http://doi.org/10.35784/iapgos.8023 *received: 02.11.2025 | revised: 13.11.2025 | accepted: 18.11.2025 | available online: 20.12.2025*

DEVELOPMENT AND VERIFICATION OF A MODULAR OBJECT-ORIENTED FUZZY LOGIC CONTROLLER ARCHITECTURE FOR CUSTOMIZABLE AND EMBEDDED APPLICATIONS

Rahim Mammadzada
Azerbaijan State Oil and Industry University, Department of Instrumentation Engineering, Baku, Azerbaijan

Abstract. *This paper presents an open-source and lightweight, object-oriented fuzzy logic controller architecture designed to overcome key limitations of popular platforms like MATLAB and LabVIEW. These traditional tools, while widely used, are closed-source and license-dependent, creating barriers to portability, system-level integration, and long-term flexibility. This modular framework breaks the controller into reusable components implemented as separate classes, all coordinated by a central controller class that manages the full inference cycle from input fuzzification to output defuzzification. This central controller can be easily customized for different control scenarios, as demonstrated here through validation on both single-input and multi-input cases. The results confirm that the architecture delivers reliable and consistent performance while scaling smoothly to handle increased input complexity without redesigning core components. This approach offers a transparent, maintainable, and flexible alternative that empowers developers with full control over their fuzzy logic implementations and supports integration in a variety of software and embedded environments.*

Keywords: C# languages, object oriented programming, fuzzy logic, open-source software

OPRACOWANIE I WERYFIKACJA MODUŁOWEJ, OBIEKTOWO ZORIENTOWANEJ ARCHITEKTURY KONTROLERA LOGIKI ROZMYTEJ DLA KONFIGUROWALNYCH I WBUDOWANYCH APLIKACJI

Streszczenie. *W artykule przedstawiono lekką, zorientowaną obiektowo architekturę kontrolera logiki rozmytej o otwartym kodzie źródłowym, zaprojektowaną w celu przezwyciężenia kluczowych ograniczeń popularnych platform, takich jak MATLAB i LabVIEW. Te tradycyjne narzędzia, choć szeroko stosowane, są zamknięte i zależne od licencji, tworząc bariery dla przenośności, integracji na poziomie systemu i długoterminowej elastyczności. Ta modułowa struktura dzieli kontroler na komponenty wielokrotnego użytku zaimplementowane jako oddzielne klasy, wszystkie koordynowane przez centralną klasę kontrolera, która zarządza pełnym cyklem wnioskowania od rozmywania danych wejściowych do rozmywania danych wyjściowych. Ten centralny kontroler można łatwo dostosować do różnych scenariuszy sterowania, co zademonstrowano tutaj poprzez walidację zarówno w przypadku pojedynczego wejścia, jak i wielu wejść. Wyniki potwierdzają, że architektura ta zapewnia niezawodną i spójną wydajność, jednocześnie płynnie skalując się w celu obsługi zwiększonej złożoności danych wejściowych bez konieczności przeprojektowywania podstawowych komponentów. Takie podejście oferuje przejrzystą, łatwą w utrzymaniu i elastyczną alternatywę, która zapewnia programistom pełną kontrolę nad implementacjami logiki rozmytej i wspiera integrację z różnymi środowiskami programistycznymi i wbudowanymi.*

Słowa kluczowe: języki C#, programowanie obiektowe, logika rozmyta, oprogramowanie typu open source

Introduction

When considering a platform for developing fuzzy logic-based control techniques, the first thing that comes to mind is MATLAB's fuzzy logic addon. However, some limitations that come with the usage of licence-based programs are overlooked. These limitations can affect each user differently. Some may be students who can't afford the full license or do not have a computer strong enough to run the software, or in some cases, the closed-source nature of the software may limit the overall capabilities of the engineers and programmers who rely on this software. It is also worth noting that the price, policy, and overall safety of the commercial software can also change due to changes of ownership, rebranding, or hacker attacks, which are all outside the user's control. Thus, this paper explores an alternative way a fuzzy controller can be made and verified without relying on commercial software.

Introduced by Lotfi Zadeh in the 1960s, fuzzy logic has enabled the development of controllers capable of handling imprecision and uncertainty in a structured, rule-based manner [34]. Fuzzy Logic Controller (FLC) have since found wide application in control systems that operate under incomplete information or nonlinear behaviour [31]. Their strength lies in their ability to model human-like reasoning, making them effective where traditional control strategies may struggle [2]. As a result, FLC is commonly deployed in areas such as temperature regulation in HVAC applications [11, 16], power electronics [3, 36], industrial applications [22], and many more [24, 30].

In most cases, fuzzy logic controllers are developed using graphical environments such as MATLAB or LabVIEW [4, 10, 28]. These platforms offer fast prototyping, simulation tools, and built-in fuzzy logic libraries that simplify controller design. However, such convenience often comes at a cost. Limitations arise regarding portability, licensing, integration with broader software systems, and the ability to fully customize the control architecture [9]. For instance, exporting controllers to embedded systems or integrating them with desktop applications may require complex workarounds or additional toolboxes [20].

These platform-level constraints can lead to missed opportunities. Developers may be discouraged from tailoring the controller architecture to their specific system, and access to advanced software design patterns such as modularity or reusability is often limited. Additionally, reliance on proprietary environments reduces transparency, reproducibility, and long-term maintainability, all of which are increasingly important in modern engineering workflows [6, 9, 18].

Existing alternatives to traditional environments may grant better alignment with long-term software and integration goals. One such alternative is the implementation of fuzzy logic controllers using open-source solutions or the development of a fully custom implementation [15, 26, 29, 31, 35]. This can be effectively realized through object-oriented programming languages as well [10], which are inherently well-suited for fuzzy logic due to their modular structure. Such an approach grants full code ownership promotes transparency, and eliminates reliance on commercial platforms. Open-source development further enables the creation of flexible, portable, and scalable fuzzy control systems, where core components are organized into self-contained classes. This structure simplifies extension, accelerates debugging, and supports seamless integration into larger applications and control frameworks.

Among various object-oriented programming (OOP) languages, C# presents a compelling option. It was developed by Microsoft in the early 2000s as part of the .NET initiative, combining the expressiveness of modern programming languages with the rigor of structured design. What makes it especially appealing for control-oriented applications is not just its object-oriented architecture, but the way it encourages clean, maintainable, and highly modular code. This makes it ideal

for fuzzy logic controller development and verification [13], where a clear separation between membership functions, rule structures, and output mechanisms is essential. Furthermore, C# development environments such as Visual Studio provide mature debugging tools and intuitive code management, making it accessible for both academic and hobbyist developers. Unlike many domain-specific tools, C# offers the versatility to go beyond prototypes, opening the door for seamless integration into larger industrial, desktop, game-related, or web-based systems [23].

