variations for case 1.

The figure 3.10 represents evaluation reference voltage for speed control for the two controlling methods, the voltage percentage gradually increases with increasing motor speed.

III.6.2 Case 2: Load Disturbance

This case presents a comparative speed response of a BLDC motor using two different control strategies a classical PI controller and a Fuzzy PI controller. In this case a load torque will applied equal to **Tr = 3 Nm** at **t = 0.05 s**

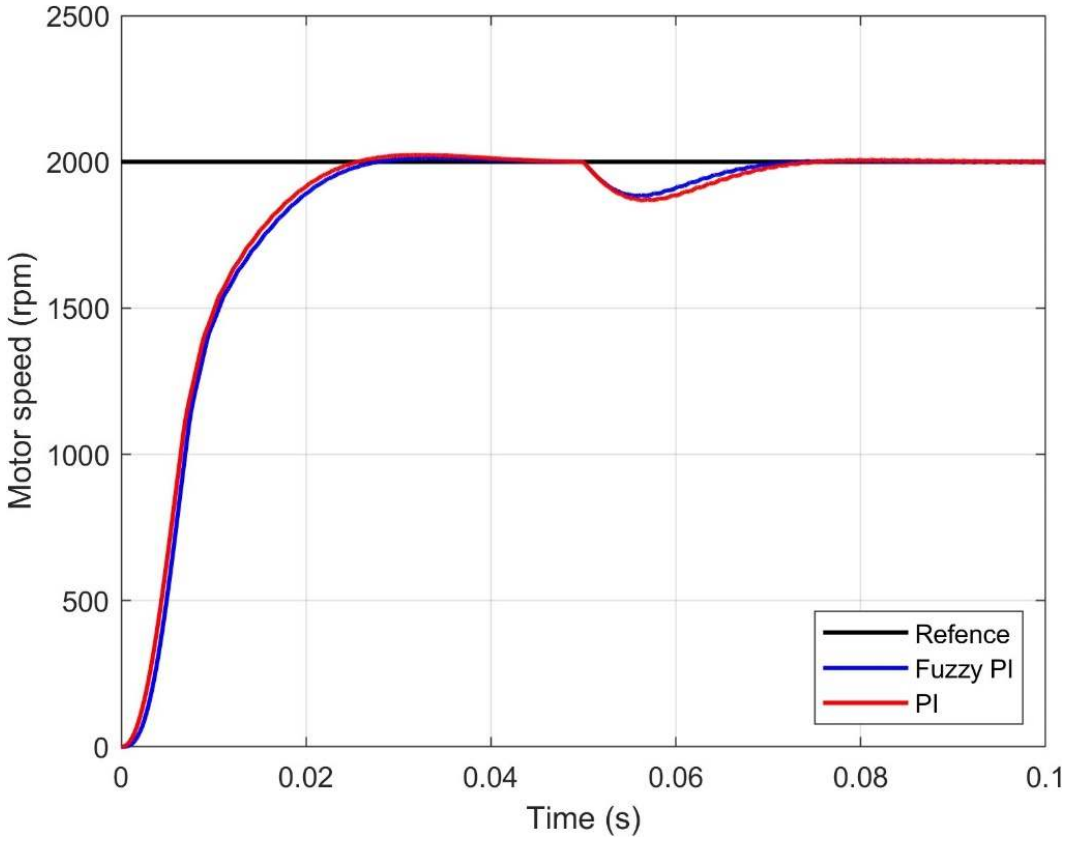

Figure III.11: Response of the BLDC speed motor case 2.

