machines

MDPI

Article

UPAFuzzySystems: A Python Library for Control and Simulation with Fuzzy Inference Systems

Martín Montes Rivera [1,*], Ernesto Olvera-Gonzalez [2] and Nivia Escalante-Garcia [2]

[1] Research and Postgraduate Studies Department, Universidad Politécnica de Aguascalientes (UPA), Aguascalientes 20342, Mexico
[2] Laboratorio de Iluminación Artificial, Tecnológico Nacional de México Campus Pabellón de Arteaga, Carretera a la Estación de Rincón Km. 1, Pabellón de Arteaga 20670, Mexico; jose.og@pabellon.tecnm.mx (E.O.-G.); nivia.eg@pabellon.tecnm.mx (N.E.-G.)
* Correspondence: martin.montes@upa.edu.mx

Abstract: The main goal of control theory is input tracking or system stabilization. Different feedback-computed controlled systems exist in this area, from deterministic to soft methods. Some examples of deterministic methods are Proportional (P), Proportional Integral (PI), Proportional Derivative (PD), Proportional Integral Derivative (PID), Linear Quadratic (LQ), Linear Quadratic Gaussian (LQG), State Feedback (SF), Adaptative Regulators, and others. Alternatively, Fuzzy Inference Systems (FISs) are soft-computing methods that allow using the human expertise in logic in IF–THEN rules. The fuzzy controllers map the experience of an expert in controlling the plant. Moreover, the literature shows that optimization algorithms allow the adaptation of FISs to control different processes as a black-box problem. Python is the most used programming language, which has seen the most significant growth in recent years. Using open-source libraries in Python offers numerous advantages in software development, including saving time and resources. In this paper, we describe our proposed UPAFuzzySystems library, developed as an FISs library for Python, which allows the design and implementation of fuzzy controllers with transfer-function and state-space simulations. Additionally, we show the use of the library for controlling the position of a DC motor with Mamdani, FLS, Takagi–Sugeno, fuzzy P, fuzzy PD, and fuzzy PD-I controllers.

Keywords: intelligent control; fuzzy logic; fuzzy inference systems; open source; Python

Citation: Montes Rivera, M.; Olvera-Gonzalez, E.; Escalante-Garcia, N. UPAFuzzySystems: A Python Library for Control and Simulation with Fuzzy Inference Systems. *Machines* **2023**, *11*, 572. https://doi.org/10.3390/machines11050572

Academic Editors: Shuai Li, Dechao Chen, Mohammed Aquil Mirza, Vasilios N. Katsikis, Dunhui Xiao and Predrag S. Stanimirovic

Received: 17 April 2023
Revised: 7 May 2023
Accepted: 10 May 2023
Published: 22 May 2023

1. Introduction

The main goal of control theory is to be tracking the input or system stabilization that produces an adjusted output associated with a process that responds to different behaviors. Achieving this implies designing a control law adapted to the process model. To obtain this model means using physics and mathematical principles with parametrizing stage after modeling to achieve similar behavior to the process [1,2].

A feedback Computer-Controlled System (CCS) has an input that samples the Process Variable (PV) with an Analog–Digital Converter (ADC). Then, the CCS uses this information to calculate an algorithm that considers the Reference Variable (RV) or desired value for PV by producing an analog output—converted from digital with a Digital–Analog Converter (DAC). All the conversions and steps in the CCS are shown in Figure 1 [3].

CCSs use algorithms ranging from deterministic to soft methods with probabilistic techniques. Some examples of deterministic methods are Proportional (P), Proportional Integral (PI), Proportional Derivative (PD), Proportional Integral Derivative (PID), Linear Quadratic (LQ), Linear Quadratic Gaussian (LQG), State Feedback (SF), and Adaptive Regulators [1,3].

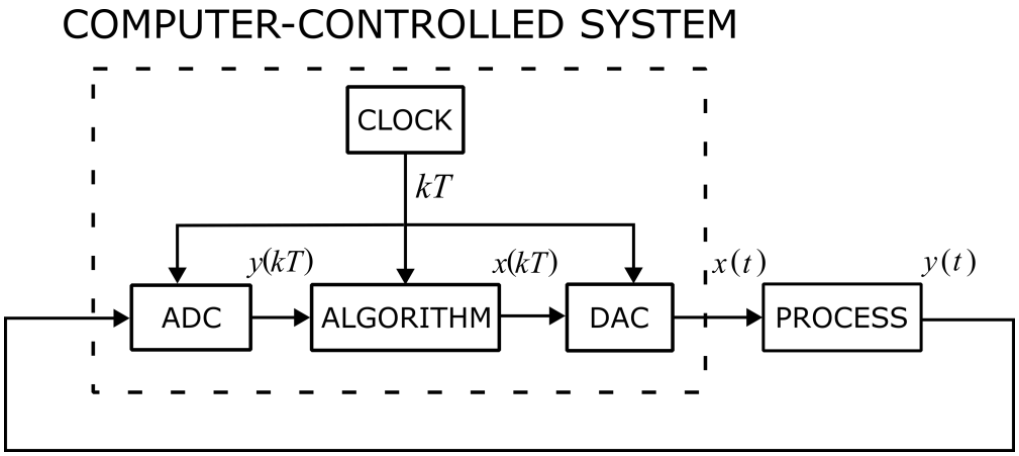

Figure 1. CCS diagram, including the ADC and DAC converters.

Alternatively, Fuzzy Inference Systems (FISs) are soft-computing methods that offer a linguistic way of dealing with complex processes and the possibility of translating human experience into logic in IF–THEN rules.

The FISs were introduced in the early 1970s by Lotfi A. Zadeh. This invention represented a breakthrough in set theory, as fuzzy logic mimics the human decision process. In 1975, Ebrahim Mamdani initiated the FIS to control a steam engine and boiler by creating linguistic synthesis control rules based on human expert operators, obtaining the first fuzzy controller [4].

The fuzzy controllers make it possible to design the control law with rules that represent the experience of the experts in controlling the plant. Moreover, the literature shows that algorithms such as Least Square Estimator (LSE), Genetic Algorithms (GAs), Particle Swarm Optimization (PSO), and Gradient Descent (GD), among others, allow the optimization of FISs for controlling different processes such as black-box problems [5–10].

FIS controllers simplify the control of complex systems without linearities and time variations, but their programming and configuration are more complex than classical approaches. However, there are several commercial alternatives for implementing and simulating FIS controllers.

The MATLAB™ fuzzy logic toolbox enables the design and implementation of FIS for control, modeling, and simulation [11]. Current research papers that implement the MATLAB™ fuzzy logic toolbox investigate the following: reducing chemical oxygen demand in low-strength wastewater [12], controlling the speed of motors for robots [13], designing the control of a three-phase grid-connected inverter using a Raspberry Pi system [14], the design, modeling, and simulation of one-degree of freedom inverted pendulum [15], and designing a single-stage photovoltaic system with energy recovery control [16].

Additionally, "IT2-FLS" is another toolbox of MATLAB™ for implementing fuzzy logic of interval type 2. It includes different construction stages, including design, description, and implementation. However, it does not contain parameters or methods specific to control systems [17].

A commercial alternative for implementing and controlling FIS is National Instruments, whose LabVIEW™ software provides a graphical programming framework. In addition, LabVIEW™ allows the implementation of real-time controllers using the same brand of diving instruments, such as USB-6001/6002/6003/6211 and others [18]. LabVIEW with FIS has shown promising results in controlling Carbon Dioxide Fertilization in a Greenhouse Environment [18], Air Valve for soprano recorders with automatic note recognition [19], Gain Scheduling of PID Controller Based on Fuzzy Systems [20], and a suspension system for a quarter-hour car [21], to name a few.

On the other hand, according to the 2022 TIOBE (The Importance of Being Earnest) index, Python is the most widely used programming language and has experienced significant growth in recent years (Figure 2) [22].

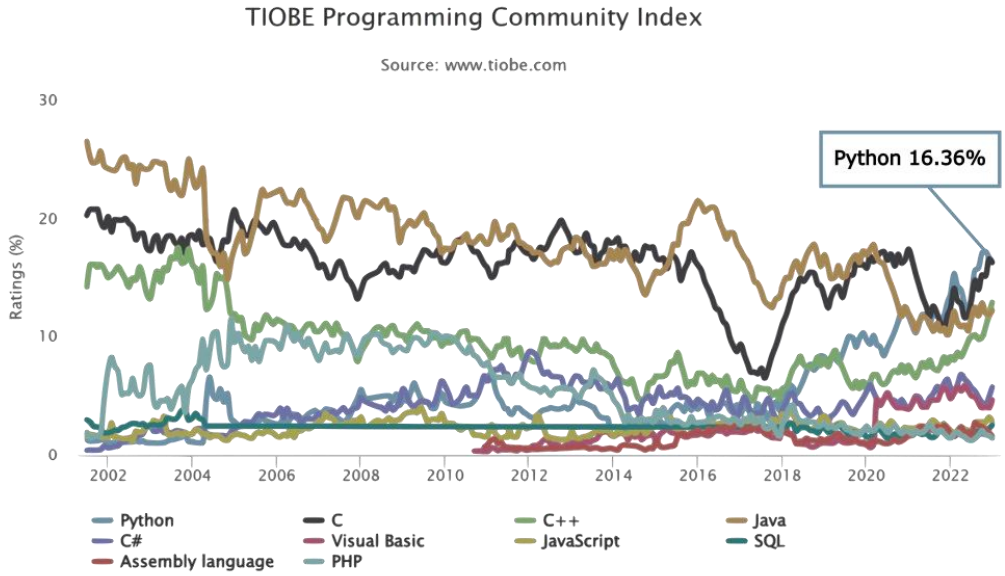

Figure 2. Python usage comparison according to the TIOBE index [22]. Source: www.tiobe.com (accessed on 16 February 2023).

Python has specific libraries for implementing simulations and control using classical approaches with model analysis and control laws. Therefore, Python should have an open-source library for implementation and simulation with FIS controllers. In addition, most of the control libraries or toolboxes in Python are open-source—a philosophy for access to programs that allows code modifications, the sharing of contributions, and use under license restrictions intended to limit responsibility.

Using open-source libraries in Python offers numerous advantages for software development. First, open-source libraries approve developers to save time using algorithms already implemented for different proposals [23]. Second, they provide high quality and reliability because they are constantly updated and maintained by a large and diverse community of developers [24]. Third, open-source libraries are cost-effective because they do not require licensing fees, aiding organizations to allocate more resources to other areas of their project [25]. Finally, using open-source libraries in Python has led to faster time-to-market and better product quality [26].

Despite that, there are other open-source alternatives for simulating FISs. They are related to developing expert systems and fuzzy logic in software applications. In addition, the control domain is not their priority, and they cannot simulate controllers with mathematical model descriptions, transfer functions, and inputs as a ramp or step response.

According to the Python Package Index (PyPI), the most relevant libraries for the implementation of FIS and fuzzy control include the average number of downloads based on Shield's IO statistics [27]:

- "Fuzzy-logic-toolbox": Library licensed in 2020 based on the behavior in MATLAB™ without simulation of fuzzy controllers with an average of 47 downloads per month [28].
- "Scikit-fuzzy": A library licensed in 2012 to popularize fuzzy logic in Python, agreeing to simulate and describe FIS using rounding arithmetic with IEEE (Institute of Electrical and Electronics Engineers) standards. However, simulations with transfer functions or mathematical models lack control structures such as fuzzy PID controllers. Shield's IO statistics state that it has an average of 26,000 times per month.

- "fuzzylab": Licensed in 2007, this is a library based on Octave Fuzzy Logic Toolkit 0.4.6. This library allows the simulation of FISs without including the implementation of controls with transfer functions, mathematical models, or other control structures in its methods. However, its developers used it to control an autonomous robot's navigation system [29,30], averaging 53 downloads per month.
- "fuzzython": Released in 2013, it allows the construction of FIS, including the Mamdani, Sugeno, and Tsukamoto models, but it misses tools for working with fuzzy control or the simulation of systems with transfer function or state-space model descriptions [31], with an average of 12 downloads per month.
- "Type2Fuzzy": Licensed in 2007, this library allows work with type 2 FIS, in general descriptions for software applications, but it does not include methods for working with transfer functions, state-space models, and fuzzy control [32] with an average of 53 downloads per month.
- "Fuzzy-machines": This is a 2018 licensed library for working with FIS but does not include methods for working with fuzzy controllers, transfer functions, or state-space descriptions, with an average of 18 times per month.
- "pyfuzzylite": A 2007 licensed Library for developing FIS and controllers 2007 over a graphic interface. It allows working in Mamdani, Takagi–Sugeno, Larsen, Tsukamoto, Inverse Tsukamoto, and Hybrids. However, it does not include fuzzy PID controllers or methods for simulation with transfer functions and state-space representations [33] from an average of 302 downloads per month.
- "Simpful": It depends on "numpy" and "scipy" libraries. It has properties of polygonal and functional models. It allows the definition of fuzzy rules as text strings in natural language, the description of complex fuzzy two rules built with logical operators, and Mamdani and Takagi–Sugeno interference methods. However, it does not consider parameters for automatic control [33]. Shield's IO statistics state that it has an average of 113,000 times per month.
- "pyFume": It collects classes and methods for the antecedent set and associated parameters of a Takagi–Sugeno (TS) fuzzy system from data using the Simpful library. The antecedent set and related parameters of a Takagi–Sugeno fuzzy model are extracted from data and then building an executable fuzzy model using the Simpful library. It only applies fuzzy logic and does not consider automatic control parameters [34]. Shield's IO statistics state that it has an average of 120,000 times per month.

In this work, we describe our proposal UPAFuzzySystems library developed in the Universidad Politécnica de Aguascalientes (UPA) as an FISs library for Python, which supports the design and implementation of the fuzzy controller with transfer function and state-state representations already published in open-source license in [34]. In addition, our proposal uses the "control" library for simulation. According to Shield's statistics IO, the "control" library is the most used library for simulating controllers in Python, with 57,000 downloads per month. Furthermore, our proposal includes P, PD, PI, and PID fuzzy controller structures. This proposal is a novel idea since no other Python library allows the simulation and test of fuzzy controllers with transfer functions and state-space models. Furthermore, there are no other alternatives in Python for designing fuzzy controllers with PID structures. Moreover, we compare all these libraries in terms of their capabilities in designing FISs and fuzzy controllers and simulating them (Table 1).

Table 1. Comparison of Python libraries for FISs and their capabilities, including UPAFuzzySystems.

Library	Design of FISs	Design of FISs Controllers	PID FISs Controllers	Simulation of FISs Controllers with Transfer Functions and State-Space Models
Fuzzy-logic-toolbox	Yes	No	No	No
Scikit-fuzzy	Yes	Yes (Only Mamdani controller)	No	No
fuzzylab	Yes	No	No	No
fuzzython	Yes	No	No	No
Type2Fuzzy	Yes (Type 2)	No	No	No
Fuzzy-machines	Yes	No	No	No
pyfuzzylite	Yes	Yes	No	No
Simpful	Yes	No	No	No
pyFume	Yes	No	No	No
UPAFuzzySystems	Yes	Yes	Yes	Yes

2. Materials and Methods

FISs belong to the soft-computing methods because, unlike hard-computing methods, they consent to work with tolerances and imprecisions and make decisions under uncertainty. Moreover, our natural language deals with imprecision in inference because we allow fuzzy boundaries instead of well-defined ones [35,36].

FISs authorize using membership degrees in sets to express logic with fuzzy boundaries. The membership value μ defines the degree of membership in the ranges $[0, 1]$.

A membership function $\mu(x)$ defines μ for each x point of the universe X in a fuzzy set A as in Equation (1).

$$A = \{x, \mu(x) | x \in X\} \quad (1)$$

For example, deciding whether a person is young or old, with an age's universe in the range $[0, 100]$ years, one could use well-defined bounds (Figure 3a) to express a falsely narrow categorization or use FISs with trapezoidal $\mu(x)$ to correctly represent the imprecisions and uncertainties of the problem, as shown in Figure 3b.

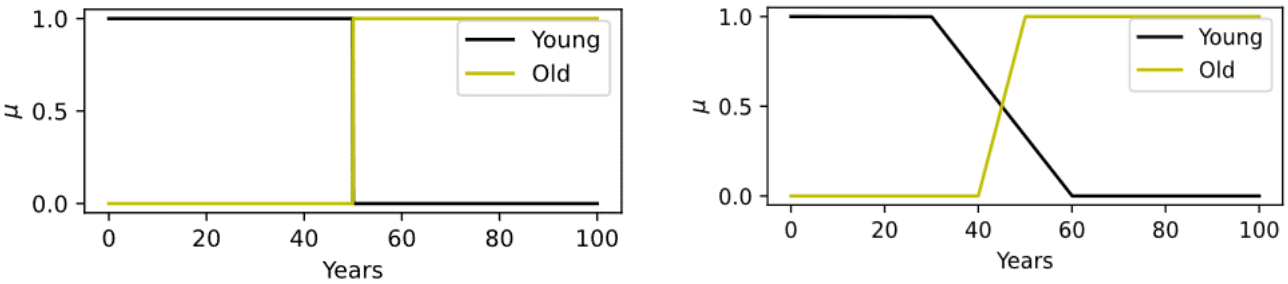

Figure 3. Person Age. (**a**) Crisp logic with clearly defined boundaries. (**b**) Fuzzy logic with unclear boundaries.