This paper presents an implementation of a fuzzy logic controller built entirely in C# using OOP techniques. The architecture includes separate classes for membership functions, a rule base, defuzzification modules, and a reusable main controller logic that combines all of these components. The design emphasizes clarity, modularity, and code reuse. All components have been released in an open-source repository to encourage community feedback, transparency, and further development. This work demonstrates that C# can be used not just as a programming language but as a foundation for robust fuzzy control systems outside conventional platforms.

1. Literature review

Fuzzy logic controllers have been widely adopted in place of or in combination with conventional proportional-integral-derivative (PID) controllers, particularly in thermal and environmental applications. Notable areas include HVAC systems [11, 16], battery temperature regulation [36], intelligent building climate control [1, 5], and biomass-based heating systems [27]. These implementations leverage fuzzy logic's strength in handling nonlinearities, uncertainties, and time-varying conditions, often where classical PID control alone may not provide sufficient adaptability or accuracy.

Fuzzy controllers are also used in other areas. Some examples include robot motion control [14, 23], and even oil well automation [22]. In these systems, fuzzy logic helps manage uncertain data or changes in system behavior. Dumitrescu et al. [12] used it to handle sensor errors in emergencies. Brucal et al. [5] used it to save energy by adjusting air conditioning based on occupancy. These show that fuzzy logic can adapt well to real conditions.

Most of these studies use software like MATLAB, Simulink, or LabVIEW [6, 9, 10, 28]. These tools are easy to use and powerful. But they are not free or open to modification. This makes them hard to use in custom or low-cost applications. They are also not ideal for running real-time systems on embedded hardware.

To solve this, several researchers have pursued open-source implementations, most notably in Python. Libraries like PyIT2FLS [15], Simpful [29], and Scikit-ANFIS [35] allow for modular, customizable, and interpretable fuzzy logic development. Tang and Ahmad [31] offered a comprehensive review of such tools, emphasizing their role in modelling uncertain and nonlinear systems. Saatchi [26] contributed a theoretical survey of fuzzy logic concepts, and Spolaor et al. [29] provided a Python toolkit to lower the barrier to entry for fuzzy control. These developments signal a shift toward more transparent and reproducible research infrastructures.

However, an underexplored area is the implementation of fuzzy logic controllers in C#, a language well-suited to industrial environments due to its strong support for object-oriented programming, real-time hardware interfacing, and .NET integration. While Zhang W. et al. [37] discussed OOP-related abstraction issues in open-source projects, no recent works provide reusable, structured fuzzy logic implementations in C#. Ferrara et al. [13] explained why it is important to write code that is easy to test and verify. C# excels in this domain, but it is rarely used in fuzzy logic research.

The current work aims to bridge this gap by introducing a modular framework for building fuzzy logic controllers using the C# programming language. The framework is designed to be open, portable, and efficient, with a focus on clean code, reuse, and real-time use. It lets users build fuzzy systems from scratch by defining fuzzy sets, rules, and logic without relying on closed or paid software. This makes it useful for learning, quick testing, and use in embedded systems. By choosing C# instead of tools like MATLAB, the project shows that a working fuzzy logic controller can be created using free and flexible tools. While it is still early in development and not yet a full toolbox, it shows strong potential for future growth and use in more complex systems.

2. Theoretical framework

Fuzzy logic control is widely used to manage systems that involve uncertainty, nonlinearity, or incomplete information. Originally introduced by Lotfi Zadeh, the core idea is to model how humans make decisions using approximate values and linguistic terms rather than exact equations [34]. A typical fuzzy controller is shown in Fig. 1, which has three main steps: it begins by converting numerical inputs into degrees of membership across fuzzy sets such as low, medium, or high. These values are then evaluated using a rule base composed of intuitive if-then statements that capture logical relationships. Finally, the fuzzy output is converted into a single crisp value that can drive a control action [26, 31].

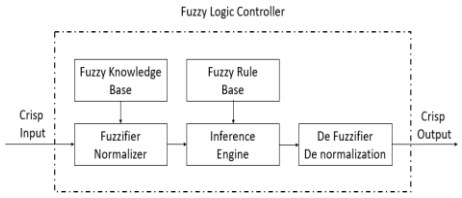

Fig. 1. Fuzzy logic controller architecture

This structure works well in cases where building precise models is difficult or where system behaviour does not follow strict patterns. Although platforms like MATLAB offer graphical tools for fuzzy design, implementing the same logic in a general-purpose programming language can offer more control and portability. This is where object-oriented programming becomes a strong fit. By dividing the system into clearly defined parts the code becomes easier to maintain, adapt, and reuse. The following sections describe how this approach was applied in the present work using C Sharp.

2.1. Object-oriented fuzzy logic architecture

To provide a clear foundation for implementation, this subsection focuses on the theoretical structure behind the proposed controller. The architecture is organized into five key classes: two dedicated to managing membership functions (one for individual fuzzy sets and one for collections), two for the rule base (handling single rules and the full rule set), and one core class for defuzzification Fig. 2.

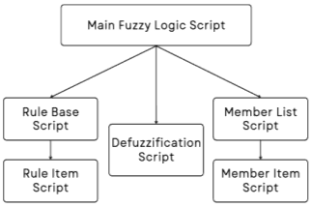

Fig. 2. Proposed modular object-oriented fuzzy logic controller architecture

Unlike traditional fuzzy logic implementations, where these elements are often combined in a single block or GUI-based tool, this approach deliberately separates them to improve traceability and maintainability at the code level. The main fuzzy logic class acts as the controller brain, bringing together these modules and orchestrating the inference cycle. It behaves similarly to what MATLAB's Fuzzy Logic Toolbox offers, but in this case, all source code is available and editable. This not only makes

the logic transparent but also gives developers and engineers full control to modify, expand, or embed the system into larger applications. What follows is a breakdown of each part of this architecture, focusing on their internal roles and the design motivations behind their object-oriented structure.

2.2. Definition of membership functions

Membership functions are one of the key parts of a fuzzy logic system [34]. They handle the task of translating crisp input values into fuzzy degrees of belonging, a process called fuzzification. This allows the controller to work with terms like Low, Medium, or High instead of relying on precise values alone. Two of the most common shapes used for this are triangles and trapezoids. Triangular membership functions are popular because of their simplicity and fast computation. They have a single peak in the middle and straight slopes on both sides Fig. 3. Trapezoidal shapes build on this by adding a flat region at the top, which helps model wider zones of full membership and more gradual transitions Fig. 4 [26, 31].