This Figure.11 represents the performance of the Fuzzy PI and PI controller in controlling the motor speed. The motor speed for both controllers increases with a good performance from **0 rpm** to the reference speed of **2000 rpm** over a period of approximately **0.025 s**, with more overshoot in PI than in FPI controller, which represents the transient regime. The speed stabilizes at the **2000 rpm** setpoint from **t = 0.025s** to **t = 0.05 s**, corresponding to the steady-state regime. At **t = 0.05 s**, a load torque is applied, causing a drop in speed. The PI controller shows a more significant drop to approximately **1870 rpm**. In the other way, the Fuzzy PI controller demonstrates better response, with the speed only falling to about **1900 rpm**. After this drop, both systems recover, with the Fuzzy PI controller returning to the **2000 rpm** reference speed more quickly and with less undershoot than the conventional PI controller.

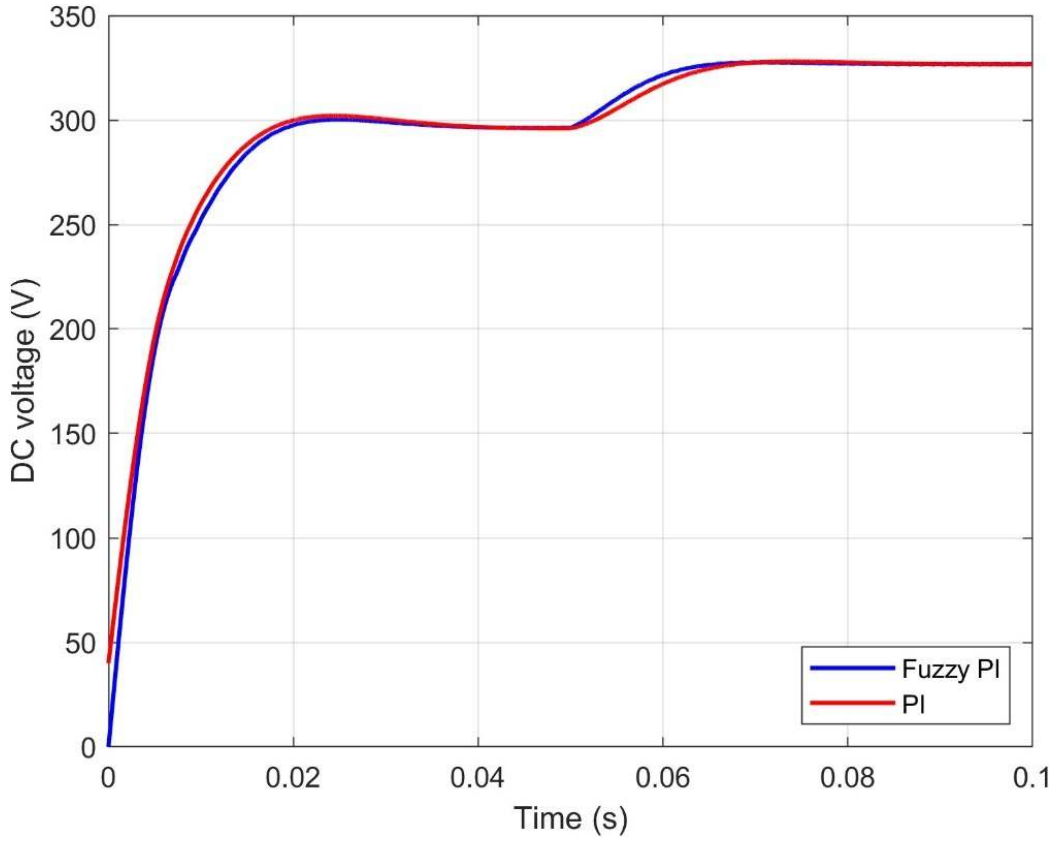

Figure III.12: DC voltage variations for case 2.

The Figure.12 represents DC voltage curve of the two controllers. The voltage increases rapidly from 0 v to **300 V** in about **0.02s** according to the accelerate the motor. Then it stabilizes at **300 V**. When the load torque is applied the voltage quickly increase and settles at a new, upper steady-state value of approximately **325 V**.