The $\mu(x)$ forms have a significant impact on the FISs behavior. The most common membership functions include the triangle in Equation (2) as a function of the vertices a, b, c; the trapezoid in Equation (3) depending on vertices a, b, c, d; the Gaussian in Equation (4) as a function of the parameters c and σ; the generalized bell in Equation (5) subject to parameters a, b, c, and others. Moreover, one can assign μ directly based on empirical

values or in the statistical metrics of a dataset by specifying the raw membership values and the corresponding element of the universe as in Equation (6) [37].

$$\boldsymbol{triangle}(x; a, b, c) = \begin{cases} 0, & x \leq a \\ \frac{x-a}{b-a}, & a \leq x \leq b \\ \frac{c-x}{c-b}, & b \leq x \leq c \\ 0, & c \leq x \end{cases} \tag{2}$$

$$\boldsymbol{trapezoid}(x; a, b, c, d) = \begin{cases} 0, & x \leq a \\ \frac{x-a}{b-a}, & a \leq x \leq b \\ 1, & b \leq x \leq c \\ \frac{d-x}{d-c}, & c \leq x \leq d \\ 0, & d \leq x \end{cases} \tag{3}$$

$$\boldsymbol{gaussian}(x; c, \sigma) = e^{-\frac{1}{2}\left(\frac{x-c}{\sigma}\right)^2} \tag{4}$$

$$\boldsymbol{bell}(x; a, b, c) = \frac{1}{1 + \left|\frac{x-c}{a}\right|^{2b}} \tag{5}$$

$$\boldsymbol{raw}(x, \mu) = \{(x_1, \mu_1), (x_2, \mu_2), \ldots, (x_n, \mu_n)\}, \quad i = 1, \ldots, n \tag{6}$$

The FISs derive conclusions using logical IF–THEN rules, with premises using fuzzy sets and consequences that can use fuzzy sets for Mamdani, FLSmith; functions for Takagi–Sugeno; and raw numbers in fuzzy linear and PID controllers [5,36]. The FISs structure includes the following phases: preprocessing, fuzzification, rule base, inference engine, defuzzification, and post-processing (Figure 4).

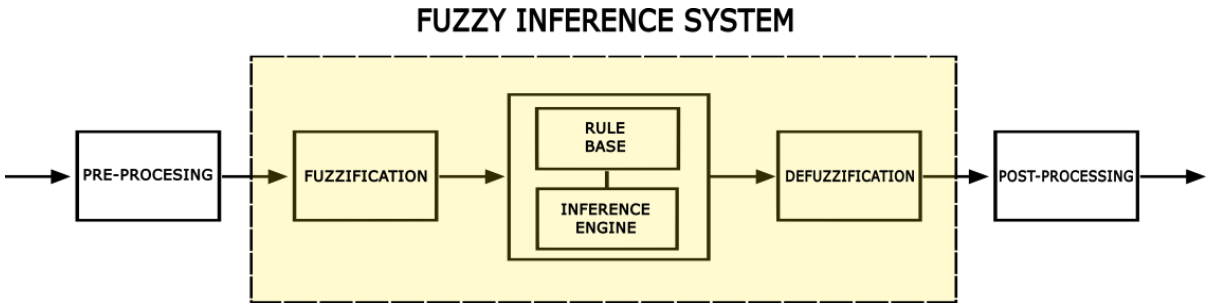

Figure 4. Structure of a Fuzzy Inference System.

IF–THEN rules imply working with connectives to define interactions midst premises represented with fuzzy sets. Moreover, these connectives agree with the definition of fuzzy composite sets in n dimensions that map interactions between premises. Connectives include the intersection (AND) and union (OR) operations, which are based on classical set theory but adapted to fuzzy logic. Equations (7) and (8) define the AND ($\wedge$) and OR ($\vee$) connectives for nonlinear FIS, and Equations (9) and (10) define the AND ($\wedge$) and OR ($\vee$) connectives for linear systems respectively, in that order [35,36].

$$\mathcal{A} \wedge \mathcal{B} \equiv \boldsymbol{min}(\mu_{\mathcal{A}}(x), \mu_{\mathcal{B}}(x)) \tag{7}$$

$$\mathcal{A} \vee \mathcal{B} \equiv \boldsymbol{max}(\mu_{\mathcal{A}}(x), \mu_{\mathcal{B}}(x)) \tag{8}$$

$$\mathcal{A} \wedge \mathcal{B} \equiv \mu_{\mathcal{A}}(x) * \mu_{\mathcal{B}}(x) \tag{9}$$

$$\mathcal{A} \vee \mathcal{B} \equiv \mu_{\mathcal{A}}(x) + \mu_{\mathcal{B}}(x) - \mu_{\mathcal{A}}(x) * \mu_{\mathcal{B}}(x) \tag{10}$$

After defining premises with their connectives, Implications ($\rightarrow$), or Equivalences ($\leftrightarrow$) enable the derivation of conclusions or consequences with rules in the form IF–THEN of Equation (11). Implications more commonly used in controllers include the Mamdani implication in Equation (12), which gathers the consequence based on the fuzzy input universe $\boldsymbol{X}$ to the fuzzy output universe $\boldsymbol{Y}$. The Mamdani and FLS controllers (FL Smidth) use this implication [5].

$$\textbf{If } f(e_1 \textbf{ is } \mathcal{A}_1, e_2 \textbf{ is } \mathcal{A}_2, \ldots, e_k \textbf{ is } \mathcal{A}_k) \textbf{ then } f(\mathrm{y}_1 \textbf{ is } \mathcal{B}_1, \mathrm{y}_2 \textbf{ is } \mathcal{B}_2, \ldots, \mathrm{y}_k \textbf{ is } \mathcal{B}_k) \tag{11}$$

$$\{\langle \langle x, y \rangle, \mu_{\mathcal{A}' \Rightarrow' \mathcal{B}}(x, y) \rangle \mid x \in X, y \in Y, \mu_{\mathcal{A}' \Rightarrow' \mathcal{B}}(x, y) = min(\mu_{\mathcal{A}}(x), \mu_{\mathcal{B}}(y))\} \tag{12}$$

Alternatively, Takagi and Sugeno contain fuzzy rules that generate a set of linear functions depending on the premises. Takagi–Sugeno controllers generate output functions depending on the error and change in error as in Equation (13). Linear and fuzzy PID controllers and Takagi–Sugeno controllers use this approach [5].

$$\textbf{If } f(e_1 \textbf{ is } \mathcal{A}_1, e_2 \textbf{ is } \mathcal{A}_2, \ldots, e_k \textbf{ is } \mathcal{A}_k) \textbf{ then } y = g(e_1, e_2, \ldots, e_k) \tag{13}$$

Finally, defuzzification allows an appropriate scalar output for the controlled process or specific application of the FIS. There are several defuzzification methods, including Center of Gravity (COG) for continuous fuzzy sets in Equation (14), Center of Gravity for Singletons (COGS) (Equation (15) for singletons and Equation (16) for discrete systems), Bisector of Area (BOA) in Equation (17), Mean of Maxima (MOM) in Equation (18), Leftmost Maxima or Smallest of Maxima (LM) in Equation (19), and Rightmost Maxima or Largest of Maxima (RM) in Equation (20) [5,36].

$$COG = \frac{\int_X \mu_A(x) x dx}{\int_X \mu_A(x) dx} \tag{14}$$

$$COGS = \frac{\sum_k \alpha_k^* S_k}{\sum_k \alpha_k^*} \tag{15}$$

$$COGS = \frac{\sum_i \mu_A(x_i) x_i}{\sum_i \mu_A(x_i)} \tag{16}$$

$$BOA = argmin_{x_j} \left(\left| \sum_{i=1}^{j} \mu_c(x_i) - \sum_{i=j+1}^{i_{max}} \mu_c(x_i) \right| \right), \; 1 < j < i_{max} \tag{17}$$

$$MOM = \frac{\sum_{i \in \mathcal{I}} x_i}{|\mathcal{I}|}, \mathcal{I} = \{i \mid \mu_c(x_i) = \mu_{max}\} \tag{18}$$

$$LM = x_{min(\mathcal{I})} \tag{19}$$

$$RM = x_{max(\mathcal{I})} \tag{20}$$

FIS controllers apply different combinations of universes, membership functions, inference rules, connectives, implications, and defuzzification. However, all the configurations have common input premises that depend on the error and output consequences to change the process state. Table 2 describes some of these configurations.

Table 2. Configurations for FIS controllers.

FIS Controller	Input Universes	Connectives	IF-THEN Rules	Defuzzification
Mamdani	Not defined	Equations (7) and (8)	Equation (11)	Equations (14) and (16)
FLS	[−1, 1]	Equations (9) and (10)	Equation (11)	Equation (17)
Linear	[−100, 100]	Equations (9) and (10)	Equation (13)	Equation (15)
Takagi–Sugeno	Not defined	Equations (9) and (10)	Equation (13)	Equation (15)
Linear P	[−100, 100]	Equations (9) and (10)	Equation (13)	Equation (15)
Linear PD	[−100, 100]	Equations (9) and (10)	Equation (13)	Equation (15)
Linear PID	[−100, 100]	Equations (9) and (10)	Equation (13)	Equation (15)
No-linear P	Not defined	Equations (7) and (8)	Equation (11)	Equations (14) and (16)
No-linear PD	Not defined	Equations (7) and (8)	Equation (11)	Equations (14) and (16)
No-linear PID	Not defined	Equations (7) and (8)	Equation (11)	Equations (14) and (16)

The fuzzy P, PD, and PID controllers start with a linear structure, as in Table 2. The goal of using the linear approach is to first design a twin of a P, PD, or PID controller for the process with classic controllers. Then, the structure is changed to a no-linear form. This change allows having a no-linear FIS controller with the benefits of derivatives and integrals without having too complex rules [36].

The P controller has two gains, GE and GU, as shown in Figure 5. First, GE sets the control deviation of the plant to be in the ranges [−100, 100] required for this controller (Table 2). Then, GU sets the gain so that it is equal to K_p of the previously designed classical P controller. Equations (21) and (22) determine the GE and GU, respectively [36].

$$GE = \frac{100}{max(|e|)} \tag{21}$$

$$GU = \frac{K_p}{GE} \tag{22}$$

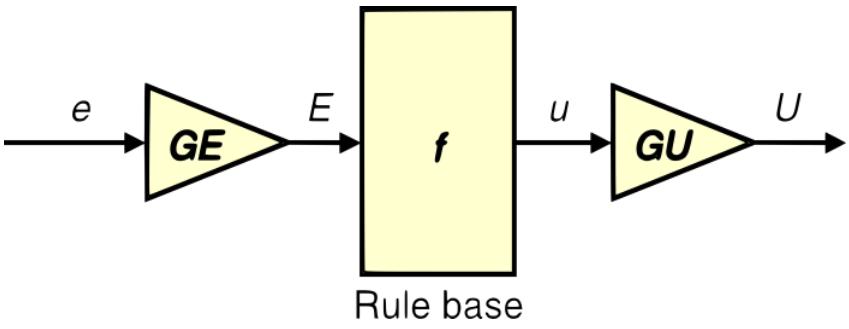

Figure 5. Structure of P fuzzy controller.

The controller FIS PD in Figure 6 has three gains GE, GU, and GCE. FIS PD, of the FIS P, uses GE to adjust the error to be in the ranges [−100, 100] and then uses Equation (22) to determine GU. Then, GCE is determined from the lead time T_d using Equation (23).

$$GCE = GE \cdot T_d \tag{23}$$

The controller FIS PID in Figure 7 uses the same structure as the controller PD with GE and GU for scaling the input with Equation (21) and obtaining the equivalent K_p with Equation (22). Similarly, it uses GCE to map the derivative gain from the derivative time T_d using Equation (23). However, FIS PID includes an integrative effect controlled by the gain GIE and maps the effect of the integrative time T_i, thus Equation (24).

$$GIE = \frac{GE}{T_i} \tag{24}$$

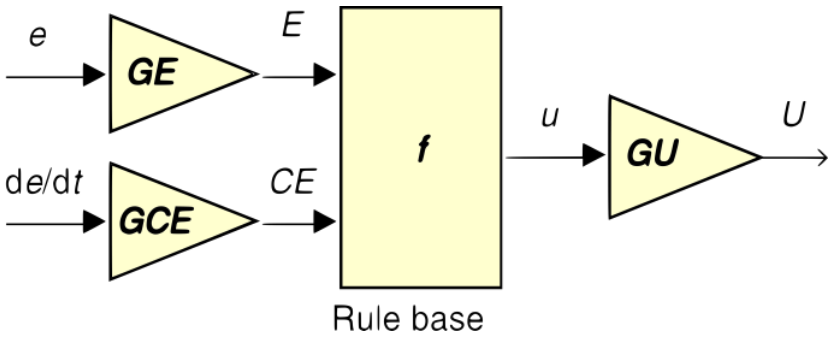

Figure 6. Structure of PD controller.

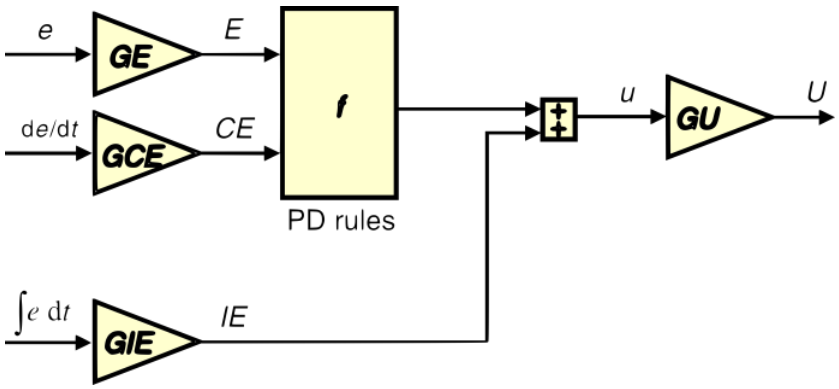

Figure 7. Structure of PID controller.

Having defined the linear structures FIS P, PD, and PID, let us explain the inference system with the IF–THEN rules. The conditions and consequences for these rules depend on the type of FIS controller. Linear P, Linear PD, and Linear PID, for example, use rules similar to those of the linear controller to create a clone of a classical controller in the design stage. Then, they switch to no-linear rules such as those used in Mamdani, FLS, or Takagi–Sugeno controllers [36]. Table 3 shows the type of rules used in different FIS controllers.

Our proposed library UPAFuzzySystems for Python is already published as an open-source license. It implements fuzzy universes with fuzzy sets, FIS systems, and FIS controllers with a simple definition of all required parameters by three different Python classes fuzzy_universe, inference_system, and fuzzy_controller. All the codes of UPAFuzzySystems are in [34].

The class fuzzy_universe is the first-level definition, i.e., all other classes require it for describing premises or consequences with fuzzy universes. A continuous or discrete fuzzy universe contains fuzzy sets defined in the universe with membership values for defining certain situations. The class fuzzy_universe also has several methods for adding and removing fuzzy sets with all membership functions from Equations (2)–(6), all of which can be found in [34].

The class fuzzy_inference is the second-level class. It allows the description of the IF–THEN rules using the premises and consequences defined with the class fuzzy_universe. Moreover, the class fuzzy_controller uses it to specify its IF–THEN rules. All methods related to fuzzy_inference class are in [33]. Finally, the class controller defines the fuzzy controllers with specific structures and simulates their behavior with transfer functions and state-space equations. All methods for working with the class fuzzy_controllers are in [34].

Table 3. Rules for the different FIS controllers.

FIS Controller	Example of Rules
Mamdani	One input: If **error** is **Neg** then **control** is **Neg** If **error** is **Zero** then **control** is **Zero** If **error** is **Pos** then **control** is **Pos** Two input: If **error** is **Neg** and **change error** is **Neg** then **control** is **Neg** If **error** is **Neg** and **change error** is **Zero** then **control** is **Neg** If **error** is **Zero** and **change error** is **Neg** then **control** is **Zero** If **error** is **Neg** and **change error** is **Pos** then **control** is **Zero** If **error** is **Zero** and **change error** is **Zero** then **control** is **Zero** If **error** is **Zero** and **change error** is **Pos** then **control** is **Zero** If **error** is **Pos** and **change error** is **Neg** then **control** is **Zero** if **error** is **Pos** and **change error** is **Zero** then **control** is **Pos** If **error** is **Pos** and **change error** is **Pos** then **control** is **Pos**
FLS	One input: If **error** is **Neg** then **control** is **Neg** If **error** is **Zero** then **control** is **Zero** If **error** is **Pos** then **control** is **Pos** Two input: If **error** is **Neg** and **change error** is **Neg** then **control** is **Neg** If **error** is **Neg** and **change error** is **Zero** then **control** is **Neg** If **error** is **Zero** and **change error** is **Neg** then **control** is **Zero** If **error** is **Neg** and **change error** is **Pos** then **control** is **Zero** If **error** is **Zero** and **change error** is **Zero** then **control** is **Zero** If **error** is **Zero** and **change error** is **Pos** then **control** is **Zero** If **error** is **Pos** and **change error** is **Neg** then **control** is **Zero** if **error** is **Pos** and **change error** is **Zero** then **control** is **Pos** If **error** is **Pos** and **change error** is **Pos** then **control** is **Pos**
Linear	One input: If **error** is **Neg** then **control** is $-$**100** If **error** is **Zero** then **control** is **0** If **error** is **Pos** then **control** is **100** Two inputs: If **error** is **Neg** and **change error** is **Neg** then **control** is $-$**200** If **error** is **Neg** and **change error** is **Pos** then **control** is **0** If **error** is **Pos** and **change error** is **Neg** then **control** is **0** If **error** is **Pos** and **change error** is **Pos** then **control** is **200**
Takagi–Sugeno	One input: If error is Neg then control is $\boldsymbol{a}\cdot\boldsymbol{error}+\boldsymbol{b}$ If error is Zero then control is **0** If error is Pos then control is $\mathbf{c}\cdot\boldsymbol{e}\mathbf{rror}+\mathbf{d}$ Two inputs: If error is Neg and change error is Neg then control is $\boldsymbol{a}\cdot\boldsymbol{error}+\boldsymbol{b}\cdot\boldsymbol{cherror}+\boldsymbol{c}$ If error is Neg and change error is Pos then control is **0** If error is Pos and change error is Neg then control is **0** If error is Pos and change error is Pos then control is $\boldsymbol{d}\cdot\boldsymbol{error}+\boldsymbol{e}\cdot\boldsymbol{cherror}+\boldsymbol{c}$
Linear P	If **error** is **Neg** then **control** is $-$**100** If **error** is **Zero** then **control** is **0** If **error** is **Pos** then **control** is **100**
Linear PD	If **error** is **Neg** and **change error** is **Neg** then **control** is $-$**200** If **error** is **Neg** and **change error** is **Pos** then **control** is **0** If **error** is **Pos** and **change error** is **Neg** then **control** is **0** If **error** is **Pos** and **change error** is **Pos** then **control** is **200**
Linear PID	If **error** is **Neg** and **change error** is **Neg** then **control** is $-$**200** If **error** is **Neg** and **change error** is **Pos** then **control** is **0** If **error** is **Pos** and **change error** is **Neg** then **control** is **0** If **error** is **Pos** and **change error** is **Pos** then **control** is **200**

3. Results and Discussion

The results in this section cover the steps and results in implementing fuzzy universes, fuzzy inference systems, and fuzzy controllers with UPAFuzzySystems.