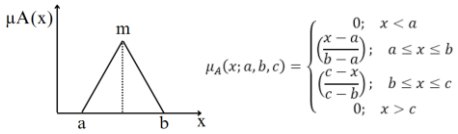

Fig. 3. Triangular membership function with corresponding conditions

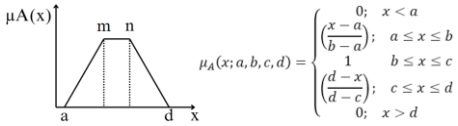

Fig. 4. Trapezoidal membership function with corresponding conditions

In the proposed architecture, the goal is to support both of these shapes using a similar code structure. This can be done by storing key points of the function in an integer array. The first and last values set the range of the base, while the in-between values define the slopes and peaks. With this setup, the same logic can be used to create a triangle and a trapezoid shape, which serve as the basis for how the controller interprets its input.

2.3. Definition of rule base

The rule base is the reasoning core of a fuzzy logic controller. It connects different input conditions to control actions using simple if-then logic expressed in linguistic terms [34]. Each fuzzy rule defines a relationship such as: if condition one is High and condition two is Medium, then the output is Low. By combining multiple rules, the system forms a decision-making framework that can handle uncertainty and overlapping values in a way that mirrors human reasoning [26, 31].

To implement this structure in code, each rule is represented as an object that stores its input conditions and the resulting output action. This allows individual rules to be modified dynamically or generated programmatically using a mapping structure. Similar to how membership functions are grouped, these rule objects are collected into a rule base, which the system can iterate through during evaluation. This modular approach keeps the logic transparent and adaptable, making it easy to expand, debug, or integrate into more complex systems. It also supports different application needs, whether the rules are manually defined or automatically generated.

2.4. Definition of defuzzification

In the final stage of the fuzzy logic process, defuzzification converts the aggregated fuzzy output into a single crisp value that can be used for control actions [34]. In this implementation, the defuzzifier only requires the weighted outputs from the fuzzy rules, making the process streamlined and modular. For simplicity and performance, the system uses a zero-order Sugeno method Fig. 5, assuming that all output membership functions are evenly spread around their midpoints [26, 31].

This assumption simplifies the calculation to a weighted average of the output values, which remains effective in many practical cases. The design allows this component to remain independent from the rest of the controller, while still supporting accurate inference. Additionally, the class structure is flexible enough to support other defuzzification methods in the future. This enables developers to expand the system's capabilities without needing to redesign or rewrite existing modules, maintaining the clarity and adaptability expected from a well-structured object-oriented system.

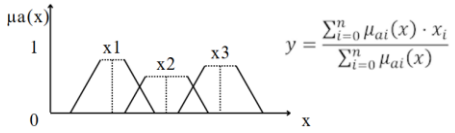

Fig. 5. Sugeno defuzzification method

3. Code base of the proposed controller

Apart from the discussed theoretical framework, this work also provides an in-depth breakdown of the source code of this controller for easier replication of the environment and results of all scenarios presented in this study. The full codebase consists of seven scripts, one for each class. Two of which are used to define membership functions, another two are used to define the rule base and one script for handling defuzzification. The final two scripts are for the fuzzy logic setup that combines all the other scripts and a program script for surface-level interaction to quickly validate the controller with any input values. Both the fuzzy logic script and the program script are written uniquely for each case scenario. Thus, they are shown in the validation section when presenting examples with chosen inputs.

3.1. Member item script

The goal of the "Member Item" script is to create and store the information necessary to represent the fuzzy membership shapes. In Fig. 6, the indexed representation of the commonly used fuzzy shapes is given, where instead of letters, brackets with numbers represent at which array index each point is located inside a numeric array.

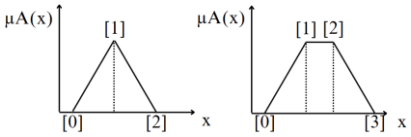

Fig. 6. Indexed representation of triangular and trapezoidal membership functions

In the code representation, these indexed values are stored within a float array called "points", additionally, another float variable named "weight" is used to store the membership degree. To improve traceability and debugging, the name of the input parameter and the linguistic definition of the membership function are also stored as a string.

```
internal float[] points;
internal float weight = 0;
internal string nameMember = "";
internal string nameParameter = "";
```

In order to enforce the correct declaration of this class as a variable, a constructor that shares the same name as its script called "Member Item" is used to demand all of these variables.

```
internal MemberItem(string nameParameter, string
nameMember, float[] points)
{
    this.points = points;
    this.nameMember = nameMember;
    this.nameParameter = nameParameter;
}
```

Inside the same class, a separate function is written for the calculation of the membership degree for a given input called "OnCalculateWeight". However, since float array "points" can be declared with a length of three for triangular and a length of four for trapezoidal membership functions, it is necessary to check the length first before performing the calculation of the weight of correspondence of the input parameter value with the current membership instance.

```
internal void OnCalculateWeight(float value)
{
    if (points.Length == 3)
        OnCalculateForTriangle(value);
    else if (points.Length == 4)
        OnCalculatedForTrapezoid(value);
}
```

In the case when the "points" length is three, the "OnCalculateForTriangle" function is called. Fig. 7 describes its conditions in a more graphical form followed by the code implementation.

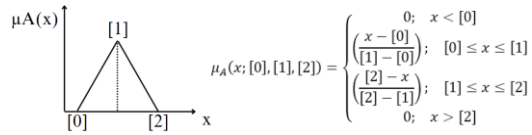

Fig. 7. Indexed triangular membership function with corresponding conditions

```
private void OnCalculateForTriangle(float value)
{
    if (value < points[0])
        weight = 0;
    else if (points[0] <= value && value <= points[1])
        weight = (value-points[0])/(points[1]-points[0]);
    else if (points[1] <= value && value <= points[2])
        weight = (points[2]-value)/(points[2]-points[1]);
    else if (value > points[2])
        weight = 0;
}
```

In the case when the "points" length is four, the "OnCalculateForTriangle" function is called. Fig. 8 describes its conditions in a more graphical form followed by the code implementation.

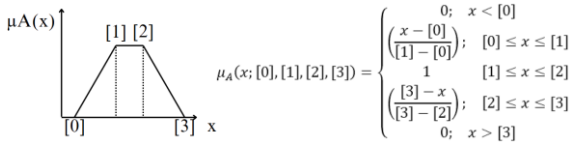

Fig. 8. Indexed trapezoidal membership function with corresponding conditions

```
private void OnCalculatedForTrapezoid(float value)
{
    if (value < points[0])
        weight = 0;
    else if (points[0] <= value && value <= points[1])
        weight = (value-points[0])/(points[1]-points[0]);
    else if (points[1] <= value && value <= points[2])
        weight = 1;
    else if (points[2] <= value && value <= points[3])
        weight = (points[3]-value)/(points[3]-points[2]);
    else if (value > points[3])
        weight = 0;
}
```

Both of these functions contain all of the corresponding conditions to check to see if the given value lies within the support range of the triangular or the trapezoidal membership function. For both of them, if the current value of the target parameter falls outside their range, then the "weight" variable is assigned a value of zero, and on the contrary, if the target parameter value falls into the peak region, then "weight" variable is assigned a value of one. Similarly, for both shapes, there are two slopes that have different calculation equations. These slopes are determined by the position with respect to the peak, if the slope is located on the left, then it is considered the ascending, while the slope on the right is considered the descending slope.