III.6.3 Case 3: Setpoint Change

This case presents a comparative speed response of a BLDC motor using two different control strategies a classical PI controller and a Fuzzy PI controller.

In this case the speed will change at **t = 0.05s** from **3000 rpm** to **1000 rpm**.

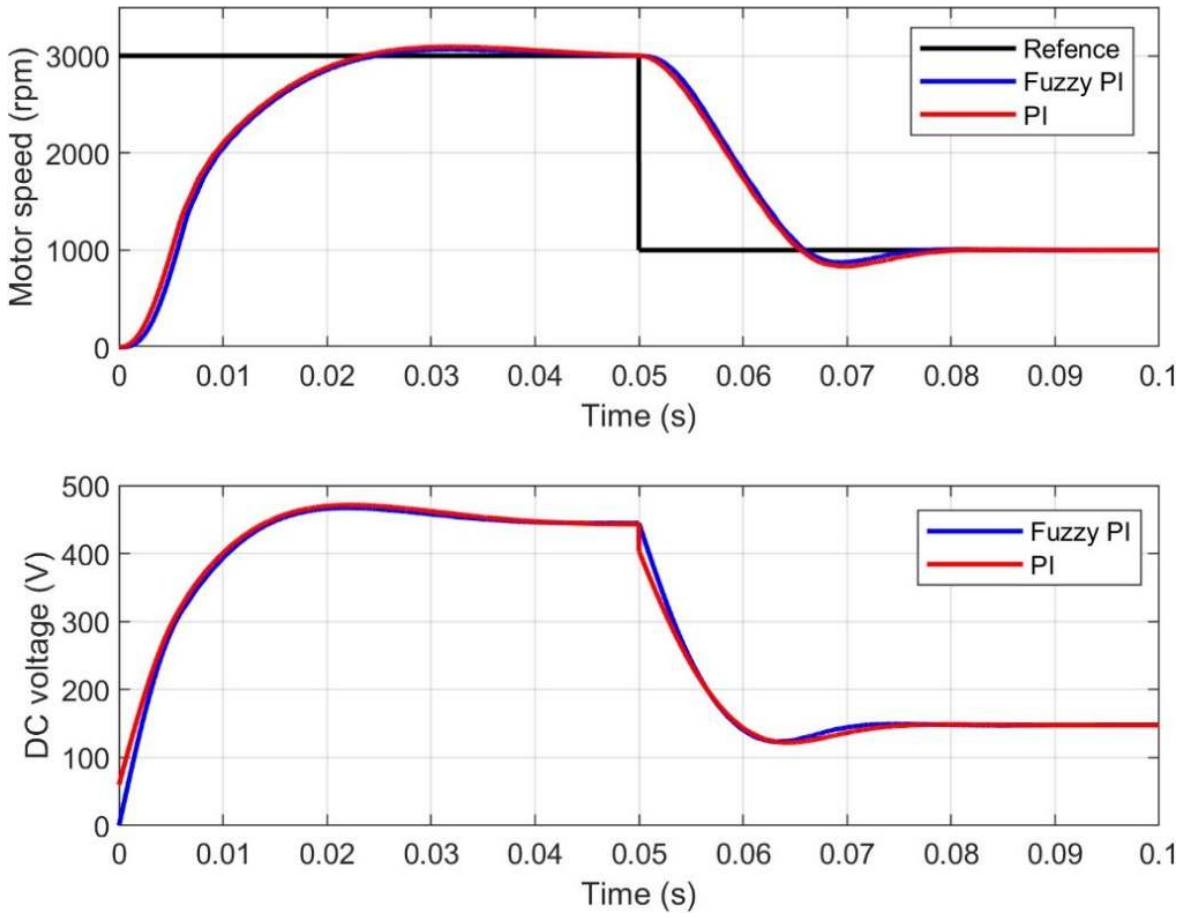

Figure III.13: Speed/voltage response of a BLDC motor case 3.