3.1. Important Libraries

When working with figures in Python, the "matplotlib" library is used and imported in line 1 of Code 1. Then, in line 2, the proposed library "UPAFuzzySystems" is inserted to implement fuzzy universes, inference systems, and controllers. Then, we import the "numpy" library in line 3 to define vectors, matrices, and operations with numbers. Finally, the "control" library in line 4 allows defining transfer functions and state-space equations for simulation. The Python code to import the libraries is in Code 1.

Code 1. Importing main libraries in Python.

```
1 import matplotlib.pyplot as plt
2 import UPAFuzzySystems as UPAfs
3 import numpy as np
4 import control as cn
```

3.2. Fuzzy Universes with Fuzzy_Universe Class

After importing the main libraries, an example of fuzzy universes with a description of Collision Distances from 0 to 60 m in near, middle, and long-range allows explaining the use of the fuzzy_universe class in the "UPAFuzzySystems" library.

Code 2 in line 1 defines the universe from 0 to 60 with 100 samples using a "numpy" library. Then, line 2 corresponds to an instance of the class fuzzy_universe, passing the name of the universe and the instruction to perform a continuous universe.

Next, we add fuzzy sets for all collision regions in lines 3–5 by passing the name of the fuzzy set, the type of fitness function, and its vertices. This example uses trapezoidal and triangular membership functions according to Equations (2) and (3), respectively.

Finally, in line 6, we access the plot of the fuzzy_universe, and in lines 7–10, the figure was modified to change the axis names. All the Python code for the implementation is in Code 2.

Code 2. Python code for the description of a fuzzy universe.

```
1  distances = np.linspace(0,60,100)
2  DistanceColision = UPAfs.fuzzy_universe('Collision Distance',distances,'continuous')
3  DistanceColision.add_fuzzyset('close-range','trapmf',[0,0,5,15])
4  DistanceColision.add_fuzzyset('mid-range','trimf',[10,20,30])
5  DistanceColision.add_fuzzyset('long-range','trapmf',[25,40,60,60])
6  DistanceColision.view_fuzzy()
7  ax = plt.gca()
8  ax.set_xlabel('Collision Distance (m)')
9  ax.set_ylabel("$\mu$")
10 plt.show()
```

Figure 8 shows the results obtained with the view_fuzzy method executed in line 6 of Code 2 and the updated information in the axis.

3.3. Fuzzy Inference System with Inference_System Class

The IF–THEN rules of an inference system require premises and consequences. Therefore, we define a consequence of speed for the premise defined in Section 3.2. We follow the approach described in Code 2 and create the consequence in Figure 9.

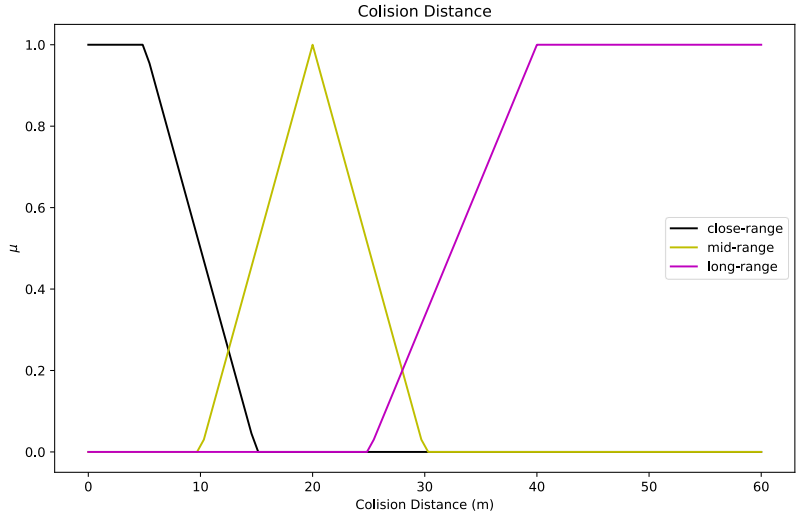

Figure 8. View obtained of a fuzzy universe of collision distance with UPAFuzzySystems library.

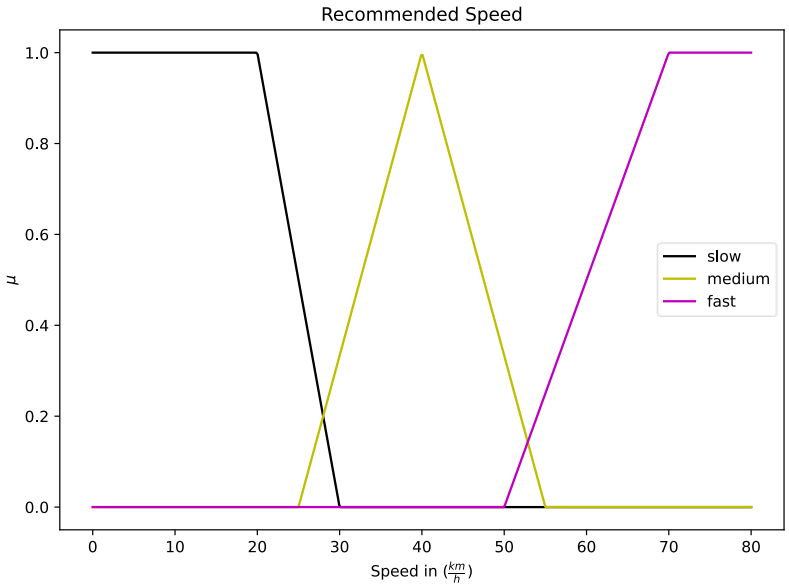

Figure 9. Fuzzy universe of recommended speeds with UPAFuzzySystems library.

After that, we set the rules for the FIS: in this case, a slow speed for a near collision distance, a medium speed for a medium-range collision distance, and a fast speed for a far-range collision distance. In other words, the rules:

1. IF **Collision Distance** is **short-range** → **Speed** is **slow**
2. IF **Collision Distance** is **mid-range** → **Speed** is **medium**
3. IF **Collision Distance** is **long-range** → **Speed** is **fast**

Code 3 defines the inference system. First, we create an instance of the class inference_system and pass the name to assign in line 1. Then, lines 2 and 3 define the premise and the consequence, respectively. Next, lines 4–6 describe the rules for the FIS, passing a list of premises with a list per premise with its name and the corresponding fuzzy set name, a list of connectives, and a list of consequences with a list per consequence with its name and the corresponding fuzzy set name.

Hence, in line 7, the type of rules is configured. In this case, the Mamdani system with its configuration is shown in Table 2. Then, in line 8, the FIS is created. To display the system's surface in line 9, call the method surface_fuzzy_system and pass a list of "numpy" arrays, one per input premise, as before. Finally, lines 10–13 handle the axis

labels. The Python code to define the inference system is in Code 3. The surface obtained is in Figure 10.

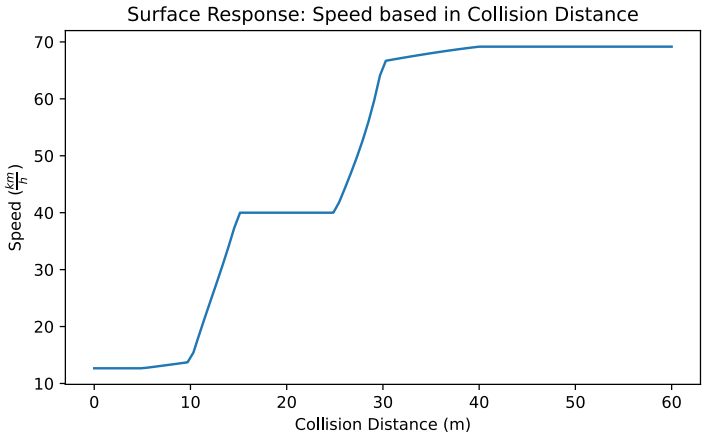

Figure 10. Surface obtained from inference system with UPAFuzzySystems library.

Code 3. Python code for defining inference system and its rules.

```
1  Speed_Collision = UPAfs.inference_system("Speed based in Collision Distance")
2  Speed_Collision.add_premise(DistanceColision)
3  Speed_Collision.add_consequence(Speed)
4  Speed_Collision.add_rule([['Collision Distance','close-range']],[],[['Recommended Speed','slow']])
5  Speed_Collision.add_rule([['Collision Distance','mid-range']],[],[['Recommended Speed','medium']])
6  Speed_Collision.add_rule([['Collision Distance','long-range']],[],[['Recommended Speed','fast']])
7  Speed_Collision.configure('Mamdani')
8  Speed_Collision.build()
9  Speed_Collision.surface_fuzzy_system([distances])
10 ax = plt.gca()
11 ax.set_xlabel(r"Collision Distance (m)")
12 ax.set_ylabel(r"Speed ($\frac{km}{h}$)")
13 plt.show()
```

3.4. Fuzzy Controller with Fuzzy_Controller Class

Let us define the Direct Current (DC) motor position as the process variable to control. The parameters and the transfer function are in Table 4. Ref. [38] details the DC motor used and the process for obtaining its model.

Table 4. Parameters and transfer function for controlling position in a DC motor.

Parameter	Description	Value
J	Inertial Coefficient	3.2284×10^{-6} kg·m^2
b	Viscous Friction Coefficient	3.5077×10^{-6} $\frac{Ns}{m}$
K	Electromotive Force	0.0274 $\frac{RPM}{V}$
R	Armor Resistance	4 Ω
L	Armor Inductance	2.75×10^{-6} H
te	Simulation Time	1.0 s
ns	Total Number of Samples	1500
tf	Transfer Function	$\frac{K}{s((Js+b)(Ls+R)+K^2)}$

Code 4 shows the code for defining the transfer function, its parameters, a test input, and a test signal with disturbances using the control library. The test input starts at zero and gradually changes in 5 ms to the equivalent radian for 45° (0.78539816 rad) 0.25 s later. Lines 1–7 give the parameters, line 8 describes the universe, line 9 generates the input, line 10 defines the starting seed for randomness, line 11 represents the uniform random

perturbation around 10% of the reference input, line 12 defines the s term for the frequency space, and line 13 stipulates the transfer function.

Code 4. Code for defining transfer function and its parameters in a DC motor.

```
1  J = 3.2284E-6
2  b = 3.5077E-6
3  K = 0.0274
4  R = 4
5  L = 2.75E-6
6  te = 1.0
7  ns = 500
8  T = np.linspace(0,te,ns)
9  Input = np.array([(np.radians(45)*min((t-0.25)/0.005,1)) if t> 0.25 else 0 for t in T])
10 np.random.seed(0)
11 Perturbation = np.array([np.random.uniform(-1,1)*i*0.1 for i in Input])
12 s = cn.TransferFunction.s
13 TF = K/(s*((J*s+b)*(L*s+R)+K**2))
```

Once we define the input and the plant's transfer function, we describe the controller's structure. In this case, a feedback controller with a fuzzy control system (Figure 11).

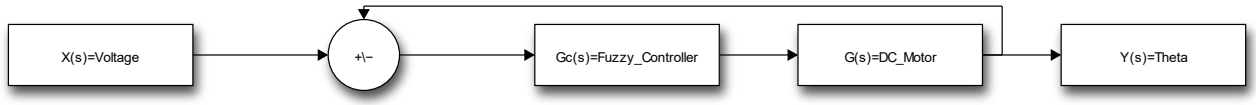

Figure 11. Feedback controller for controlling position in a DC motor with a fuzzy system.

Moreover, we define a control structure, including random perturbations expected as a 10% reference input or $X(s)$, to test the fuzzy controllers' robustness (Figure 12).

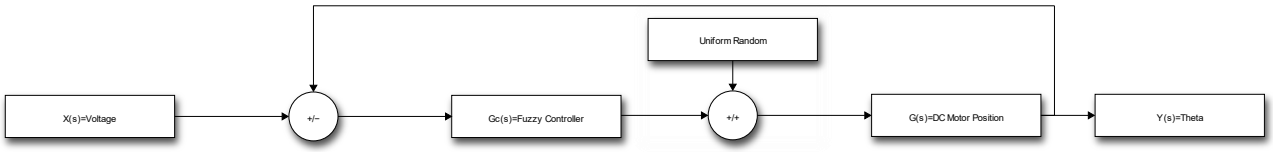

Figure 12. Feedback controller for controlling position in a DC motor with a fuzzy system and random disturbances.

3.4.1. One-Input Mamdani Fuzzy Controller

After defining the system, we design the fuzzy controller with premises and consequences concerning the error and the control behavior. Therefore, we implement these premises following the approach in Code 2. First, defining a single error input with a universe in the ranges $[-100, 100]$ of the angular position in rad and then a single output controller with a universe $[-20, 20]$ volt (V). Figure 13 shows the premise, and Figure 14 shows the consequence.

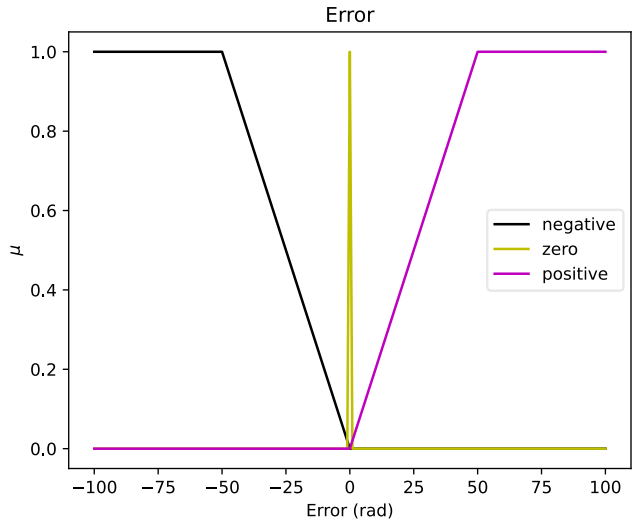

Figure 13. Error premise for position control in a DC motor.

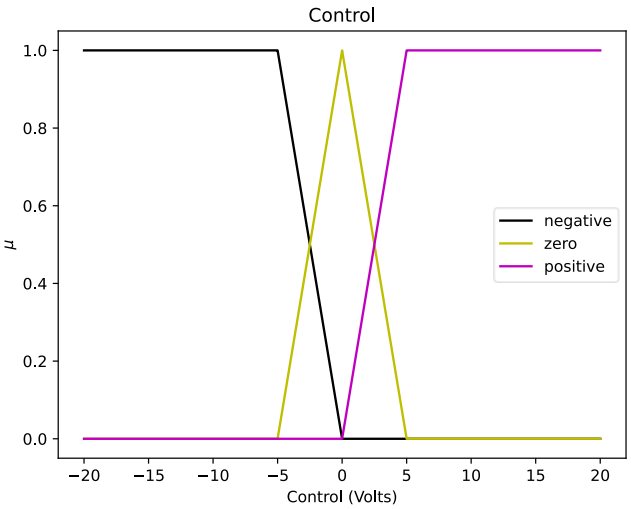

Figure 14. Control consequence for position control in a DC motor.

Subsequently obtaining premises and consequences, we define the rules for the FIS using the approach described in Code 3. The fuzzy rules used for the controller are:

1. IF **Error** is **Negative** → **Control** is **Negative;**
2. IF **Error** is **Zero** → **Control** is **Zero**;
3. IF **Error** is **Positive** → **Control** is **Positive.**

Next, we set the system configuration given in Table 2 for a Mamdani inference system. Then, we simulate the FIS. The surface generated for the simulation of the rules is shown in Figure 15.

Then, using the inference system defined and stored in the variable Mamdani1, we create an instance of the class fuzzy_controller, passing the inference system, the type of inference system (in this case, fuzzy with one input), the transfer function, and the sampling time, as in line 1 of Code 5. Line 2 builds the controller. Line 3 provides the fuzzy controller for simulation with processes defined in the space of the state model. Finally, line 4 returns the system with the transfer function for simulation.

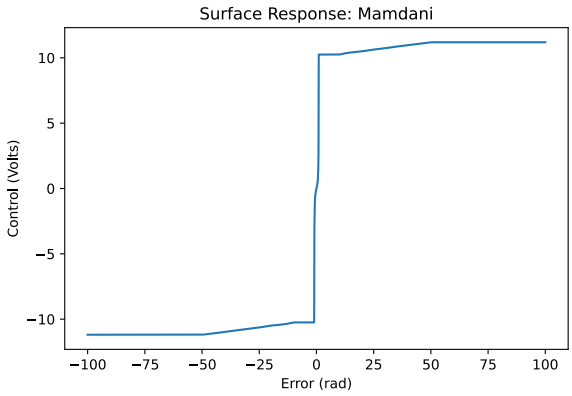

Figure 15. Surface response of Mamdani fuzzy controller for controlling position in a DC motor.

Code 5. Python code for defining fuzzy controller with UPAFuzzySystems library.

```
1 FuzzController = UPAfs.fuzzy_controller(Mamdani1,typec='Fuzzy1',tf=TF,DT = T[1])
2 FuzzController.build()
3 FuzzControllerBlock = FuzzController.get_controller()
4 FuzzSystemBlock = FuzzController.get_system()
```

Once we obtained the system with the transfer function for simulation in line 4 of Code 5, we solved the simulation using the control library in line 1 and plotted the results in lines 2–7 of Code 6.