3.2. Member list script

By using the "Member Item" scripts as building blocks, a complete fuzzy set for any input or output variables can be defined as shown in Fig. 9. This is achieved by simply creating an array of "Member Item" inside the "Member List" class.

```
internal MemberItem[] members;
```

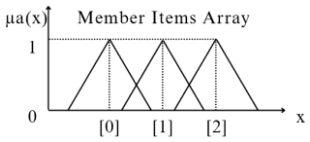

Fig. 9. Membership functions indexed as an array of member items

Just like with the "Member Item", the "Member List" class also uses a constructor to define all of the member items inside it when this class is declared inside the main fuzzy logic script.

```
internal MemberList(MemberItem[] pass) => members = pass;
```

Since all "Member Item" are declared within the "Member List", it is more convenient to assign their weights collectively using a single function that iterates through them Fig. 10.

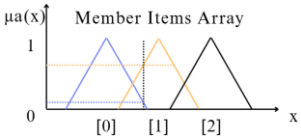

Fig. 10. Assignment of weights of membership items based on the current value

```
internal void OnCalculateWeights(float value)
{
    for (int i = 0; i < members.Length; i++)
    members[i].OnCalculateWeight(value);
}
```

Since the membership weight for the output is determined by the rule base instead of direct input fuzzification, a separate version of the function is required. For convenience, this version reuses the same method name through method overloading in C#. It assigns weights to each output member item based on the activated rules, supporting a consistent and modular implementation of the fuzzy inference process.

```
internal void OnCalculateWeights(RuleItem[,] rules,
int[,] ruleTable)
{
    for (int i = 0; i < ruleTable.GetLength(0); i++)
        for (int j = 0; j < ruleTable.GetLength(1); j++)
            if(rules[i,j].weight>members[ruleTable[i,j]].we
        ight)
            members[ruleTable[i,j]].weight=rules[i,j].weigh
        t;
}
```

This modular design enables flexible evaluation of input values across various scenarios and supports rule verification within the fuzzy inference system later on.

3.3. Rule item script

A class called "Rule Item" is created with the purpose of representing each rule. This class encapsulates two conditions and one consequent, each mapped to fuzzy sets using linguistic variables. It also includes a "weight" variable for storing each rule's weight during inference.

```
internal float weight = 0;
private string input_1 = "";
private string input_2 = "";
private string result = "";
```

Additionally, inside the "Rule Item" script, there is a function for setting the values of these variables at once.

```
internal void OnSet(float weight, string input_1, string
input_2, string result)
{
    this.weight = weight;
    this.input_1 = input_1;
    this.input_2 = input_2;
    this.result = result;
}
```

3.4. Rule item script

Just like with the membership functions, an array of "Rule Item" is used to represent the "Rule List" class.

```
internal RuleItem[] ruleItems;
```

The "Rule List" class also contains a constructor that has the goal of assigning each rule weight based on the inputs required to activate it. For this, this constructor requires the three "Member List" variables that represent both the inputs and the output, with the two-dimensional rule base representing the full rule table. Since the rule base is stored as a two-dimensional array, the iteration inside this structure also requires a nested for-loop approach to correctly set the output membership function of each rule. Before this loop, the length of both dimensions of the two-dimensional "Rule Item" array is set, and inside the loop, each element of this array is created as an empty class followed by their "OnSet" function being called to set the weight of each rule. Upon setting each rule's weight, the minimum of both inputs for this rule is selected as the weight value, while which output membership should receive this weight is defined by the corresponding two-dimensional array's integer value at the current iteration index.

```
internal RuleBase(MemberList input_1, MemberList input_2,
MemberList output, int[,] rules)
{
    ruleItems = new RuleItem[input_1.members.Length,
input_2.members.Length];

    for (int i = 0; i < input_1.members.Length; i++)
        for (int j = 0; j < input_2.members.Length; j++)
        {
        ruleItems[i,j] = new RuleItem();
        ruleItems[i,j].OnSet(Math.Min(input_1.members[i].w
    eight, input_2.members[j].weight),
        $"{input_1.members[i].OnGetName()}",
        $"{input_2.members[j].OnGetName()}",
        $"{output.members[rules[i,j]].OnGetName()}");
    }
}
```

Apart from this version, one more slightly modified version of the constructure is required for single input case as well. The key difference here is that, this version of the constructure only requires one input. However, to preserve the overall structure the two-dimensional array is still used but with one dimension length set to one.

```
internal RuleBase(MemberList input, MemberList Output,
int[,] rules)
{
    ruleItems = new RuleItem[1, input.members.Length];

    for (int j = 0; j < input.members.Length; j++)
        {
        ruleItems[0, j] = new RuleItem();
        ruleItems[0,j].OnSet(Math.Min(input.members[j].wei
    ght, float.MaxValue),
        $"{input.members[j].OnGetName()}",
        "",
        $"{Output.members[rules[0, j]].OnGetName()}");
    }
}
```

Apart from this version, one more slightly modified version of the construct is required for a single input case as well. The key difference here is that this version of the constructor only requires one input. However, to preserve the overall structure the two-dimensional array is still used but with one dimension length set to one.

3.5. Defuzzification script

The script titled "Defuzzification" contains a function that performs the core calculation of a zero-order Sugeno fuzzy inference system. This method is widely used for its simplicity and effectiveness in producing crisp outputs based on fuzzy rule evaluations [26, 34]. In this case, the output membership functions are defined using symmetric triangular shapes, a common choice due to their balanced structure and computational efficiency. Because these shapes have an equal spread on both sides, the midpoint accurately represents the fuzzy set's output value. The defuzzification function multiplies each midpoint by its corresponding weight, sums the results, and divides by the total weight to produce a final crisp output.

```
internal    float    OnSugenoMethod(RuleItem[,]    rules,
MemberList output, int[,] ruleTable)
{
    float sum = 0, weights = 0;
    for (int i = 0; i < ruleTable.GetLength(0); i++)
        for (int j = 0; j < ruleTable.GetLength(1); j++)
        {
            int outIdx = ruleTable[i, j];
            float z = output.members[outIdx].points[1];
            float w = rules[i, j].weight;
            sum += w * z;
            weights += w;
        }
    return sum / weights;
}
```

This approach ensures fast and smooth control decisions, especially in real-time software systems [26, 31]. While this configuration is sufficient for validating the controller in this study, the modular design also supports adding alternative defuzzification methods in the future, making it easier to handle different membership shapes or more advanced requirements. In terms of code representation, this involves iteration through all of the output values, where the sum of each output membership function multiplied by its weight is divided by the total sum of the weights.