The Figure III.13 (top) represents the speed response of the controllers. In the beginning both the PI and Fuzzy PI controllers drive the motor speed from **0 rpm** up to the first reference of **3000 rpm**, reaching it in about **0.025 s** with a less overshoot in FUZZY PI than in PI. The speed is stable until t = 0.05s, suddenly the reference speed is changed to **1000 rpm**. Both controllers react by reducing the motor's speed. The Fuzzy PI controller appears to settle at the new reference speed slightly faster and with less undershoot than the conventional PI controller, which dips below **1000 rpm** before stabilizing.

The Figure III.13 (bottom) represents the DC voltage curve. The voltage rising rapidly from **0 V** to **470 V** according of the motor accelerating. It then stabilizes around **450 V**. When the motor speed is reduced at **t = 0.05s**, the voltage quickly drops and settles at a new, lower steady-state value of approximately **150 V**.

III.6.4 Case 4: Parameter Variation

In this case we change the stator resistance to **Rs = 11 Ω.** Then we compare between the two-controllers response.

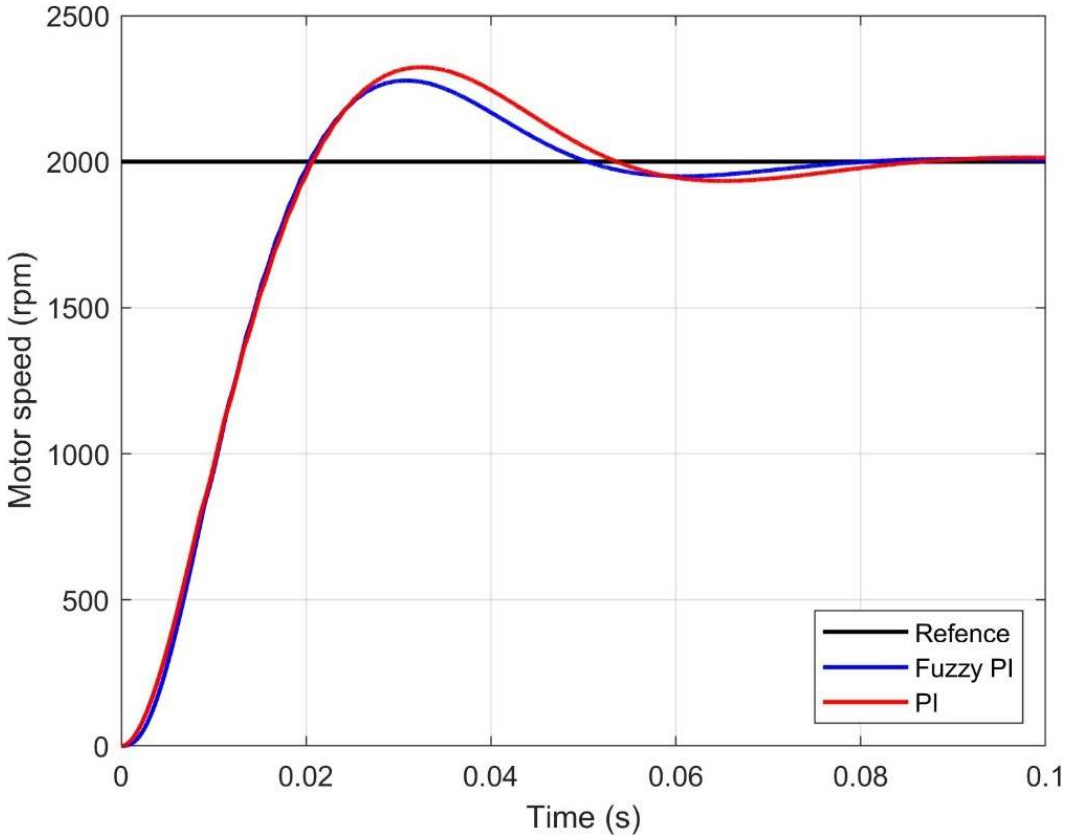

Figure III.14: Speed response of a BLDC motor case 4.