Code 6. Python code for simulation of fuzzy controller and plotting results.

```
1 T, Theta = cn.input_output_response(FuzzSystemBlock,T,Input,0)
2 plt.plot(T,Theta,label='Process Variable')
3 plt.plot(T,Input,label='Reference')
4 plt.xlabel("time (s)")
5 plt.ylabel("position (rad)")
6 plt.legend()
7 plt.show()
```

Figure 16 shows the simulation results of the DC motor position (blue) controlled with the fuzzy controller while summited to the input created (orange). The absolute error is 1.92×10^{-11}, and the absolute percentage error is 2.45×10^{-9}% in steady conditions. The maximum value reached is 8.47×10^{-1} rad, the minimum value is 0.00 rad, the start value is 0.00 rad, the end value is 7.85×10^{-1} rad, the time rising is 3.86×10^{-2} s, the overshoot is 7.87%, the time peak is 7.46×10^{-2} s, and the settling time is 1.25×10^{-1} s.

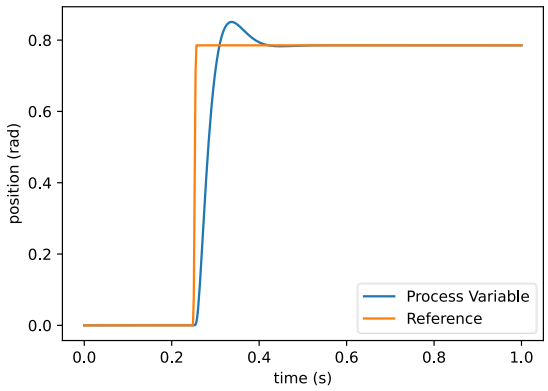

Figure 16. DC motor position controlled with a Mamdani defined with UPAFuzzySystems library.

Similarly, we test the Mamdani controller with one input, but this time including the disturbances at the input of the plant as in the block diagram in Figure 12. The obtained results are shown in Figure 17.

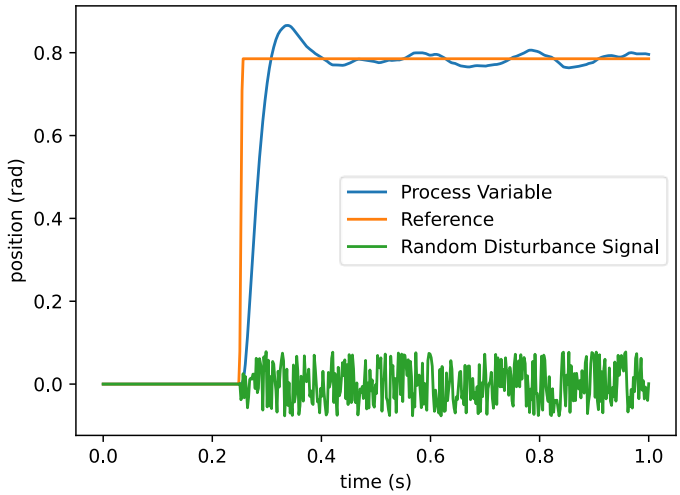

Figure 17. DC motor position controlled with a Mamdani controller, including disturbances in the input of the plant.

Figure 17 shows the simulation results of the DC motor position (blue) controlled with the fuzzy controller while summited to the input created (orange) and the disturbance signal (green). The absolute error is 1.20×10^{-2}, and the absolute percentage error is 1.53% in steady conditions. The maximum value reached is 8.60×10^{-1} rad, the minimum value is 0.00 rad, the start value is 0.00 rad, the end value is 7.97×10^{-1} rad, the time rising is 3.86×10^{-2} s, the overshoot is 7.86%, the time peak is 7.26×10^{-2} s, and the settling time is 6.38×10^{-1} s.

3.4.2. One-Input FLS Controller

Using the same rules as for the Mamdani controller with one input and the same fuzzy sets, we define the FLS controller. Nevertheless, we configure the UPAFuzzySystems library to work with the FLS structure and modify the connectives and implications described in Table 2. Code 7 shows the lines that define the inference system for the FLS controller. Line 7 is the one that differs from a Mamdani controller and configures the inference system for the FLS controller structure.

Code 7. Lines for defining one input FLS controller in the UPAFuzzySystems library.

```
1  FLS1 = UPAfs.inference_system('FLS controller')
2  FLS1.add_premise(Error_universe)
3  FLS1.add_consequence(Control_universe)
4  FLS1.add_rule([['Error','negative']],[],[['Control','negative']])
5  FLS1.add_rule([['Error','zero']],[],[['Control','zero']])
6  FLS1.add_rule([['Error','positive']],[],[['Control','positive']])
7  FLS1.configure('FLSmidth')
8  FLS1.build()
```

For coding the FLS inference system, we simulate the system's surface over the input universe using the method described in Code 3 and obtain the results shown in Figure 18.

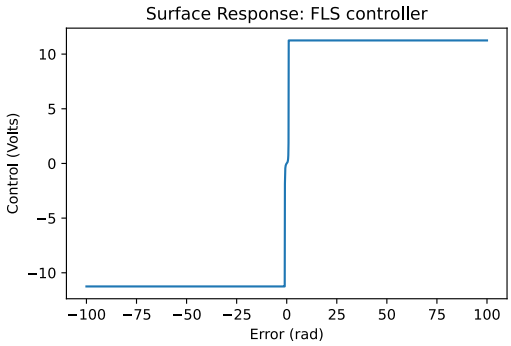

Figure 18. Surface obtained with the one-input FlS controller.

In the next step, we build and simulate the controller using the methods in Code 5 and Code 6. The simulation results show a smoother response than the Mamdani controller and avoid overshooting the system response (Figure 19). The absolute error is 1.35×10^{-3}, and the absolute percentage error is 1.72×10^{-1}% at steady state. The maximum value achieved is 7.84×10^{-1} rad, the minimum value -2.39×10^{-16} rad, the start value -8.40×10^{-17} rad, the final value 7.84×10^{-1} rad, the rise time 1.71×10^{-1} s, the overshoot 0.00%, the peak time 7.22×10^{-1} s, and the settling time 3.71×10^{-1} s.

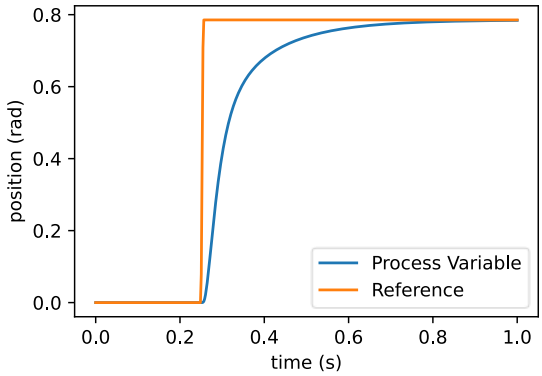

Figure 19. DC motor position controlled with an FLS controller defined with UPAFuzzySystems library.

Similarly, we test the FLS controller with one input, including the disturbances at the input of the plant, as in the block diagram in Figure 12. The results are shown in Figure 20.

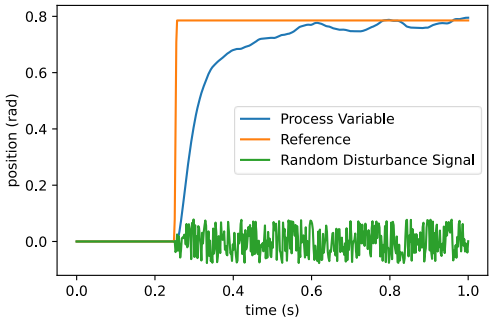

Figure 20. DC motor position controlled with an FLS controller, including disturbance signal.

Figure 20 shows the results of the FLS controller with one input and signal disturbance for the DC motor position (blue) while summited to the input created (orange) and the disturbance signal (green). The absolute error is 7.18×10^{-3}, and the absolute percentage error is 9.14×10^{-1}% in steady conditions. The maximum value reached is 7.93×10^{-1} rad, the minimum value is -2.39×10^{-16} rad, the start value is -8.40×10^{-17} rad, the end value

is 7.93×10^{-1} rad, the time rising is 1.99×10^{-1} s, the overshoot is 0.00%, the time peak is 7.22×10^{-1} s, and the settling time is 6.80×10^{-1} s.

3.4.3. One Input Takagi–Sugeno Controller

In this case, we define a control sequence in the Takagi–Sugeno configuration, as shown in Table 2. The premise is the error as in the Mamdani and FLS controllers with one input, but now we define two fuzzy sets. Therefore, the consequent definition has two functions related to the error in the premises. Afterward, we introduce the rules with the inference system class. The Python code to define the fuzzy universe with fuzzy sets and the inference system for the Takagi–Sugeno controller with one input is in Code 8.

Lines 1–4 of Code 8 define the fuzzy sets of the premise error using the same procedure as for the Mamdani and FLS controllers. Next, lines 10 and 13 specify the fuzzy sets consequence with second-order functions defined concerning the input denoted by x. Then, lines 19–23 describe the premises, consequences, and rules. After that, line 25 configures the FIS to work with the Takagi–Sugeno structure in Table 2. Finally, line 27 builds the controller.

Code 8. Python code for the fuzzy universe and inference system for the one-input Takagi–Sugeno controller.

```
1  Error_universe = UPAfs.fuzzy_universe('Error', np.arange(-100,101,1), 'continuous')
2  Error_universe.add_fuzzyset('negative','trimf',[-200,-100,100])
3  Error_universe.add_fuzzyset('positive','trimf',[-100,100,200])
4  Error_universe.view_fuzzy()
5  ax = plt.gca()
6  ax.set_xlabel("Error (rad)")
7  ax.set_ylabel("$\mu$")
8  plt.show()
9
10 Control_universe = UPAfs.fuzzy_universe('Control', np.arange(-20,22,2), 'continuous')
11 Control_universe.add_fuzzyset('negative','eq','-0.001*(x[0])**2+0.4*x[0]')
12 Control_universe.add_fuzzyset('positive','eq','0.001*(x[0])**2+0.4*x[0]')
13 Control_universe.view_fuzzy()
14 ax = plt.gca()
15 ax.set_xlabel("Control (Volts)")
16 ax.set_ylabel("$\mu$")
17 plt.show()
18
19 TSG1 = UPAfs.inference_system('Takagi-Sugeno One Input')
20 TSG1.add_premise(Error_universe)
21 TSG1.add_consequence(Control_universe)
22 TSG1.add_rule([['Error','negative']],[],[['Control','negative']])
23 TSG1.add_rule([['Error','positive']],[],[['Control','positive']])
24
25 TSG1.configure('Sugeno')
26
27 TSG1.build()
```

Figures 21 and 22 display the plots of the premise and consequences of the fuzzy sets in the one-input Takagi–Sugeno controller. The consequence plot shows the functions depending on the error premise.

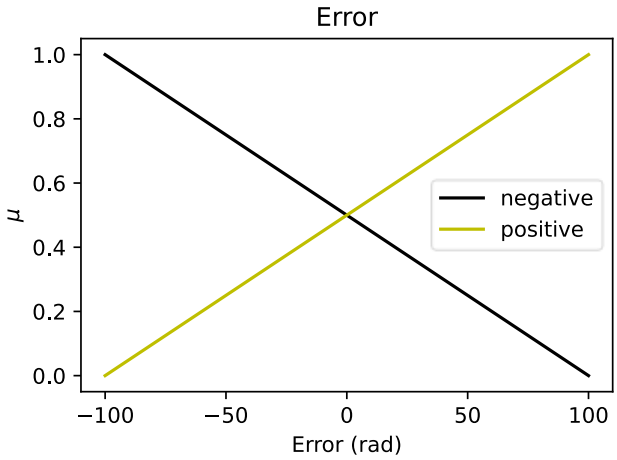

Figure 21. Premise fuzzy sets of the one-input Takagi–Sugeno controller.

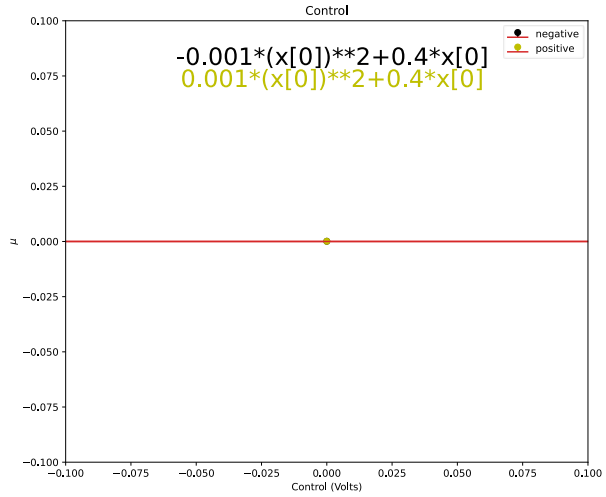

Figure 22. Consequence fuzzy sets of the one-input Takagi–Sugeno controller.

The surface simulation using the method described in Code 3 is shown in Figure 23, where the output is a second-order response resulting from the second-order functions in the output universe.

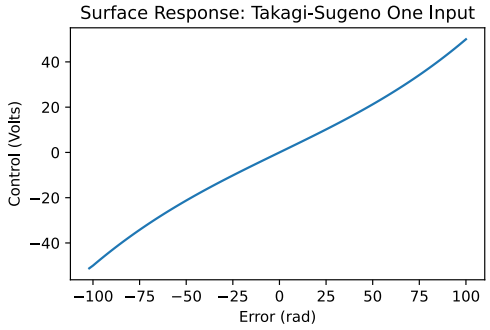

Figure 23. Surface response of the one-input Takagi–Sugeno controller.

Finally, we configure the controller using the same method as in Code 5 and present the simulation results with the approach in Code 6. Figure 24 shows the results of the simulation. The absolute error is 1.22×10^{-8}, and the absolute percentage error is 1.55×10^{-6}% under stable conditions. The maximum value obtained is 7.85×10^{-1} rad, the minimum value is 0.00 rad, the start value is 0.00 rad, the end value is 7.85×10^{-1} rad, the rise time is

1.25×10^{-1} s, the overshoot is 0.00%, the peak time is 7.22×10^{-1} s, and the settling time is 1.91×10^{-1} s.

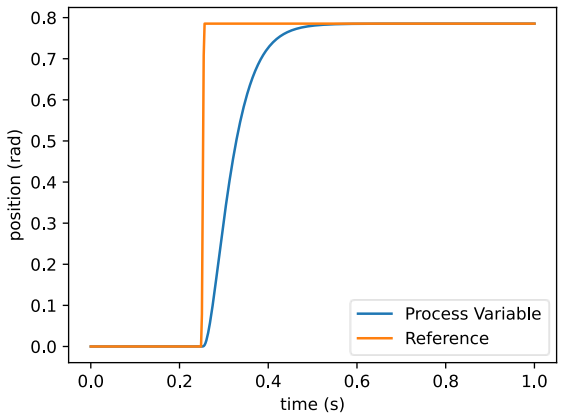

Figure 24. Controller response of the one-input Takagi–Sugeno controller.

Again, we test the Takagi–Sugeno controller with one input, including the disturbance at the input of the plant, as in the block diagram in Figure 12. The results are in Figure 25.

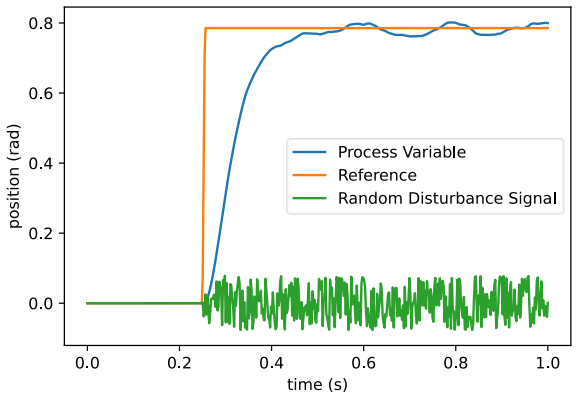

Figure 25. Controller response of the one-input Takagi–Sugeno controller with disturbances.

Figure 25 shows the results of the Takagi–Sugeno controller with one input and signal disturbance for the DC motor position (blue) while summited to the input created (orange) and the disturbance signal (green). The absolute error is 1.35×10^{-2}, and the absolute percentage error is 1.72% in steady conditions. The maximum value reached is 8.00×10^{-1} rad, the minimum value is 0.00 rad, the start value is 0.00 rad, the end value is 7.99×10^{-1} rad, the time rising is 1.33×10^{-1} s, the overshoot is 1.60×10^{-1}%, the time peak is 5.28×10^{-1} s, and the settling time is 6.54×10^{-1} s.

3.4.4. Two-Input Mamdani Controller

The definition of the two-input Mamdani controller implies the definition of two premises, the first for the error and the second as the change in it or its derivative. Therefore, we specify three universes: one for the error premise, one for the change in the error premise, and another for the control consequence. The required code uses the same approach as in Code 2. The premises and consequences for the two-input Mamdani controller are in Figures 26–28.

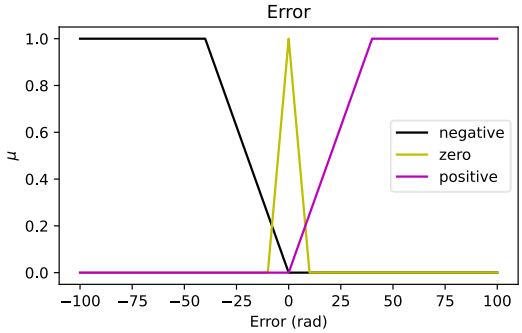

Figure 26. Error universe as a premise for a two-input Mamdani controller.

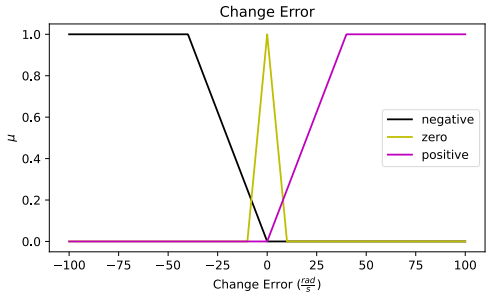

Figure 27. Change in error universe as a premise for a two-input Mamdani controller.