4. Initial validation of the proposed controller

Since the main objective of this paper is to describe and validate the proposed controller rather than focus on a particular case, the scenarios given here serve as a textbook example, which would not differ from any real-world application in terms of setup and validation. To verify the functionality and reliability of the proposed fuzzy logic controller, a series of test scenarios were conducted. Firstly, a scenario with single input and single output (SISO) is set up and tested, followed by the addition of a second input parameter for multiple input and single output (MISO) case testing. For both of these scenarios, custom "Fuzzy Logic Controller" scripts were written by utilizing the already discussed architecture. The main goal here is to ensure that the controller works with correct fuzzy reasoning and defuzzification behaviour, as well as to evaluate the object-oriented architecture's robustness in handling edge cases and overlapping membership evaluations [7]. For more clarity, additional print functions were used to obtain a console debug verification of the code execution, and the resulting outputs were also noted in this work. In addition to the print functions, graphical representations of each step were also made to further demonstrate the accuracy of this input.

4.1. Validation for SISO case

For the first validation case, a simple single input and single output scenario is presented. Here, the assumed goal is to keep the temperature of an arbitrary object at a constant, by adjusting the heating element appropriately based on the current temperature of the arbitrary object. The membership functions used for the input and output here are graphically shown in Fig. 11 and 12.

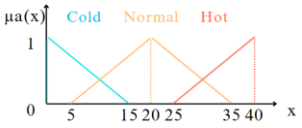

Fig. 11. Graphical illustration of the temperature input membership functions

For the temperature input, the cold region is defined between 0 and 15, the normal region as 5 to 35, and the hot region as 25 to 40 Celsius. These values are chosen for testing purposes.

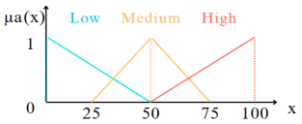

Fig. 12. Graphical illustration of the heater intensity output membership functions

For the temperature input, the low region is defined between 0 and 50, the medium region as 25 to 75, and the high region as 50 to 100 percent of the heater power. For this scenario, the following custom "Fuzzy Logic" constructor was written using the previously discussed scripts as building blocks.

```
internal FuzzyLogicController(float temperature)
{
    MemberList membersTemperature = new MemberList(new
MemberItem[] {
    new MemberItem("Temperature", "Cold", new float[] {0,
0, 15}),
    new MemberItem("Temperature", "Normal", new float[]
{5, 20, 35}),
    new MemberItem("Temperature", "Hot", new float[] {25,
40, 40})});
    membersTemperature.OnCalculateWeights(temperature);

    MemberList memberOutput = new MemberList(new
MemberItem[] {
    new MemberItem("Heater_Intensity", "Low", new float[]
{0, 0, 50}),
    new MemberItem("Heater_Intensity", "Medium", new
float[] {0, 50, 100}),
    new MemberItem("Heater_Intensity", "High", new
float[] {50, 100, 100})});

    int[,] ruleTable = new int[,] {
    {(int)Output.High, (int)Output.Medium,
(int)Output.Low}};

    RuleBase ruleBase = new RuleBase(membersTemperature,
memberOutput, ruleTable);

    memberOutput.OnCalculateWeights(ruleBase.ruleItems,
ruleTable);

    Console.Write($"Output: {new
Defuzzyfier().OnSugenoMethod(ruleBase.ruleItems,
memberOutput, ruleTable)}%\n");
}
```

Once the "Fuzzy Logic Controller" script is set up for this case, the constructor function input variable is set at 31 Celsius. After running the script, the following is printed in the consoles.

```
Temperature: 31
    -Cold: 0
    -Normal: 0.26666668
    -Hot: 0.4

Rules:
    -Temperature Normal give: Heater_Intensity Medium
0.26666668
    -Temperature Hot give: Heater_Intensity Low 0.4

Heater_Intensity: 0
    -Low: 0.4
    -Medium: 0.26666668
    -High: 0

Output: 20%
```

By analyzing the information printed in the consoles, we can conclude that for this case, when the temperature is 31 Celsius, it falls into the normal region by 0.26 or 26% and the hot region by 0.4 or 40%, which we can verify visually by drawing a line in the graphical representation as shown in Fig. 13.

Apart from the membership degrees of the temperature, the currently active rules are also printed. From here, we can see that the rule base is also accurate, which states that when the temperature is "Hot", the heater should be set to "Low". Lastly, to verify that the final output of 0.2 or 20%, we can draw the output membership functions to visually confirm that the center of gravity corresponds with the obtained output Fig. 14.

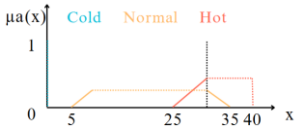

Fig. 13. Active memberships functions when temperature input is 31 Celsius

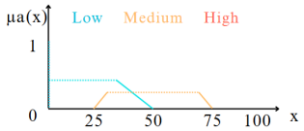

Fig. 14. Graphical illustration of heater intensity output for the SISO case

4.2. Validation for MISO case

A more complex but also commonly encountered scenario is when multiple factors must be taken into consideration before adjusting the output. For this reason, this work also validates the proposed object-oriented fuzzy logic controller for the MISO scenario as well. This scenario builds upon the SISO case, by keeping the temperature and the heater intensity the same, adding one more input, and extending the rule base into a full 2-dimensional array. For this case, an additional parameter is chosen as pressure Fig. 15, since it is one of the commonly used parameters taken into consideration in thermal applications.

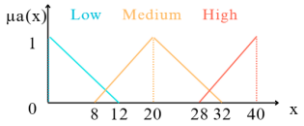

Fig. 15. Graphical illustration of the pressure input membership functions

For this case, the pressure is defined as low from 0 to 12, normal from 8 to 32 and high from 28 to 40. With addition of the additional parameter, the rule base also needs to be expanded to support all of the possible cases. In the Table 1, complete rule based which will be defined inside the code is shown.