The Figure III.14 represent a compare of the performance of a Fuzzy PI controller against a conventional PI controller for motor speed controlling. Both controllers demonstrate good transient response, quickly accelerating the motor from **0 rpm** to the setpoint **2000 rpm**. The Fuzzy PI controller shows a superior performance, reaching the **2000 rpm** reference with less overshoot and undershoot and settling faster than the PI controller.

III.7 Conclusion

In this chapter, a conventional PI and Fuzzy PI controller for a BLDC motor were optimized using the Jaya algorithm. The study presented the tuning process of two different

controllers applied to the BLDC system, leveraging the parameter-free optimization capability of the Jaya algorithm. The model was implemented in MATLAB/Simulink, and the dynamic performance of the system in several cases was analyzed in detail.

The results demonstrate that the Fuzzy PI controller tuned by Jaya provides improved dynamic response compared to systems based on classical PI controller tuned by Jaya, particularly in terms of rise time, overshoot, settling time, and steady-state error. The fuzzy logic approach also offers better adaptability to variations in motor operating conditions.

It is worth noting that the Jaya algorithm, due to its simplicity and effectiveness, represents a promising alternative for the optimization of intelligent control system parameters.

Conclusion and Future Work

This thesis embarked on an investigation into advanced speed control strategies for Brushless DC (BLDC) motors, with a primary objective of enhancing control performance beyond the capabilities of traditional methods. The core of this work was centered on the design, optimization, and comparative analysis of a conventional Proportional-Integral (PI) controller and a Fuzzy Logic-based PI (Fuzzy PI) controller. To ensure optimal performance, the parameters for both controllers were fine-tuned using the Jaya optimization algorithm, a simple yet powerful metaheuristic technique.

The research was methodically structured, beginning with a foundational exploration of Fuzzy Logic Control in Chapter 1, which established the theoretical basis for developing an intelligent control system capable of handling the nonlinearities inherent in motor systems. Chapter 2 provided a comprehensive overview of the Jaya algorithm, highlighting its advantages in solving complex optimization problems without requiring algorithm-specific parameter tuning, thus simplifying the controller design process.

The culmination of this research was presented in Chapter 3, where both the Jaya-optimized conventional PI and the Jaya-optimized Fuzzy PI controllers were implemented and tested in a simulation environment for the speed control of a BLDC motor. The simulation results provided compelling evidence supporting the principal hypothesis of this thesis. The Fuzzy PI controller consistently demonstrated a superior performance profile compared to its conventional counterpart. Specifically, the Fuzzy PI controller achieved a faster transient response, exhibited minimal overshoot, and showed greater robustness in rejecting load disturbances, leading to a smoother and more stable speed regulation.

In conclusion, this work successfully demonstrates the efficacy of integrating fuzzy logic with conventional control structures and leveraging modern optimization algorithms for tuning. The findings confirm that the Jaya-optimized

Fuzzy PI controller is a highly effective and robust solution for the speed control of BLDC motors, offering significant improvements in dynamic response and stability over the optimized conventional PI controller. This approach not only enhances the performance of the motor drive system but also showcases the potential of intelligent, self-tuning control methodologies in modern industrial applications.

Future Work:

Building upon the findings of this thesis, several promising avenues for future research can be pursued:

Experimental Validation: The most critical next step is to move beyond simulation and implement the proposed Jaya-optimized Fuzzy PI controller on a physical hardware setup. This would validate the simulation results and provide insights into the controller's real-world performance and practical challenges.

Exploration of Other Intelligent Controllers: Future work could involve investigating other advanced intelligent control schemes, such as Adaptive Neuro-Fuzzy Inference Systems (ANFIS) or controllers based on Artificial Neural Networks (ANNs), to potentially achieve even greater levels of performance and adaptability.