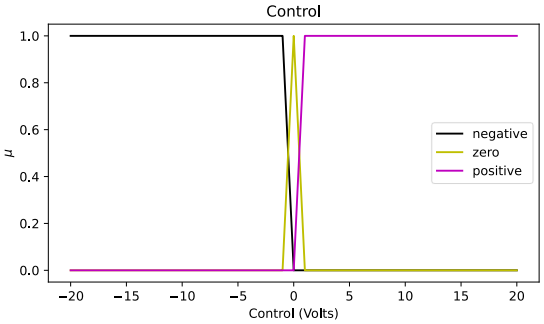

Figure 28. Control universe consequence for a two-input Mamdani controller.

Having specified the premises and consequences of the two-input Mamdani controller, we define the nine rules in the instance of the inference_system class of the UPAFuzzySystems library for each possible combination between inputs, following the structure described in Table 3. Thus, the specified rules are:

1. If **error** is **Neg** and **change error** is **Neg** then **control** is **Neg**;
2. If **error** is **Neg** and **change error** is **Zero** then **control** is **Neg**;
3. If **error** is **Zero** and **change error** is **Neg** then **control** is **Zero**;
4. If **error** is **Neg** and **change error** is **Pos** then **control** is **Zero**;
5. If **error** is **Zero** and **change error** is **Zero** then **control** is **Zero**;
6. If **error** is **Zero** and **change error** is **Pos** then **control** is **Zero**;
7. If **error** is **Pos** and **change error** is **Neg** then **control** is **Zero**;
8. If **error** is **Pos** and **change error** is **Zero** then **control** is **Pos**;
9. If **error** is **Pos** and **change error** is **Pos** then **control** is **Pos**.

The code to define these rules in the proposed library with the two-input Mamdani controller is in Code 9.

Code 9. Python code with rules for a two-input Mamdani controller using the UPAFuzzySystems library.

```
Mamdani2 = UPAfs.inference_system('Mamdani')
Mamdani2.add_premise(Error_universe)
Mamdani2.add_premise(ChError_universe)
Mamdani2.add_consequence(Control_universe)
Mamdani2.add_rule([['Error','negative'],['Change Error','negative']],['and'],[['Control','negative']])
Mamdani2.add_rule([['Error','negative'],['Change Error','zero']],['and'],[['Control','negative']])
Mamdani2.add_rule([['Error','zero'],['Change Error','negative']],['and'],[['Control','zero']])
Mamdani2.add_rule([['Error','negative'],['Change Error','positive']],['and'],[['Control','zero']])
Mamdani2.add_rule([['Error','zero'],['Change Error','zero']],['and'],[['Control','zero']])
Mamdani2.add_rule([['Error','positive'],['Change Error','negative']],['and'],[['Control','zero']])
Mamdani2.add_rule([['Error','zero'],['Change Error','positive']],['and'],[['Control','zero']])
Mamdani2.add_rule([['Error','positive'],['Change Error','zero']],['and'],[['Control','positive']])
Mamdani2.add_rule([['Error','positive'],['Change Error','positive']],['and'],[['Control','positive']])
Mamdani2.configure('Mamdani')
Mamdani2.build()
```

Then, we obtain the surface response for the two-input Mamdani controller by simulating the error and the change in error inputs as defined in Code 3. Nevertheless, we now use two inputs for the surface simulation. Figure 29 shows the simulated surface.

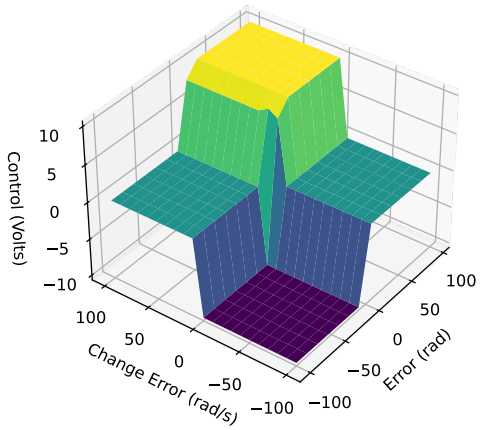

Figure 29. Two-input Mamdani controller surface using the UPAFuzzySystems library.

After defining the rules and simulating the surface, we configure the fuzzy_controller as before, but this time, we set the type of the controller to Fuzzy2, since the controller now has two inputs. The Python code for the configuration changes only in line one with the Fuzzy2 definition (Code 10).

Code 10. Python code for configuring two-input Mamdani controller using the UPAFuzzySystems library.

```
MamdaniController = UPAfs.fuzzy_controller(Mamdani2,typec='Fuzzy2',tf=TF,DT = T[1])
MamdaniController.build()
MamdaniControllerBlock = MamdaniController.get_controller()
MamdaniSystemBlock = MamdaniController.get_system()
```

Finally, we simulate the controller using the approach described in Code 6. Figure 30 shows the obtained controller's response. The absolute error is 2.00×10^{-6} rad, and the absolute percentage error is 2.54×10^{-4}% at steady state. The maximum value reached is 8.12×10^{-1} rad, the minimum value is -7.99×10^{-18} rad, the start value is -4.28×10^{-18} rad, the final value is 7.85×10^{-1} rad, the rise time is 2.66×10^{-2} s, the overshoot time is 3.33%, the peak time is 4.46×10^{-2} s, and the settling time is 1.47×10^{-1} s.

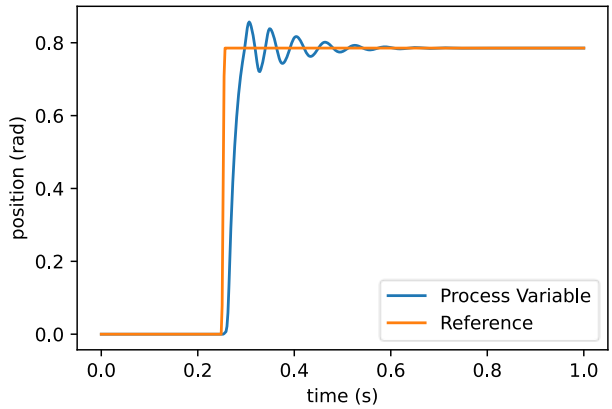

Figure 30. Two-input Mamdani controller response using the UPAFuzzySystems library.

Similarly, we test the Mamdani controller with two inputs, including the disturbances at the input of the plant, as shown in the block diagram in Figure 12. The results are shown in Figure 31.

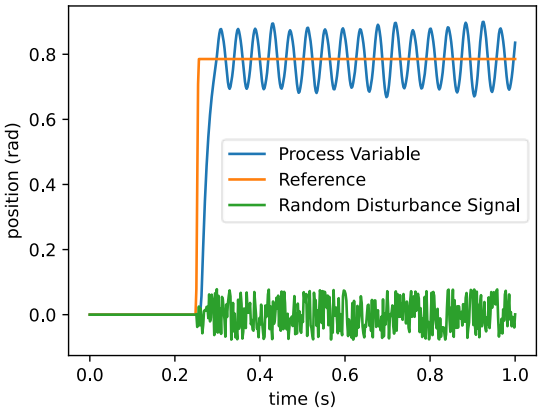

Figure 31. Two-input Mamdani controller response with disturbances.

Figure 31 shows the results of the Mamdani controller with two inputs and signal disturbance for the DC motor position (blue) while summited to the input created (orange) and the disturbance signal (green). The absolute error is 2.88×10^{-2}, and the absolute percentage error is 3.66% in steady conditions. The maximum value reached is 8.25×10^{-1} rad, the minimum value is -7.99×10^{-18} rad, the start value is -4.28×10^{-18} rad, the end value is 7.57×10^{-1} rad, the time rising is 2.66×10^{-2} s, the overshoot is 9.09%, the time peak is 6.62×10^{-1} s, and the settling time is 7.14×10^{-1} s.

3.4.5. Two-Input FLS Controller

Using the same rules as for the two-input Mamdani controller and the same fuzzy sets, we define the two-input FLS controller. Nevertheless, we configure the UPAFuzzySystems library to work with the FLS structure and change the connectives and implications described in Table 2. Figure 32 displays the simulated surface for the two-input FLS controller.

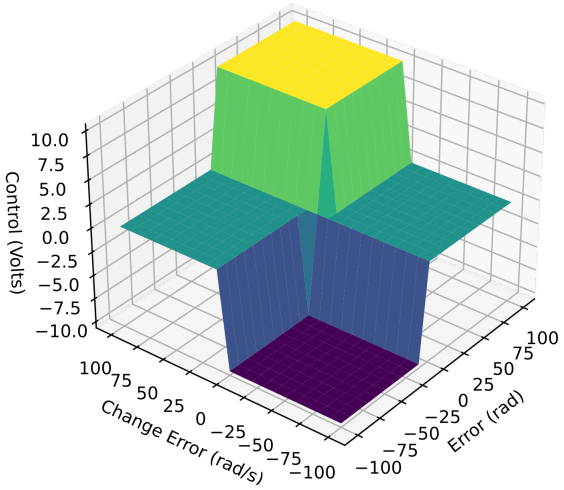

Figure 32. Two-input FLS controller surface with the UPAFuzzySystems library.

Finally, we simulate the FLS controller using the approach described in Code 6. Figure 33 represents the controller's response obtained. This time, the absolute error is 3.59×10^{-4}, and the absolute percentage error is 4.58×10^{-2}%. The maximum value reached is 7.85×10^{-1} rad, the minimum value is 0.00 rad, the start value is 0.00 rad, the final value is 7.85×10^{-1} rad, the rise time is 2.37×10^{-1} s, the overshoot is 0.00 %, the time peak is 7.22×10^{-1} s, and the settling time is 3.79×10^{-1} s.

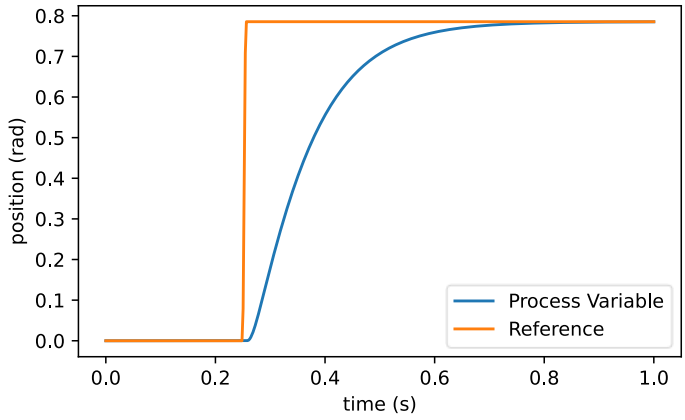

Figure 33. Two-input FLS controller response by the UPAFuzzySystems library.

Again, we test the FLS controller with two inputs, including the disturbances at the input of the plant, as shown in the block diagram in Figure 12. The results are shown in Figure 34.

Figure 34 shows the results of the FLS controller with two inputs and signal disturbance for the DC motor position (blue) while summited to the input created (orange) and the disturbance signal (green). The absolute error is 1.11×10^{-2}, and the absolute percentage error is 1.41% in steady conditions. The maximum value reached is 7.96×10^{-1} rad, the minimum value is -5.79×10^{-5} rad, the start value is 0.00 rad, the end value is 7.96×10^{-1} rad, the time rising is 2.67×10^{-1} s, the overshoot is 0.00 %, the time peak is 7.22×10^{-1} s, and the settling time is 6.74×10^{-1} s.

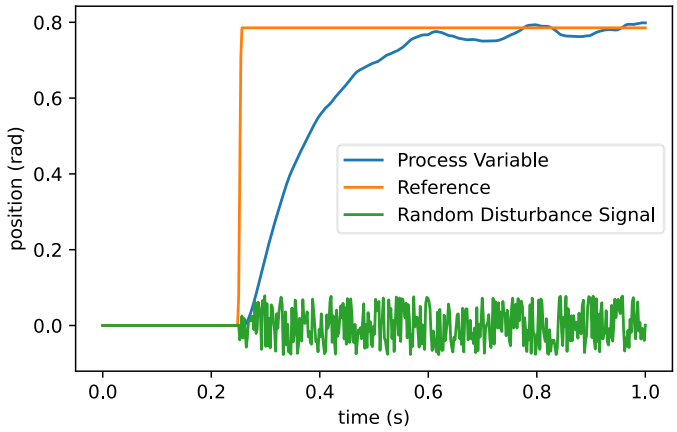

Figure 34. Two-input FLS controller response with disturbances.

3.4.6. Two Inputs Takagi–Sugeno Controller

In this case, we define the control consequence in the Takagi–Sugeno configuration for two inputs, as described in Table 1. The premises are identical to the Mamdani and FLS controllers with two inputs. Therefore, the consequence definition now has three functions related to the error and change in error premises. Next, we introduce the rules with the inference system class. Again, these are the same as the Mamdani and FLS controllers with two inputs. The Python code to define the fuzzy universe with fuzzy sets and the inference system for the Takagi–Sugeno controller with two inputs is in Code 11.

Figure 35 displays the output consequence, which differs from the two-input Mamdani and FLS controllers, in that the output premises are functions of the error and its change. Here, the premises are identical to those in Figures 26 and 27.

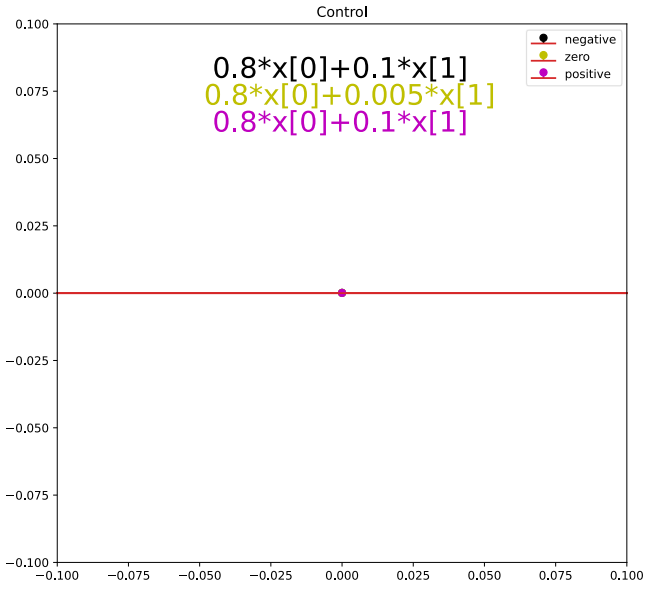

Figure 35. Consequence of the two-input Takagi–Sugeno controller by the UPAFuzzySystems library.

After defining the universes and configuring the FIS, we simulate the Takagi–Sugeno surface reaction with two inputs according to the method in Code 3. The results are in Figure 36.

Code 11. Code for defining the inference system and the fuzzy universe for the two-input Takagi–Sugeno controller.

```
Error_universe = UPAfs.fuzzy_universe('Error', np.arange(-100,101,1), 'continuous')
Error_universe.add_fuzzyset('negative','trapmf',[-100,-100,-40,0])
Error_universe.add_fuzzyset('zero','trimf',[-10,0,10])
Error_universe.add_fuzzyset('positive','trapmf',[0,40,100,100])
Error_universe.view_fuzzy()

ChError_universe = UPAfs.fuzzy_universe('Change Error', np.arange(-100,101,1), 'continuous')
ChError_universe.add_fuzzyset('negative','trapmf',[-100,-100,-40,0])
ChError_universe.add_fuzzyset('zero','trimf',[-10,0,10])
ChError_universe.add_fuzzyset('positive','trapmf',[0,40,100,100])
ChError_universe.view_fuzzy()

Control_universe = UPAfs.fuzzy_universe('Control', np.arange(-20,22,2), 'continuous')
Control_universe.add_fuzzyset('negative','eq','0.8*x[0]+0.1*x[1]')
Control_universe.add_fuzzyset('zero','eq','0.8*x[0]+0.005*x[1]')
Control_universe.add_fuzzyset('positive','eq','0.8*x[0]+0.1*x[1]')
Control_universe.view_fuzzy()

TSG2 = UPAfs.inference_system('Takagi-Sugeno Two Inputs')
TSG2.add_premise(Error_universe)
TSG2.add_premise(ChError_universe)
TSG2.add_consequence(Control_universe)

TSG2.add_rule([['Error','negative'],['Change Error','negative']],['and'],[['Control','negative']])
TSG2.add_rule([['Error','negative'],['Change Error','zero']],['and'],[['Control','negative']])
TSG2.add_rule([['Error','zero'],['Change Error','negative']],['and'],[['Control','zero']])
TSG2.add_rule([['Error','negative'],['Change Error','positive']],['and'],[['Control','zero']])
TSG2.add_rule([['Error','zero'],['Change Error','zero']],['and'],[['Control','zero']])
TSG2.add_rule([['Error','positive'],['Change Error','negative']],['and'],[['Control','zero']])
TSG2.add_rule([['Error','zero'],['Change Error','positive']],['and'],[['Control','zero']])
TSG2.add_rule([['Error','positive'],['Change Error','zero']],['and'],[['Control','positive']])
TSG2.add_rule([['Error','positive'],['Change Error','positive']],['and'],[['Control','positive']])

TSG2.configure('Sugeno')

TSG2.build()
```

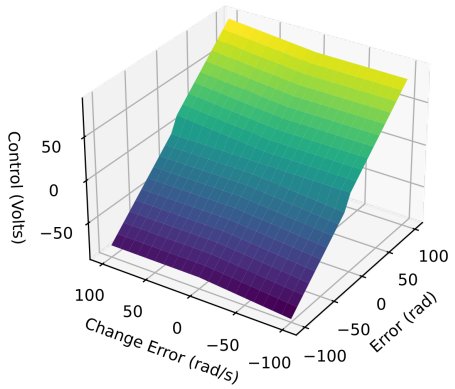

Figure 36. Surface of the two-input Takagi–Sugeno controller using the UPAFuzzySystems library.