Table. 1. Rule base used for the MISO case

Temperature	Pressure		
	Low	Medium	High
Cold	High	Medium	Low
Normal	Medium	Medium	Low
Hot	Low	Low	Low

Once the new parameter and the updated rule base are known, the previously written code can be updated to test and validate the MISO case as shown below.

```
internal FuzzyLogicController(float temperature, float
pressure)
{
    MemberList membersTemperature = new MemberList(new
MemberItem[] {
    new MemberItem("Temperature", "Cold", new float[] {0,
0, 15}),
    new MemberItem("Temperature", "Normal", new float[]
{5, 20, 35}),
    new MemberItem("Temperature", "Hot", new float[] {25,
40, 40})});
     membersTemperature.OnCalculateWeights(temperature);

    MemberList memberPressure = new MemberList(new
MemberItem[] {
```

```
    new MemberItem("Pressure", "Low", new float[] {0, 0,
12}),
    new MemberItem("Pressure", "Medium", new float[] {8,
20, 32}),
    new MemberItem("Pressure", "High", new float[] {28,
40, 40})});
    memberPressure.OnCalculateWeights(pressure);

    MemberList memberOutput = new MemberList(new
MemberItem[] {
    new MemberItem("Heater_Intensity", "Low", new float[]
{0, 0, 50}),
    new MemberItem("Heater_Intensity", "Medium", new
float[] {0, 50, 100}),
    new MemberItem("Heater_Intensity", "High", new
float[] {50, 100, 100})});

    int[,] ruleTable = new int[,] {
    {(int)Output.High, (int)Output.Medium,
(int)Output.Low},
    {(int)Output.Medium, (int)Output.Medium,
(int)Output.Low},
    {(int)Output.Low, (int)Output.Low, (int)Output.Low}};

    RuleBase ruleBase = new RuleBase(membersTemperature,
memberPressure, memberOutput, ruleTable);

    memberOutput.OnCalculateWeights(ruleBase.ruleItems,
ruleTable);

    Console.Write($"Output: {new
Defuzzyfier().OnSugenoMethod(ruleBase.ruleItems,
memberOutput, ruleTable)}%\n");
}
```

In this case, when calling the "Fuzzy Logic Controller" script, the constructor function requires two inputs chosen as the temperature kept at the same 31 Celsius and the newly introduced pressure variable set to 31 bars. This is done to see the impact of the second parameter and expanded rule based on the final decision for the heater intensity output. After running the script again, the following was printed on the consoles.

```
Temperature: 31
    -Cold: 0
    -Normal: 0.26666668
    -Hot: 0.4

Pressure: 31
    -Low: 0
    -Medium: 0.083333336
    -High: 0.25

Rules:
    -Temperature Normal & Pressure Medium gives:
Heater_Intensity Medium 0.083333336
    -Temperature Normal & Pressure High gives:
Heater_Intensity Low 0.25
    -Temperature Hot & Pressure Medium gives:
Heater_Intensity Low 0.083333336
    -Temperature Hot & Pressure High gives:
Heater_Intensity Low 0.25

Heater_Intensity: 0
    -Low: 0.25
    -Medium: 0.083333336
    -High: 0

Output: 6.2500005%
```

By analyzing the consoles again, we can conclude that the second input pressure at 31 bars falls into the medium region by 0.08 or 8% and the high region by 0.25 or 25%, which can be verified by drawing the graphical representation in Fig. 16.

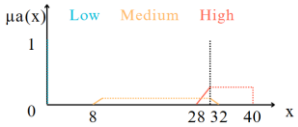

Fig. 16. Active memberships functions when pressure input is 31 bars

Apart from the inputs, the active rules of the current rule base are also printed. Since the rule base was set to lower the heater intensity if either the temperature or the pressure was too high, this time the final output of 0.0625 or 6.25% was lower than the SISO case. This indicates that the addition of pressure impacted the final decision of the controller appropriately, and the output membership functions are represented in Fig. 17.

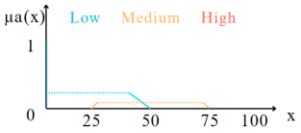

Fig. 17. Graphical illustration of heater intensity output for the MISO case

5. Comparison with the MATLAB framework

MATLAB, through its Fuzzy Logic Toolbox, remains one of the most widely used environments for developing and analyzing fuzzy controllers. To enhance confidence in the proposed framework, a comparative validation was performed against MATLAB using the same SISO and MISO cases from initial validation tests. The same universes of discourse, membership functions, and complete rule base were reimplemented in MATLAB to ensure identical inference behavior. Afterwards, a set of randomly generated input signals was created in MATLAB to represent measurement noise and operating variability. These inputs were applied to the MATLAB fuzzy controller, and the resulting output responses were recorded. The same input sequence was then used to test the proposed C# based object-oriented fuzzy logic controller, implemented through a loop-based evaluation structure while maintaining the same rule base and parameter configuration. Quantitative evaluation between the two systems was performed using Root Mean Square Error (RMSE), Mean Absolute Error (MAE), and Maximum Absolute Difference (Max Diff), which collectively measure the overall, average, and worst-case deviations between the output signals.

5.1. Results & benchmarks for the SISO case

Fig. 18 shows the MATLAB model used for the SISO validation case. The setup includes a random number generator block to create varying input values, an FLC block implementing the same rule base and membership functions as the initial validation, and "To Workspace" blocks to record both the input and output signals for plotting. This ensured identical test conditions for both controllers. The resulting input signals were produced during a ten-step simulation using a fixed random seed to ensure repeatability, and are plotted in Fig. 19.

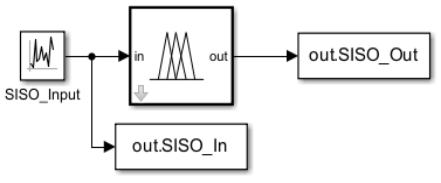

Fig. 18. MATLAB subsystem of the SISO fuzzy logic controller with random signal generator and data capture blocks

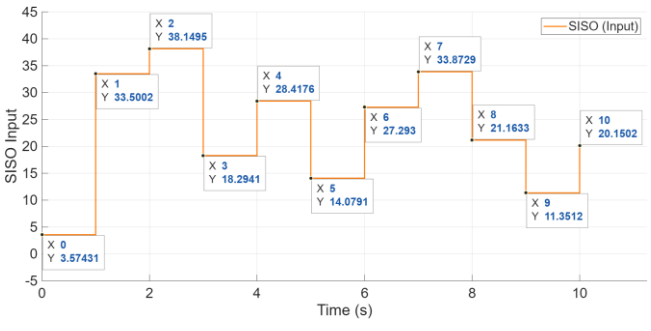

Fig. 19. Input signal of the SISO case showing 10 random samples generated during a 10-step simulation (seed = 10000; Min = 0; Max = 40; Sample Time = 1)

The corresponding numerical values of these samples are summarized in Table 2, which serves as the baseline dataset for evaluating both the MATLAB and object-oriented fuzzy logic controllers under identical conditions.