Advanced Optimization Algorithms: While the Jaya algorithm proved effective, a comparative study with other state-of-the-art optimization techniques (e.g., Particle Swarm Optimization, Genetic Algorithm, or hybrid algorithms) could be conducted to explore further performance gains.

Sensorless Control: A significant extension of this research would be to integrate the proposed control strategy into a sensorless BLDC motor control scheme, which would reduce system cost and complexity while enhancing mechanical reliability.

References

[1] Kalnins, K., & Bluks, A. (2024). BLDC Motor Speed Control with Digital Adaptive PID-Fuzzy Controller and Reduced Harmonic Content. *Energies*, *17*(6), 1311. https://www.mdpi.com/1996-1073/17/6/1311

[2] Cook, J. (2023, April 10). *What is a brushless DC motor? Controls, applications & types*.

[3] Zadeh, L. A. (1965). Fuzzy sets. *Information and Control*, *8*(3), 338–353.

[4] Ross, T. J. (2010). *Fuzzy logic with engineering applications* (3rd ed.). John Wiley & Sons.

[5] Klir, G. J., & Yuan, B. (1995). *Fuzzy sets and fuzzy logic: Theory and applications*. Prentice Hall.

[6] Pedrycz, W., & Gomide, F. (2007). *Fuzzy systems engineering: Toward human-centric computing*. Wiley-IEEE Press.

[7] Jang, J. S. R., Sun, C. T., & Mizutani, E. (1997). *Neuro-Fuzzy and soft computing: A computational approach to learning and machine intelligence*. Prentice Hall.

[8] Mamdani, E. H., & Assilian, S. (1975). An experiment in linguistic synthesis with a fuzzy logic controller. *International Journal of Man-Machine Studies*, *7*(1), 1–13.

[9] Takagi, T., & Sugeno, M. (1985). Fuzzy identification of systems and its applications to modeling and control. *IEEE Transactions on Systems, Man, and Cybernetics*, *(1)*, 116–132.

[10] Driankov, D., Hellendoorn, H., & Reinfrank, M. (1996). *An introduction to fuzzy control*. Springer.

[11] Sivanandam, S. N., Sumathi, S., & Deepa, S. N. (2007). *Introduction to fuzzy logic using MATLAB*. Springer.

[12] Passino, K. M., & Yurkovich, S. (1998). *Fuzzy control*. Addison-Wesley.

[13] Mahmud, S. M. A. M., Motakabber, A. H. M., Alam, Z., & Nordin, A. N. (n.d.). Control BLDC motor speed using PID controller. *International Journal of Engineering and Technology*.

[14] Yedamale, P. (2003). *Brushless DC (BLDC) motor fundamentals*. Microchip Technology Inc.

[15] Madaan, P. (2013, February 11). *Brushless DC motors – Part I: Construction and operating principles*. Cypress Semiconductor.

[16] Brejl, M., Princ, M., & Sustek, P. (2006, May). *BLDC motor with Hall sensors and speed closed loop, driven by eTPU on MPC5554* (AN3006 Rev. 1). Freescale Semiconductor.

[17] Kim, S.-H. (2017). *Brushless direct current motors*. (pp. 389–416).

[18] Güzelkaya, M., Eksin, İ., & Yeşil, E. (2003). Self-tuning of PID-type fuzzy logic controller coefficients via relative rate observer. *Engineering Applications of Artificial Intelligence*, *16*(3), 227–236.

[19] Wu Zhi, Q., & Mizumoto, M. (1996). PID type fuzzy controller and parameters adaptive method. *Fuzzy Sets and Systems*, *78*(1), 23–35. https://doi.org/10.1016/0165-0114(95)00115-8

[20] Nam, Y. J., & Park, J. H. (1999). Parallel structure and tuning of a fuzzy PID controller. *Automatica*, *35*(4), 517–529.