We then define the controller configuration according to Code 5, modified for the Takagi–Sugeno structure, and plot the results using the approach in Code 6. Figure 37 shows the controller response. The absolute error is 3.16×10^{-12}, and the absolute percentage error is 4.02×10^{-10}% under stable conditions. The maximum value obtained is 7.96×10^{-1} rad,

the minimum value is 0.00 rad, the start value is 0.00 rad, the final value is 7.85×10^{-1} rad, the rise time is 6.66×10^{-2} s, the overshoot is 1.37%, the time peak is 1.21×10^{-1} s, and the settling time is 8.47×10^{-2} s.

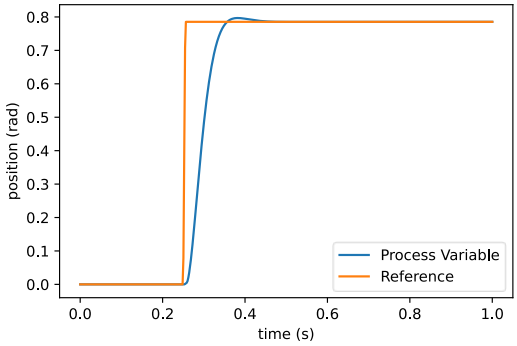

Figure 37. Control response of the two-input Takagi–Sugeno controller using the UPAFuzzySystems library.

Similarly, we test the Takagi–Sugeno controller with two inputs, including the disturbances at the input of the plant, as shown in the block diagram in Figure 12. The results are in Figure 38.

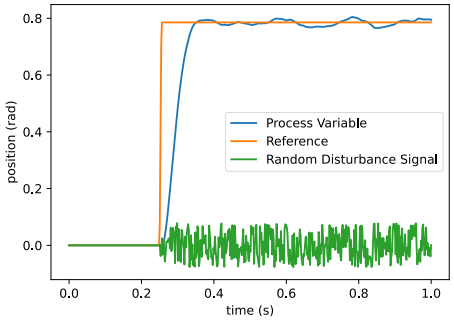

Figure 38. Control response of the two-input Takagi–Sugeno controller with disturbances.

Figure 38 shows the results of the two-input Takagi–Sugeno controller with signal disturbance for the DC motor position (blue) while summited to the input created (orange) and the disturbance signal (green). The absolute error is 1.07×10^{-2}, and the absolute percentage error is 1.37% in steady conditions. The maximum value reached is 8.02×10^{-1} rad, the minimum value is 0.00 rad, the start value is 0.00 rad, the end value is 7.96×10^{-1} rad, the time rising is 6.66×10^{-2} s, the overshoot is 7.07×10^{-1}%, the time peak is 5.22×10^{-1} s, and the settling time is 6.34×10^{-1} s.

3.5. P, PD, and PID Fuzzy Controllers

3.5.1. Linear P Fuzzy Controller

The UPAFuzzySystems library also contains the linear P controller with the structure shown in Figure 5. To work with this structure, the configured inference base configured is linear. Therefore, the fuzzy_universe and inference_system classes configure the premises, consequences, and rules to define a linear controller. Since the P fuzzy controller requires no change in error input, the linear system definition uses only a premise in the error case and a consequence controller, which are comparable in Table 2. Code 12 provides the premise, consequence, and required rules.

Code 12. Code for defining the one-input fuzzy linear system used in the linear P fuzzy controller.

```
Error_universe = UPAfs.fuzzy_universe('Error', np.arange(-100,101,1), 'continuous')
Error_universe.add_fuzzyset('negative','trimf',[-200,-100,100])
Error_universe.add_fuzzyset('positive','trimf',[-100,100,200])
Error_universe.view_fuzzy()
ax = plt.gca()
ax.set_xlabel("Error (rad)")
ax.set_ylabel("$\mu$")
plt.show()

Control_universe = UPAfs.fuzzy_universe('Control', np.arange(-200,202,2), 'continuous')
Control_universe.add_fuzzyset('negative',' eq','-200')
Control_universe.add_fuzzyset('positive','eq','200')
Control_universe.view_fuzzy()
ax = plt.gca()
ax.set_xlabel("Control (Volts)")
ax.set_ylabel("$\mu$")
plt.show()

LinearP = UPAfs.inference_system('Linear One Input')
LinearP.add_premise(Error_universe)
LinearP.add_consequence(Control_universe)
LinearP.add_rule([['Error','negative']],[],[['Control','negative']])
LinearP.add_rule([['Error','positive']],[],[['Control','positive']])

LinearP.configure('Linear')

LinearP.build()
```

Figures 39 and 40 display the premise and consequence plot for the linear system defined for working with the linear fuzzy P controller according to Table 2.

Employing the same approach in Code 3, we simulate the linear system's surface, which must be a line, as shown in Figure 41.

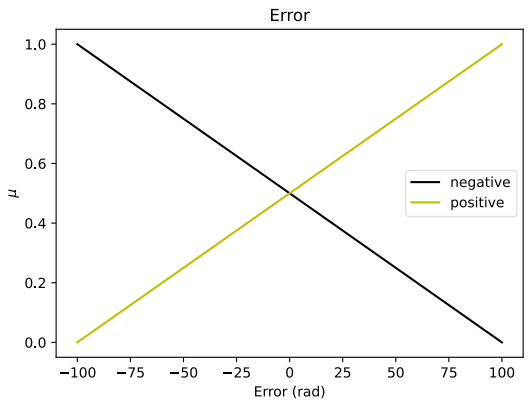

Figure 39. Premise of the one-input linear controller via the UPAFuzzySystems library.

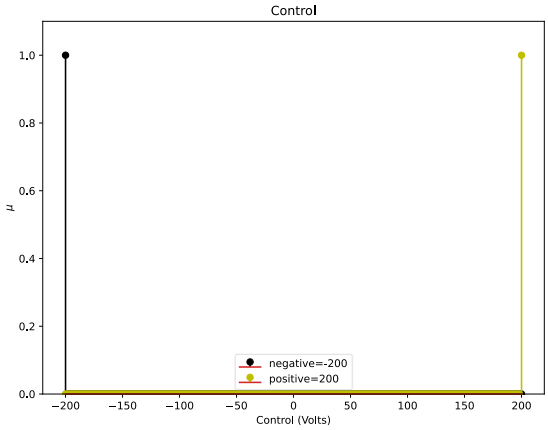

Figure 40. Consequence of the one-input linear controller using the UPAFuzzySystems library.

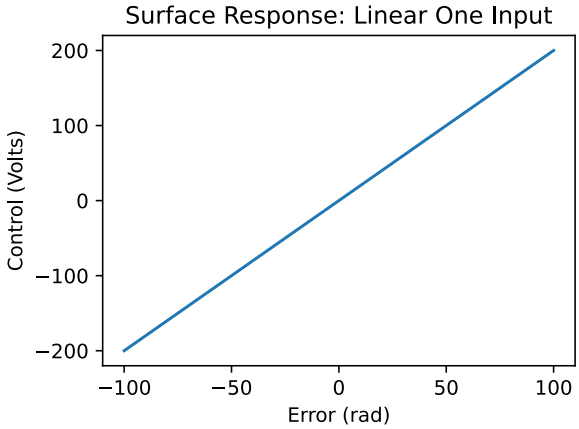

Figure 41. Surface of the one-input linear controller using the UPAFuzzySystems library.

When the linear inference system is defined, the linear P fuzzy controller must include the ***GU*** and ***GE*** gains according to Equations (21) and (22). The Python code for configuring the controller with those gains is in Code 13.

Code 13. Python code for configuring the linear fuzzy proportional controller with the UPAFuzzySystems library.

```
LinearPFuzzController = UPAfs.fuzzy_controller(LinearP,typec='P',tf=TF,DT = T[1], GE=15.91545709, GU=0.094248)
LinearPFuzzController.build()
LinearPFuzzControllerBlock = LinearPFuzzController.get_controller()
LinearPFuzzSystemBlock = LinearPFuzzController.get_system()
```

After defining the controller, we simulate it following the method in Code 6. The controller response is shown in Figure 42. The absolute error is 1.26×10^{-9}, and the absolute percentage error is 1.60×10^{-7}% in steady conditions. The maximum value reached is 9.88×10^{-1} rad, the minimum value is 0.00 rad, the start value is 0.00 rad, the final value is 7.85×10^{-1} rad, the rise time is 1.45×10^{-2} s, the overshoot is $2.58 \times 10^{+1}$%, the time peak is 3.26×10^{-2} s, and the settling time is 1.27×10^{-1} s.

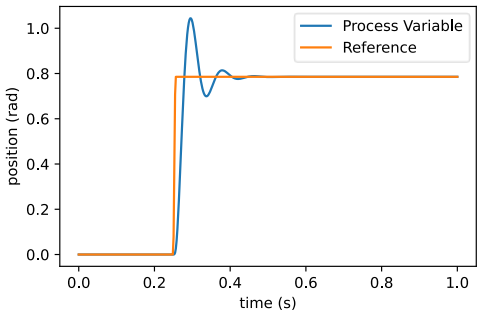

Figure 42. Response of the linear proportional fuzzy controller using the UPAFuzzySystems library.

Similarly, we test the linear proportional fuzzy controller, including the disturbances at the input of the plant, as shown in the block diagram in Figure 12. The results are in Figure 43.

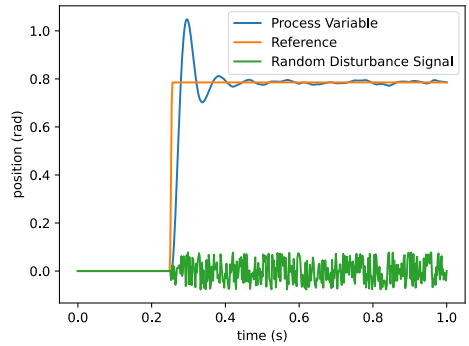

Figure 43. Response of the linear proportional fuzzy controller with disturbances.

Figure 43 shows the results of the linear proportional fuzzy controller with signal disturbance for the DC motor position (blue) while summited to the input created (orange) and the disturbance signal (green). The absolute error is 3.60×10^{-3}, and the absolute percentage error is 4.58×10^{-1}% in steady conditions. The maximum value reached is 9.93×10^{-1} rad, the minimum value is 0.00 rad, the start value is 0.00 rad, the end value is 7.89×10^{-1} rad, the time rising is 1.45×10^{-2} s, the overshoot is $2.58 \times 10^{+1}$%, the time peak is 3.46×10^{-2} s, and the settling time is 1.65×10^{-1} s.

3.5.2. Linear PD Fuzzy Controller

The Linear PD controller with the structure defined in Figure 6 implies working with a linear inference base with two inputs. Therefore, the class fuzzy_universe and inference_system configure the premises, consequences, and rules to define a two-input linear controller. The PD fuzzy controller requires two premises: the error and its change. A consequence controller, comparable in Table 2, obtains the premise, consequence, and rules needed for a two-input linear fuzzy system. Code 14 shows the complete implementation of the linear fuzzy system with two inputs.

Code 14. Code for defining the two-input fuzzy linear system with UPAFuzzySystems library.

```
1  Error_universe = UPAfs.fuzzy_universe('Error', np.arange(-100,101,1), 'continuous')
2  Error_universe.add_fuzzyset('negative','trimf',[-200,-100,100])
3  Error_universe.add_fuzzyset('positive','trimf',[-100,100,200])
4  Error_universe.view_fuzzy()
5  ax = plt.gca()
6  ax.set_xlabel("Error (rad)")
7  ax.set_ylabel("$\mu$")
8  plt.show()
9
10 ChError_universe = UPAfs.fuzzy_universe('Change Error', np.arange(-100,101,1), 'continuous')
11 ChError_universe.add_fuzzyset('negative','trimf',[-200,-100,100])
12 ChError_universe.add_fuzzyset('positive','trimf',[-100,100,200])
13 ChError_universe.view_fuzzy()
14 ax = plt.gca()
15 ax.set_xlabel(r"Change Error ($\frac{rad}{s}$)")
16 ax.set_ylabel("$\mu$")
17 plt.show()
18
19 Control_universe = UPAfs.fuzzy_universe('Control', np.arange(-200,202,2), 'continuous')
20 Control_universe.add_fuzzyset('negative',' eq','-200')
21 Control_universe.add_fuzzyset('zero','eq','0')
22 Control_universe.add_fuzzyset('positive','eq','200')
23 Control_universe.view_fuzzy()
24 ax = plt.gca()
25 ax.set_xlabel("Control (Volts)")
26 ax.set_ylabel("$\mu$")
27 plt.show()
28
29 Linear = UPAfs.inference_system('Linear')
30 Linear.add_premise(Error_universe)
31 Linear.add_premise(ChError_universe)
32 Linear.add_consequence(Control_universe)
33
34 Linear.add_rule([['Error','negative'],['Change Error','negative']],['and'],[['Control','negative']])
35 Linear.add_rule([['Error','negative'],['Change Error','positive']],['and'],[['Control','zero']])
36 Linear.add_rule([['Error','positive'],['Change Error','negative']],['and'],[['Control','zero']])
37 Linear.add_rule([['Error','positive'],['Change Error','positive']],['and'],[['Control','positive']])
38
39 Linear.configure('Linear')
40
41 Linear.build()
```

Lines 4 and 13 in Code 14 produce the plots of the error and change in error premises, as shown in Figures 44 and 45, respectively. The control consequence plot obtained with line 23 is shown in Figure 46.

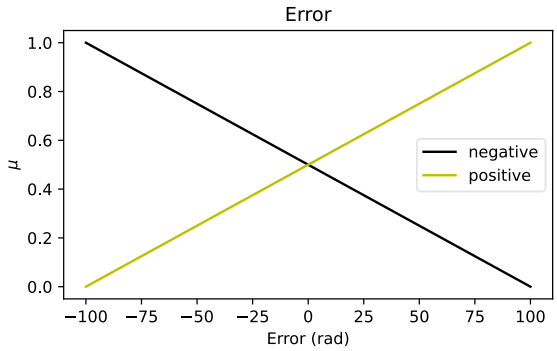

Figure 44. Premise error of the two-input linear controller using the UPAFuzzySystems library.

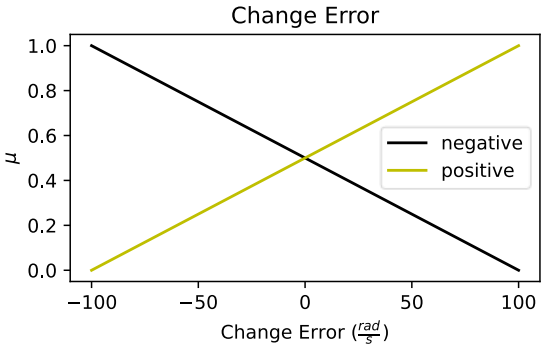

Figure 45. Premise change in error of the two-input linear controller using the UPAFuzzySystems library.

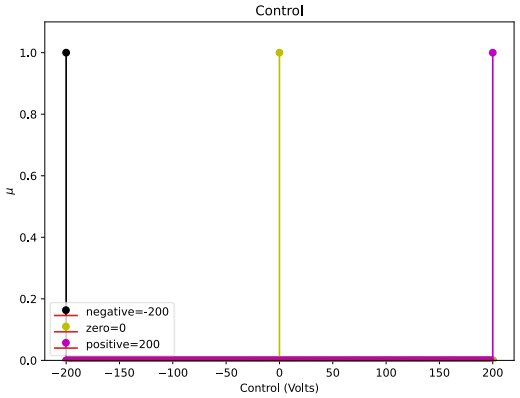

Figure 46. Consequence control of the two-input linear controller using the UPAFuzzySystems library.

After defining the two-input linear inference system, we simulate its surface following the method described in Code 3. Figure 47 shows the obtained surface with the two-input linear system.

After obtaining the linear inference system, the definition of the linear PD fuzzy controller must include the ***GU***, ***GE***, and ***GCE*** gains defined in Equations (21)–(23). The Python code configures the PD fuzzy controller with those gains in Code 15.

Finally, we simulate the linear PD fuzzy controller following the approach in Code 6. The response of the system is in Figure 48. The absolute error is 1.83×10^{-7}, and the absolute percentage error is 2.33×10^{-5}% in steady conditions. The maximum value achieved is 7.85×10^{-1} rad, the minimum value is 0.00 rad, the start value is 0.00 rad, the end value is 7.85×10^{-1} rad, the rise time is 3.86×10^{-2} s, the overshoot is 0.00%, the peak time is 7.22×10^{-1} s, and the settling time is 1.23×10^{-1} s.

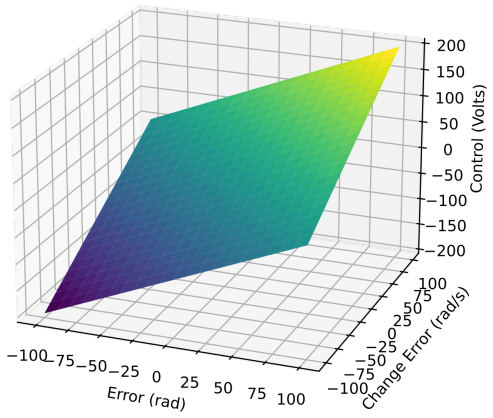

Figure 47. Surface of the two-input linear controller using the UPAFuzzySystems library.

Code 15. Python code for configuring the PD fuzzy controller in the UPAFuzzySystems library.

```
LinearPDFuzzController = UPAfs.fuzzy_controller(Linear,typec='PD',tf=TF,DT = T[1], GE=15.91545709,
GU=0.094248, GCE=0.636618283)
LinearPDFuzzController.build()
LinearPDFuzzControllerBlock = LinearPDFuzzController.get_controller()
LinearPDFuzzSystemBlock = LinearPDFuzzController.get_system()
```

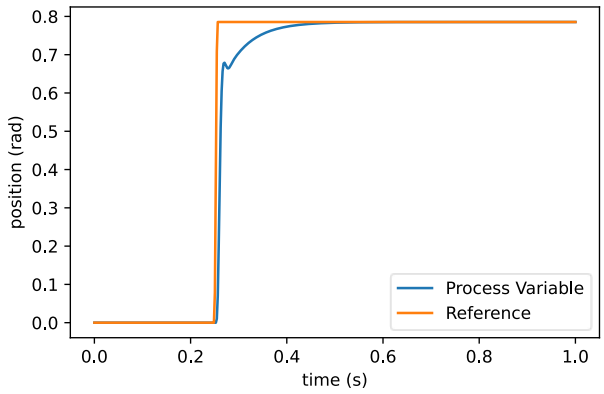

Figure 48. Response of the linear PD fuzzy controller using the UPAFuzzySystems library.