Table. 2. Recorded SISO input values obtained from MATLAB simulation

Step	Randomly Selected Input Values (Range 0 to 40) (Seed 10000)
0	3.574314286734124
1	33.500217140419508
2	38.149479030701087
3	18.294068993206167
4	28.417568816066520
5	14.079091629981573
6	27.293025100274487
7	33.872860313333973
8	21.163286204060206
9	11.351231639902682
10	20.150171844358638

After obtaining the input signals, these "To Workspace" output values were used to create the float array, which served as the input test for the SISO case, and their values were noted down from the console to Table 3.

```
float[] SISO_Input = new float[]
{
    3.574314286734124f,
    33.500217140419508f,
    38.149479030701087f,
    18.294068993206167f,
    28.417568816066520f,
    14.079091629981573f,
    27.293025100274487f,
    33.872860313333973f,
    21.163286204060206f,
    11.351231639902682f,
    20.150171844358638f
};

for (int i = 0; i < SISO_Input.Length; i++)
{
    new FuzzyLogicController(SISO_Input[i]);
}
```

For the MATLAB SISO FLC, after running the simulation, the output signal was plotted in a similar way to the same input signal and plotted in Fig. 20.

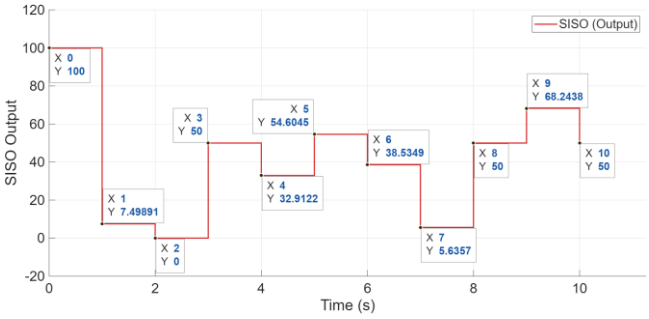

Fig. 20. Output response of the SISO fuzzy logic controller to the generated input signals

Table. 3. Comparison between MATLAB and the proposed Modular Object-Oriented Fuzzy Logic Controller under the same SISO configuration

Step	Resulting Output Signals	
	MATLAB	Proposed Controller
0	100	99.99999
1	7.498914297902460	7.4989133
2	0	0
3	50.000000000000007	50
4	32.912155919667399	32.91216
5	54.604541850092126	54.604538
6	38.534874498627559	38.53487
7	5.635698433330134	5.6356997
8	50.000000000000007	50
9	68.243841800486578	68.24384
10	50	50

To quantify how closely the two controllers aligned, Table 3 shows the MATLAB FLC output with that of the proposed object-oriented FLC. The very small numerical deviations observed confirm that the new implementation produces nearly identical behavior to MATLAB under SISO conditions.

To further verify the accuracy of the proposed fuzzy logic controller, its output was compared against MATLAB's response using three common statistical measures. The Root Mean Square Error (RMSE) reflects the overall difference between the two signals, while the Mean Absolute Error (MAE) captures the average deviation across all samples. The Maximum Absolute Difference (Max Diff) highlights the single largest deviation, providing a sense of worst-case behavior. Together, these metrics offer a balanced view of both overall and localized accuracy. The computed results using values from Table 3 for the SISO case comparison are summarized in Table 4, demonstrating a close match between both systems.

Table. 4. Error between MATLAB and the proposed Modular Object-Oriented Fuzzy Logic Controller under the same SISO conditions

Case	RMSE	MAE	Max Diff
SISO	3.0869×10^{-6}	1.9507×10^{-6}	7.6294×10^{-6}

5.2. Results & benchmarks for the MISO case

To further validate and benchmark scalability, the same process was repeated for a MISO configuration. Fig. 21 shows the MATLAB subsystem for this case, where two independent random signal generators were combined using a MUX block to serve as dual inputs to the fuzzy controller.

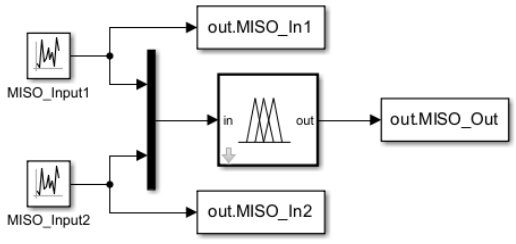

Fig. 21. MATLAB subsystem of the MISO fuzzy logic controller using multiplexed random signal generators with different seeds

Their corresponding numerical values are provided in Table 5. The individual input signals generated by each source are plotted in Fig. 22 and Fig. 23, respectively.

Table. 5. Recorded MISO input values obtained from MATLAB simulation

Step	Randomly Selected Input Values (Range 0 to 40)	
	Input 1 (Seed 51000)	Input 2 (Seed 75000)
0	24.486868523287988	6.807044095735551
1	30.799270901269871	5.990117027419673
2	3.346037642725761	35.896879842456840
3	36.854661291863145	38.859512172108289
4	16.292332343893282	31.821076624012122
5	25.229703814363901	16.834819771738175
6	35.632008014075460	22.815903603479221
7	27.158692566286163	26.891863675272027
8	16.145961571552775	11.552790296987068
9	5.176133087452564	7.746521461637003
10	35.268800815226882	35.786205733095393

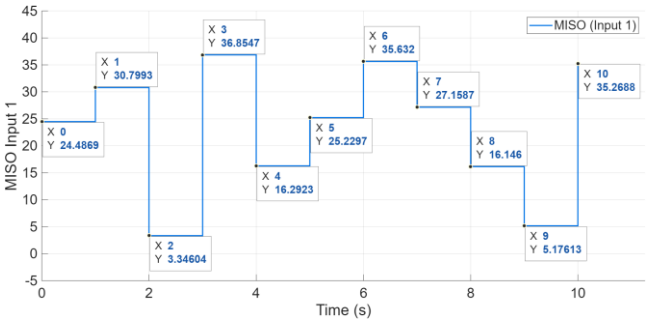

Fig. 22. First input signal of the MISO case showing 10 random samples generated during a 10-step simulation (seed = 51000; Min = 0; Max = 40; Sample Time = 1)

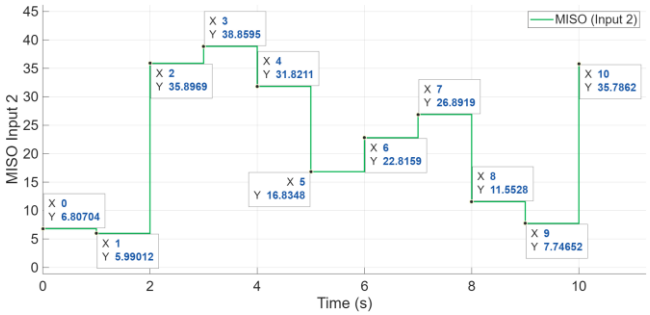

Fig. 23. Second input signal of the MISO case showing 10 random samples generated during a 10-step simulation (seed = 75000; Min = 0; Max = 40; Sample Time = 1)