Again, we test the linear PD fuzzy controller, including the disturbances at the input of the plant, as shown in the block diagram in Figure 12. The results are shown in Figure 49.

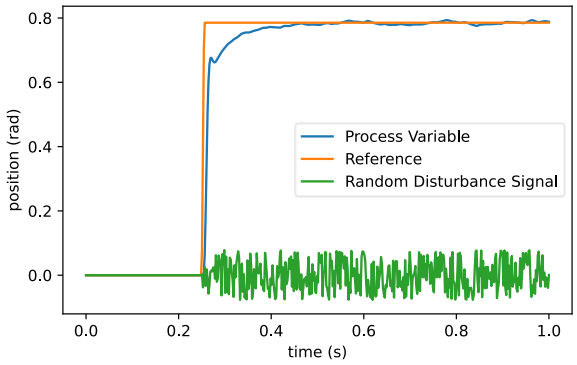

Figure 49. Response of the linear PD fuzzy controller with disturbances.

Figure 49 shows the results of the linear PD fuzzy controller with signal disturbance for the DC motor position (blue) while summited to the input created (orange) and the

disturbance signal (green). The absolute error is 3.47×10^{-3}, and the absolute percentage error is 4.42×10^{-1} % in steady conditions. The maximum value reached is 7.91×10^{-1} rad, the minimum value is 0.00 rad, the start value is 0.00 rad, the end value is 7.89×10^{-1} rad, the time rising is 3.86×10^{-2} s, the overshoot is 2.60×10^{-1}%, the time peak is 5.18×10^{-1} s, and the settling time is 1.61×10^{-1} s.

3.5.3. Linear PD-I Fuzzy Controller

The Linear PD-I fuzzy controller uses as a basis a two-input linear system as the linear PD controller. Therefore, we use the same system defined in Code 14. However, now, we add a third gain and the integration connection as shown in Figure 7. Thus, the simulation surface of the two-input linear system is the same as in Figure 47.

After that, we specify the configuration of the PD-I fuzzy controller with its gains ***GU***, ***GE***, ***GCE***, and ***GIE*** defined in Equations (21)–(24). The Python code for configuring the PD-I fuzzy controller with its gains is in Code 16.

Code 16. Configuration of PD-I fuzzy controller with the UPAFuzzySystems library.

```
1 LinearPidFuzzController = UPAfs.fuzzy_controller(Linear,typec='PD-I',tf=TF,DT = T[1],
  GE=15.91545709, GU=0.094248, GCE=0.636618283, GIE=7.234298678)
2 LinearPidFuzzController.build()
3 LinearPidFuzzControllerBlock = LinearPidFuzzController.get_controller()
4 LinearPidFuzzSystemBlock = LinearPidFuzzController.get_system()
```

Finally, we simulate the controller following the approach in Code 6. The output response is shown in Figure 50. The absolute error is 5.77×10^{-15}, and the absolute percentage error is 7.35×10^{-13}% in steady conditions. The maximum value achieved is 7.85×10^{-1} rad, the minimum value is 0.00 rad, the start value is 0.00 rad, the end value is 7.85×10^{-1} rad, the time rising is 1.05×10^{-2} s, the overshoot is 0.00%, the time peak is 7.22×10^{-1} s, and the settling time is 3.46×10^{-2} s.

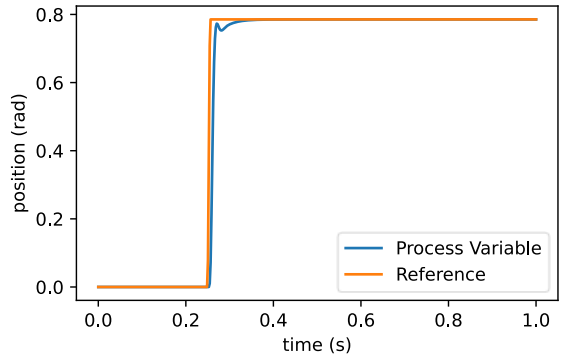

Figure 50. Response of the linear PD-I fuzzy controller using the UPAFuzzySystems library.

Similarly, we test the linear PD-I fuzzy controller, including the disturbances at the input of the plant, as shown in the block diagram in Figure 12. The results are in Figure 51.

Figure 51 shows the results of the linear PD-I fuzzy controller with signal disturbance for the DC motor position (blue) while summited to the input created (orange) and the disturbance signal (green). The absolute error is 2.05×10^{-3}, and the absolute percentage error is 2.61×10^{-1}% in steady conditions. The maximum value reached is 7.91×10^{-1} rad, the minimum value is 0.00 rad, the start value is 0.00 rad, the end value is 7.87×10^{-1} rad, the time rising is 1.05×10^{-2} s, the overshoot is 4.11×10^{-1}%, the time peak is 5.12×10^{-1} s, and the settling time is 3.46×10^{-2} s.

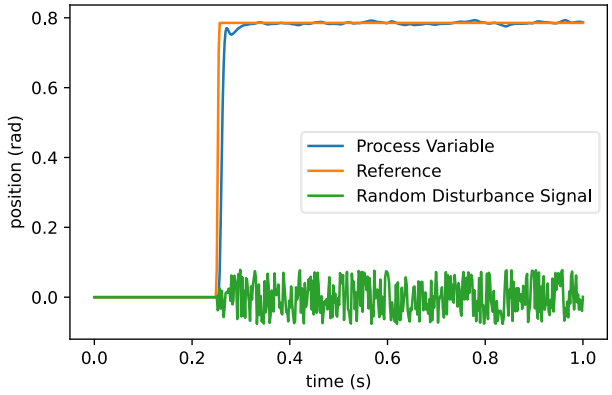

Figure 51. Response of the linear PD-I fuzzy controller with disturbances.

3.6. Controllers Comparison

Each fuzzy controller implemented using the UPAFuzzySystems library has different control characteristics produced in the control response. These characteristics are the absolute error (absolute_error), the absolute percentage error (percentage_error), the maximum value (max_value), the minimum value (min_value), the start value (start_value), the end value (end_value), the time rising (time_rising), the overshoot (overshoot), the time peak (time_overshoot), and the settling time (settling_time). Table 5 compares the control characteristics for each controller applied to the position control of a DC motor.

Additionally, Figure 52 compares graphically the control responses of all fuzzy controllers implemented following an input or reference with an angular position of 45 degrees (7.85×10^{-1} rad). As the figure shows, there are variations in the overshoot, time rising, settling time, and time peak. Still, all the controllers maintain an error below 1% of the reference in steady conditions.

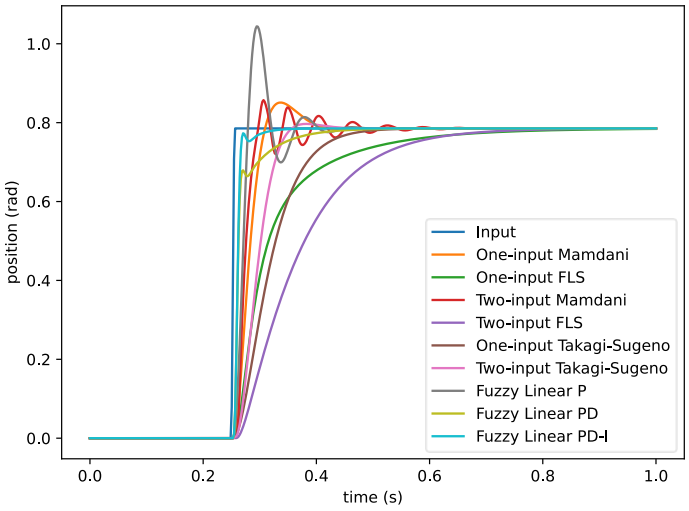

Figure 52. Comparison of control responses of fuzzy controllers implemented with the UPAFuzzySystems library.

The more essential features of a controlled system are the absolute error, the absolute percentage error, the overshoot, the settling time, and the time rising. These features identify the response speed of the controller, its stability, and how well it follows the reference. Maintaining them near zero improves the controller response. As Figure 53 shows, the fuzzy linear PD-I controller has the best results controlling the position of a DC motor.

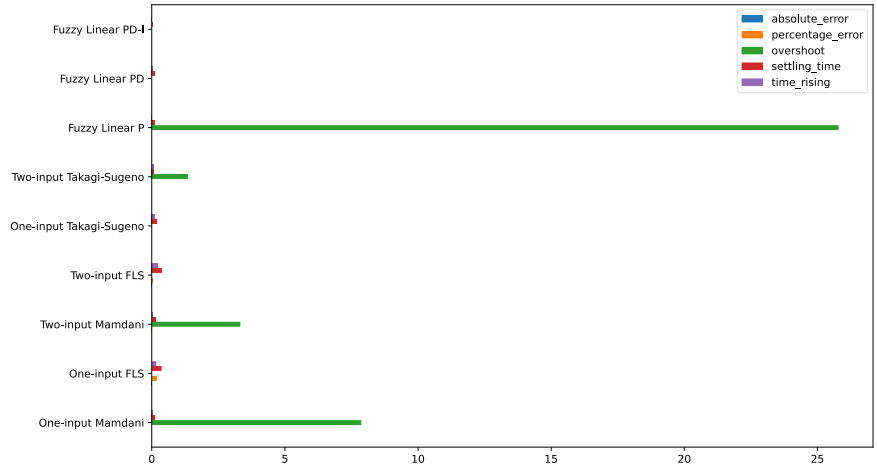

Figure 53. Comparison of the characteristics of interest in fuzzy controllers with the UPAFuzzySystems library.

However, for controlling the position of the DC motor, all the controllers maintain the reference error minimally, and the settling time is also similar between them. At the same time, the more significant dispersions are in the overshoot and the time to achieve it, as the boxplot in Figure 54 shows.

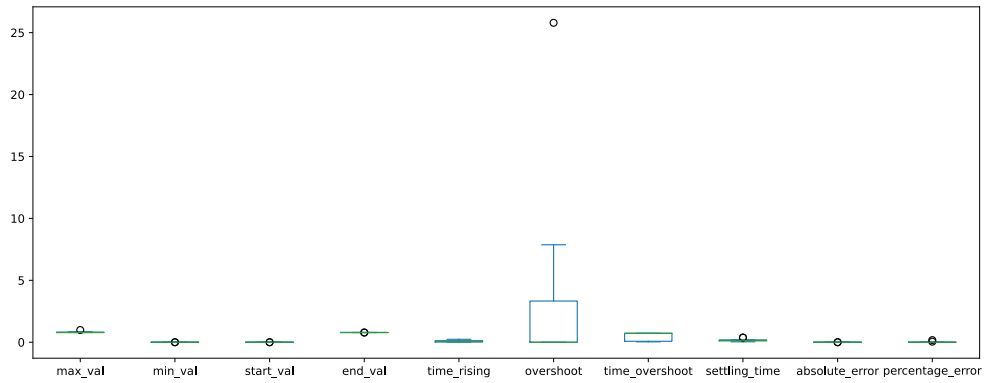

Figure 54. Boxplot of characteristics in fuzzy controllers implemented with the UPAFuzzySystems library.

We also performed an ANOVA test to verify our conclusions about the best controllers and their features of interest without disturbances. We evaluated the test without overshooting features because not all the controllers have overshot. Using the features absolute error, percentage error, settling time, and time rising as groups, we obtained the results in Table 6, which support that the fuzzy linear PD-I controller has the best results among all the fuzzy controllers because of the $p_{value} = 0.000172$.

Moreover, we also repeat the comparison between controllers analyzing their responses with disturbances in Table 7.

Additionally, we compare graphically the control responses of all fuzzy controllers implemented following an input or reference with an angular position of 45 degrees (7.85×10^{-1} rad) and 10% of the input reference uniform random disturbances (Figure 55). As the figure shows, there are variations in the overshoot, time rising, settling time, and time peak. However, due to the disturbances, all the controllers now increase the error to 3.66% of the reference in steady conditions.

Table 5. Comparison of control characteristics in fuzzy controllers implemented with the UPAFuzzySystems library.

Fuzzy Controller	One-Input Mamdani	One-Input FLS	Two-Input Mamdani	Two-Input FLS	One-Input Takagi–Sugeno	Two-input Takagi–Sugeno	Fuzzy Linear P	Fuzzy Linear PD	Fuzzy Linear PD-I
max_val (rad)	8.47×10^{-1}	7.93×10^{-1}	8.25×10^{-1}	7.96×10^{-1}	8.00×10^{-1}	8.02×10^{-1}	9.993×10^{-1}	7.91×10^{-1}	7.91×10^{-1}
min_val (rad)	0.00	-2.39×10^{-16}	7.99×10^{-18}	-5.79×10^{-1}	0.00	0.00	0.00	0.00	0.00
start_val (rad)	0.00	-8.40×10^{-17}	-4.28×10^{-18}	0.00	0.00	0.00	0.00	0.00	0.00
end_val (rad)	7.97×10^{-1}	7.93×10^{-1}	7.57×10^{-1}	7.96×10^{-1}	7.99×10^{-1}	7.96×10^{-1}	7.89×10^{-1}	7.89×10^{-1}	7.87×10^{-1}
time_rising (s)	3.86×10^{-2}	1.99×10^{-1}	2.66×10^{-2}	2.67×10^{-1}	1.33×10^{-1}	6.66×10^{-2}	1.45×10^{-1}	3.86×10^{-2}	1.05×10^{-2}
Overshoot (s)	7.86	0.00	9.09	0.00	1.60×10^{-1}	7.07×10^{-1}	2.58×10^{-1}	2.60×10^{-1}	4.11×10^{-1}
time_overshoot (s)	7.26×10^{-2}	7.22×10^{-1}	6.62×10^{-1}	7.22×10^{-1}	5.28×10^{-1}	5.22×10^{-1}	3.46×10^{-2}	5.18×10^{-1}	5.12×10^{-1}
settling_time (s)	6.38×10^{-1}	6.80×10^{-1}	7.14×10^{-1}	6.74×10^{-1}	6.54×10^{-1}	6.34×10^{-1}	1.65×10^{-1}	1.16×10^{-1}	3.46×10^{-2}
Absolute Error (rad)	1.20×10^{-2}	7.18×10^{-3}	2.88×10^{-2}	11.1×10^{-2}	1.35×10^{-1}	1.07×10^{-2}	3.60×10^{-3}	3.47×10^{-3}	2.05×10^{-3}
Percentage Error (%)	1.53	9.14×10^{-1}	3.66	1.41	1.72	1.37	4.58×10^{-1}	4.42×10^{-1}	2.61×10^{-1}

Table 6. ANOVA analysis of fuzzy controllers without overshoot.

	sum_sq	df	F	PR(>F)
C(features)	0.164450	3.0	9.066149	0.000172
Residual	0.193481	32.0		

Table 7. Comparison of control characteristics in fuzzy controllers with disturbances.

Fuzzy Controller	One-Input Mamdani	One-Input FLS	Two-Input Mamdani	Two-Input FLS	One-Input Takagi–Sugeno	Two-Input Takagi–Sugeno	Fuzzy Linear P	Fuzzy Linear PD	Fuzzy Linear PD-I
max_val (rad)	8.60×10^{-1}	7.93×10^{-1}	8.25×10^{-1}	7.96×10^{-1}	8.00×10^{-1}	$8.02E-01$	9.93×10^{-1}	7.91×10^{-1}	7.91×10^{-1}
min_val (rad)	0.00	-2.39×10^{-16}	-7.99×10^{-18}	-5.79×10^{-5}	0.00	0.00	0.00	0.00	0.00
start_val (rad)	0.00	-8.40×10^{-17}	-4.28×10^{-18}	0.00	0.00	0.00	0.00	0.00	0.00
end_val (rad)	7.97×10^{-1}	7.93×10^{-1}	7.57×10^{-1}	7.96×10^{-1}	7.99×10^{-1}	7.96×10^{-1}	7.89×10^{-1}	7.89×10^{-1}	7.87×10^{-1}
time_rising (s)	3.86×10^{-2}	1.99×10^{-1}	2.66×10^{-2}	2.67×10^{-1}	1.33×10^{-1}	6.66×10^{-2}	1.45×10^{-2}	3.86×10^{-2}	1.05×10^{-2}
Overshoot (s)	7.86	0.00	9.09	0.00	1.60×10^{-1}	7.07×10^{-1}	$2.58 \times 10^{+1}$	2.60×10^{-1}	4.11×10^{-1}
time_overshoot (s)	7.26×10^{-2}	7.22×10^{-1}	6.62×10^{-1}	7.22×10^{-1}	5.28×10^{-1}	5.22×10^{-1}	3.46×10^{-2}	5.18×10^{-1}	5.12×10^{-1}
settling_time (s)	6.38×10^{-1}	6.8×10^{-1}	7.14×10^{-1}	6.74×10^{-1}	6.54×10^{-1}	6.34×10^{-1}	1.65×10^{-1}	1.61×10^{-1}	3.46×10^{-2}
Absolute Error (rad)	1.20×10^{-2}	7.18×10^{-3}	2.88×10^{-2}	1.11×10^{-2}	1.35×10^{-2}	1.07×10^{-2}	3.60×10^{-3}	3.47×10^{-3}	2.05×10^{-3}
Percentage Error (%)	1.53	9.14×10^{-1}	3.66	1.41	1.72	1.37	4.58×10^{-1}	4.42×10^{-1}	2.61×10^{-1}

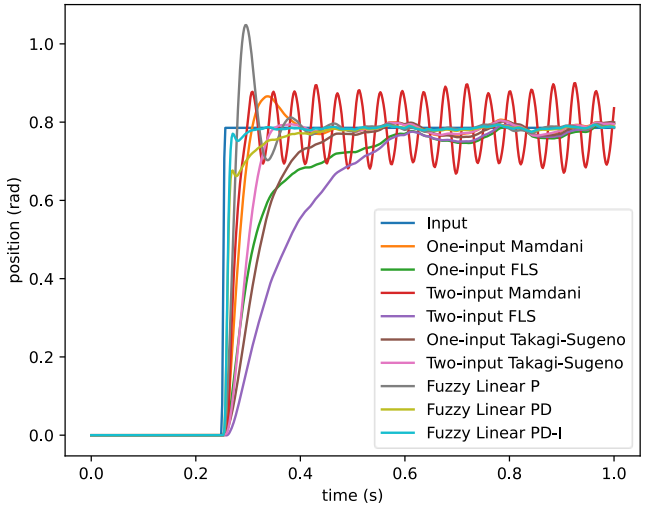

Figure 55. Comparison of control responses of fuzzy controllers with disturbances.

Again, we also compare the more essential features of a controlled system with the fuzzy controllers and signal disturbances. As Figure 56 shows, the fuzzy linear PD-I controller has the best results controlling a DC motor's position and is the most robust.

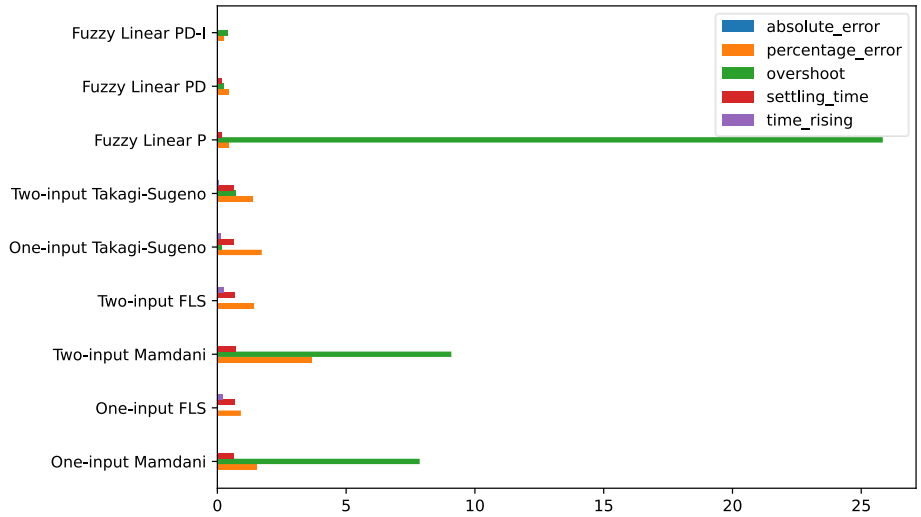

Figure 56. Comparative of characteristics of interest in fuzzy controllers with disturbances.

However, for controlling the position of the DC motor, all the controllers maintain a reference error below 4% of error, and the settling time is also similar between them. At the same time, the more significant dispersions are in the overshoot and the time to achieve it, as the boxplot in Figure 57 shows.

We also performed an ANOVA test to verify our conclusions about the best controllers and their features of interest with disturbances. Again, we evaluated the test without overshooting features because not all the controllers have overshot. Using the features absolute error, percentage error, settling time, and time rising as groups, we obtained the results in Table 8, which support that the fuzzy linear PD-I controller has the best results among all the fuzzy controllers in the presence of disturbances because of the $p_{value} = 0.000038$.

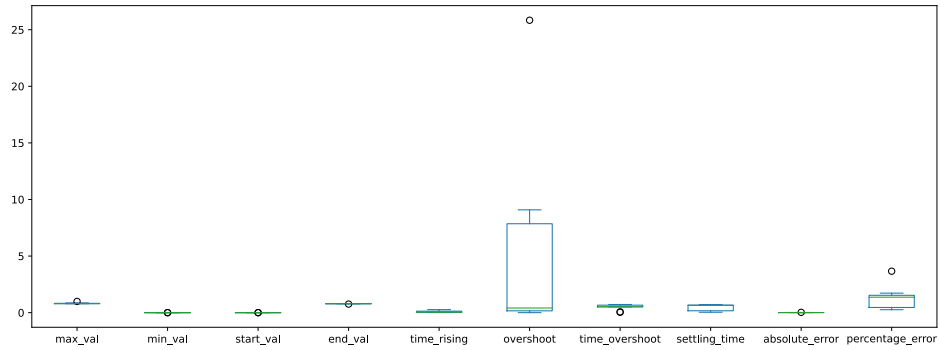

Figure 57. Boxplot of characteristics in fuzzy controllers with disturbances.

Table 8. ANOVA analysis of fuzzy controllers without overshoot.

	Sum_sq	**df**	**F**	**PR (>F)**
C(features)	9.535859	3.0	11.074266	0.000038
Residual	9.184882	32.0		

4. Conclusions

This work proposes the UPAFuzzySystems library for designing inference systems with fuzzy logic, simulation, and control with fuzzy controllers, transfer functions, and state-space models in discrete and continuous universes. To our knowledge, this proposal is the only open-source Python library that integrates these functions.

Section 3.2 showed that the UPAFuzzySystems library could define fuzzy universes for different situations, such as collision distance and recommended speed. Section 3.3 shows how to specify the rules in an FIS system linking premises, connectives, and consequences. Moreover, the library successfully simulates the problem's surface response with input arrays defining the inputs for the premise and obtaining its corresponding consequence in a continuous universe.

Additionally, the UPAFuzzySystems library also successfully controls and simulates the position of a DC motor plant (transfer function defined in Table 4 and codified in Code 4) with the fuzzy controllers: one-input Mamdani controller in Section 3.4.1, one-input FLS controller in Section 3.4.2, Takagi–Sugeno controller in Section 3.4.3, two-input Mamdani controller in Section 3.4.4, two-input FLS controller in Section 3.4.5, two-input Takagi–Sugeno controller in Section 3.4.6. Similarly, the UPAFuzzySystems library allows the implementation of one-input and two-input linear fuzzy systems, together with the P (Section 3.5.1), PD (Section 3.5.2), and PD-I (Section 3.5.3) controllers.

The controllers implemented using the proposed library reduce the steady error below 1% without disturbances and below 4% in the presence of 10% uniform random disturbances. Furthermore, we obtained interest features in control systems, such as overshoot, steady time, time rising, and time peak, that vary depending on the controller. Those variations occur because of the different changes in the control structures, such as the premises, the consequences, the connectives, the implication, the fuzzification, and the defuzzification methods. Moreover, some of these structures even include derivatives that allow predicting changes in the error behavior or integrals that allow gradual error reduction.

After performing an ANOVA analysis with the values of the features and the error with each controller, supported with a $p_{value} = 0.000038$, we concluded that the PD-I controller obtains the best error reduction and features of interest. These best features include a faster response, no overshoot, and no oscillations after reaching the reference, even in the presence of disturbances, as detailed in Section 3.6.

Our proposal successfully implements different fuzzy structures for designing FISs systems in general problems or controlling the position of a DC motor. However, following the codes we use in this work, UPAFuzzySystems can help researchers and designers solve

problems in general applications with FISs or even use fuzzy controllers in the literature to control and simulate other transfer functions or state-space models.

Future Work

Although our proposal offers several control structures mechanisms for designing and simulating FISs and fuzzy controllers, the users would also benefit if there were methods for simplifying the implementation process. Thus, we will develop mechanisms for automatically configuring the implementation of the controllers in embedded systems based on the Raspberry Pi and Arduino platforms.

Author Contributions: Conceptualization M.M.R. and E.O.-G.; methodology; software M.M.R.; validation, all authors; formal analysis, M.M.R. and E.O.-G.; investigation, all authors; resources, all authors; data curation M.M.R. and E.O.-G.; writing—original draft preparation, M.M.R. and N.E.-G.; writing—review and editing, all authors; visualization, E.O.-G. and N.E.-G.; supervision, M.M.R.; project administration, M.M.R. and N.E.-G.; funding acquisition, E.O.-G. and N.E.-G. All authors have read and agreed to the published version of the manuscript.

Funding: We acknowledge the support of Instituto Tecnológico de Pabellón and the Consejo Nacional de Ciencia y Tecnología (CONA-CYT) in Mexico for supporting this work through funds for projects INFRA-2016-01, Project No. 270665. CB-2016-01, Project No. 287818.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Astrom, K.J.; Wittenmark, B. *Adaptive Control—Astrom*, 2nd ed.; Courier Corporation: North Chelmsford, MA, USA, 2008; pp. 17–38.
2. Spirin, N.A.; Rybolovlev, V.Y.; Lavrov, V.V.; Gurin, I.A.; Schnayder, D.A.; Krasnobaev, A.V. Scientific Problems in Creating Intelligent Control Systems for Technological Processes in Pyrometallurgy Based on Industry 4.0 Concept. *Metallurgist* **2020**, *64*, 574–580. [CrossRef]
3. Åström, K.J.; Wittenmark, B. *Computer-Controlled Systems: Theory and Design*; Courier Corporation: North Chelmsford, MA, USA, 2011; p. 557.
4. Tarbosh, Q.A.; Aydogdu, O.; Farah, N.; Talib, M.H.N.; Salh, A.; Cankaya, N.; Omar, F.A.; Durdu, A. Review and Investigation of Simplified Rules Fuzzy Logic Speed Controller of High Performance Induction Motor Drives. *IEEE Access* **2020**, *8*, 49377–49394. [CrossRef]
5. Jang, J.S.R.; Sun, C.T.; Mizutani, E. Neuro-Fuzzy and Soft Computing-A Computational Approach to Learning and Machine Intelligence [Book Review]. *IEEE Trans. Autom. Control.* **1997**, *42*, 482–1484. [CrossRef]
6. Ferdaus, M.M.; Pratama, M.; Anavatti, S.G.; Garratt, M.A.; Lughofer, E. PAC: A Novel Self-Adaptive Neuro-Fuzzy Controller for Micro Aerial Vehicles. *Inf. Sci.* **2020**, *512*, 481–505. [CrossRef]
7. Chen, Y.; Zhao, T.; Peng, J.; Mao, Y. Fuzzy Fraction-Order Stochastic Parallel Gradient Descent Approach for Efficient Fiber Coupling. *Opt. Eng.* **2022**, *61*, 016108. [CrossRef]
8. Khanesar, M.A.; Branson, D. Robust Sliding Mode Fuzzy Control of Industrial Robots Using an Extended Kalman Filter Inverse Kinematic Solver. *Energies* **2022**, *15*, 1876. [CrossRef]
9. Pereira, L.F.d.S.C.; Batista, E.; de Brito, M.A.G.; Godoy, R.B. A Robustness Analysis of a Fuzzy Fractional Order PID Controller Based on Genetic Algorithm for a DC-DC Boost Converter. *Electronics* **2022**, *11*, 1894. [CrossRef]
10. Pozna, C.; Precup, R.E.; Horvath, E.; Petriu, E.M. Hybrid Particle Filter-Particle Swarm Optimization Algorithm and Application to Fuzzy Controlled Servo Systems. *IEEE Trans. Fuzzy Syst.* **2022**, *30*, 4286–4297. [CrossRef]
11. Volosencu, C. MATLAB Applications in Engineering. Available online: https://www.researchgate.net/publication/358537759_MATLAB_Applications_in_Engineering (accessed on 16 February 2023).
12. Zahmatkesh, S.; Klemeš, J.J.; Bokhari, A.; Rezakhani, Y.; Wang, C.; Sillanpaa, M.; Amesho, K.T.T.; Ahmed, W.S. Reducing Chemical Oxygen Demand from Low Strength Wastewater: A Novel Application of Fuzzy Logic Based Simulation in MATLAB. *Comput. Chem. Eng.* **2022**, *166*, 107944. [CrossRef]
13. Maghfiroh, H.; Ahmad, M.; Ramelan, A.; Adriyanto, F. Fuzzy-PID in BLDC Motor Speed Control Using MATLAB/Simulink. *J. Robot. Control* **2022**, *3*, 8–13. [CrossRef]
14. Mostafa, S.; Zekry, A.; Youssef, A.; Anis, W.R. Raspberry Pi Design and Hardware Implementation of Fuzzy-PI Controller for Three-Phase Grid-Connected Inverter. *Energies* **2022**, *15*, 843. [CrossRef]
15. Lin, X.; Liu, X. Modeling and Control of One-Stage Inverted Pendulum Body Based on Matlab. *J. Phys. Conf. Ser.* **2022**, *2224*, 012107. [CrossRef]

16. Kandemir, E.; Cetin, N.S.; Borekci, S. Single-Stage Photovoltaic System Design Based on Energy Recovery and Fuzzy Logic Control for Partial Shading Condition. *Int. J. Circuit Theory Appl.* **2022**, *50*, 1770–1792. [CrossRef]
17. Taskin, A.; Kumbasar, T. An Open Source Matlab/Simulink Toolbox for Interval Type-2 Fuzzy Logic Systems. In Proceedings of the 2015 IEEE Symposium Series on Computational Intelligence (SSCI), Cape Town, South Africa, 7–10 December 2015; pp. 1561–1568. [CrossRef]
18. Wang, J.; Niu, X.; Zheng, L.; Zheng, C.; Wang, Y. Wireless Mid-Infrared Spectroscopy Sensor Network for Automatic Carbon Dioxide Fertilization in a Greenhouse Environment. *Sensors* **2016**, *16*, 1941. [CrossRef] [PubMed]
19. Wang, C.C. Fuzzy Theory-Based Air Valve Control for Auto-Score-Recognition Soprano Recorder Machines. *J. Robot. Netw. Artif. Life* **2021**, *8*, 278–283. [CrossRef]
20. Singh, S.; Kaur, M. Gain Scheduling of PID Controller Based on Fuzzy Systems. *MATEC Web Conf.* **2016**, *57*, 01008. [CrossRef]
21. Mehmet Karadeniz, A.; Ammar, A.; Geza, H. Comparison between Proportional, Integral, Derivative Controller and Fuzzy Logic Approaches on Controlling Quarter Car Suspension System. *MATEC Web Conf.* **2018**, *184*, 02018. [CrossRef]
22. TIOBE Index—TIOBE. Available online: https://www.tiobe.com/tiobe-index/ (accessed on 16 February 2023).
23. Xu, B.; An, L.; Thung, F.; Khomh, F.; Lo, D. Why Reinventing the Wheels? An Empirical Study on Library Reuse and Re-Implementation. *Empir. Softw. Eng.* **2020**, *25*, 755–789. [CrossRef]
24. Rueden, C.T.; Schindelin, J.; Hiner, M.C.; DeZonia, B.E.; Walter, A.E.; Arena, E.T.; Eliceiri, K.W. ImageJ2: ImageJ for the next Generation of Scientific Image Data. *BMC Bioinform.* **2017**, *18*, 529. [CrossRef] [PubMed]
25. Macmillan, M.; Eurek, K.; Cole, W.; Bazilian, M.D. Solving a Large Energy System Optimization Model Using an Open-Source Solver. *Energy Strategy Rev.* **2021**, *38*, 100755. [CrossRef]
26. Guo, Y.; Leitner, P. Studying the Impact of CI on Pull Request Delivery Time in Open Source Projects—A Conceptual Replication. *PeerJ Comput. Sci.* **2019**, *5*, e245. [CrossRef] [PubMed]
27. Shields.Io: Quality Metadata Badges for Open Source Projects. Available online: https://shields.io/ (accessed on 24 January 2023).
28. GitHub—Luferov/FuzzyLogicToolBox: Fuzzy Logic Library for Python. Available online: https://github.com/Luferov/FuzzyLogicToolBox (accessed on 23 January 2023).
29. Avelar, E.; Castillo, O.; Soria, J. Fuzzy Logic Controller with Fuzzylab Python Library and the Robot Operating System for Autonomous Robot Navigation: A Practical Approach. *Stud. Comput. Intell.* **2020**, *862*, 355–369. [CrossRef]
30. ITTcs/Fuzzylab: Fuzzylab, a Python Fuzzy Logic Library. Available online: https://github.com/ITTcs/fuzzylab (accessed on 23 January 2023).
31. GitHub—Yudivian/Fuzzython: Fuzzy Logic and Fuzzy Inference Python 3 Library. Available online: https://github.com/yudivian/fuzzython (accessed on 24 January 2023).
32. GitHub—Carmelgafa/Type2fuzzy: Type-2 Fuzzy Logic Library. Available online: https://github.com/carmelgafa/type2fuzzy (accessed on 24 January 2023).
33. Fuzzylite/Pyfuzzylite: Pyfuzzylite: A Fuzzy Logic Control Library in Python. Available online: https://github.com/fuzzylite/pyfuzzylite (accessed on 24 January 2023).
34. Montes Rivera, M. GitHub—UniversidadPolitecnicaAguascalientes/UPAFuzzySystems. Available online: https://github.com/UniversidadPolitecnicaAguascalientes/UPAFuzzySystems (accessed on 1 March 2023).
35. Nguyen, H.T. *A First Course in Fuzzy and Neural Control*; Chapman & Hall/CRC Press: London, UK, 2003; ISBN 9781584882442.
36. Jantzen, J. *Foundations of Fuzzy Control: A Practical Approach*, 2nd ed.; John Wiley & Sons: Hoboken, NJ, USA, 2013; pp. 1–325. [CrossRef]
37. Dehghani, M.; Taghipour, M.; Gharehpetian, G.B.; Abedi, M. Optimized Fuzzy Controller for MPPT of Grid-Connected PV Systems in Rapidly Changing Atmospheric Conditions. *J. Mod. Power Syst. Clean Energy* **2021**, *9*, 376–383. [CrossRef]
38. Sao, K.; Kumar Singh, D.; Agrawal, A.; Scholar, P.; Professor, A.; Raman, D. Study of DC Motor Position Control Using Root Locus and PID Controller in MATLAB. *IJSRD-Int. J. Sci. Res. Dev.* **2015**, *3*, 183–190.