After obtaining both input signals, these "To Workspace" output values were used to create the float arrays, which served as the input test for the MISO case, and the object-oriented FLC output values were noted down from the console to Table 6.

```
float[] MISO_Input1 = new float[]
{
    24.486868523287988f,
    30.799270901269871f,
    3.346037642725761f,
    36.854661291863145f,
    16.292332343893282f,
    25.229703814363901f,
    35.632008014075460f,
    27.158692566286163f,
    16.145961571552775f,
    5.176133087452564f,
    35.268800815226882f
};

float[] MISO_Input2 = new float[]
{
     6.807044095735551f,
     5.990117027419673f,
     35.896879842456840f,
     38.859512172108289f,
     31.821076624012122f,
     16.834819771738175f,
     22.815903603479221f,
     26.891863675272027f,
     11.552790296987068f,
     7.746521461637003f,
     35.786205733095393f,
};

for (int i = 0; i < MISO_Input1.Length; i++)
{
    new FuzzyLogicController(MISO_Input1[i],
MISO_Input2[i]);
}
```

The output response of the MATLAB MISO FLC to these two varying input signals is plotted in Fig. 24. Compared to the SISO case, in the MISO case, the output demonstrates adaptive behavior influenced by both inputs, confirming the correct rule interaction and multi-input handling of the FLC setup.

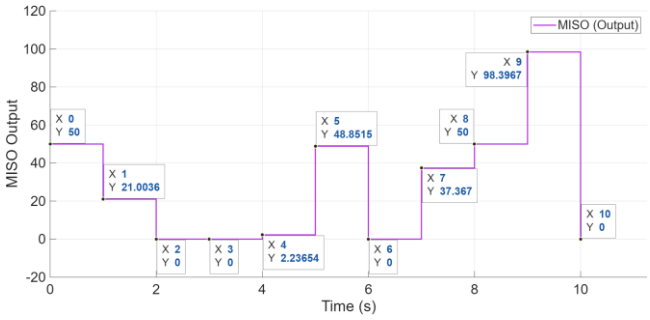

Fig. 24. Output response of the MISO fuzzy logic controller to the two generated input signals

These values were also noted down in Table 6, which provides a direct comparison between MATLAB and the proposed controller outputs under the MISO configuration.

Table. 6. Comparison between MATLAB and the proposed Modular Object-Oriented Fuzzy Logic Controller under the same MISO configuration

Step	Resulting Output Signals	
	MATLAB	Proposed Controller
0	50.000000000000000	50
1	21.003645493650644	21.003647
2	0	0
3	0	0
4	2.236542199848479	2.2365332
5	48.851480928180493	48.851482
6	0	0
7	37.366999135454947	37.366997
8	49.999999999999993	50.000004
9	98.396744144002994	98.39675
10	0	0

The same evaluation approach was applied to the MISO configuration to verify consistency under multi-input conditions using the values from Table 6. The resulting error metrics are summarized in Table 7, showing that the proposed controller closely matches the MATLAB output.

Table. 7. Error metrics between MATLAB and the proposed Modular Object-Oriented Fuzzy Logic Controller under the same MISO conditions

Case	RMSE	MAE	Max Diff
MISO	3.9661×10^{-6}	2.3842×10^{-6}	9.0599×10^{-6}

6. Implementation strategies for the adaptive and observer-based control

While the primary focus of this work is on the development and validation of an MOO-FLC (Modular Object-Oriented Fuzzy Logic Controller), the framework also provides a foundation for implementing adaptive and observer-based control strategies. Such strategies enhance controller intelligence by allowing parameters to be modified in real time or enabling the estimation of unmeasurable system states, thereby maintaining consistent performance under varying or uncertain conditions [7, 8, 17, 25, 38, 39]. In traditional fuzzy systems, membership functions, scaling factors, and rule bases are often defined once and later transformed into static lookup tables following the tuning phase.

However, in dynamic environments where operating conditions change, these fixed configurations may lose relevance, leading to reduced control accuracy and slower response. The object-oriented design of the proposed controller may address this issue by defining the required parameters as class variables or function inputs, allowing them to be modified during runtime.

One straightforward approach is to expose membership function limits, rule weights, or scaling factors through dedicated update functions. This allows the system to adjust its parameters based on performance metrics such as steady-state error or overshoot. A more advanced solution would encapsulate all controller elements within a unified data structure containing both parameter values and links to the necessary functions. This structure could then be modified dynamically by an adaptive algorithm, observer, or external learning routine.

Real-time modification of the rule base is among the more challenging aspects of adaptive fuzzy control. It can be achieved in two main ways: either by dynamically rebuilding the rule table or by adjusting rules within an existing structure without full reconstruction. Both approaches enable adaptation to changing conditions but differ in flexibility and computational cost. The current demonstration uses a multidimensional array that supports any number of inputs, though this approach is impractical for complex systems with more than two inputs, where a dedicated rule reader could be more manageable.

The first method involves dynamically rebuilding the rule table. Instead of using a fixed array with predefined indexes, the system can implement a "rule reader" that interprets rules from a flexible array structure. This approach allows complete freedom to add, remove, or modify rules during execution. However, it can be computationally expensive because each rule must be indexed, decoded, and processed at runtime, which may increase execution time on embedded hardware.

The second method keeps the rule table structure intact but allows adjustments through two alternative strategies. One is a deactivation method, where each rule maintains an internal state that determines whether it is active. This requires an additional conditional check for every rule during evaluation, but does not need extra arrays. The other is a control array approach, where rule weights can be modified directly through a separate array without changing the table itself. This avoids the if-check overhead but introduces additional memory usage for the array.

Both methods have practical applications. Dynamically rebuilding the table is ideal for systems requiring maximum adaptability, while deactivation or control array strategies provide faster runtime performance on limited hardware.

7. Discussion

This work proposed, implemented, and validated an open source, MOO-FLC (Modular Object-Oriented Fuzzy Logic Controller) architecture developed using the .NET framework and C#. Rather than attempting to replace existing fuzzy logic platforms such as MATLAB or LabVIEW, the objective was to provide a lightweight and flexible alternative capable of deeper system-level integration, especially in situations where running computation-heavy licensed software is impractical. While the proposed architecture is more flexible in terms of structure and development resource use, its runtime execution is expected to be slower than precompiled lookup table methods. However, this trade-off brings significant advantages in adaptability and transparency over the internal operation of the controller.

A major strength of the proposed framework lies in its modular design, which allows real-time modification of membership functions and rule parameters during operation. This capability opens up new possibilities for hybrid and adaptive control strategies that go beyond the static nature of traditional FLC implementations. While most current research focuses on optimizing the MATLAB FLC toolbox through new algorithms or auto-tuning routines [32, 33], the presented work explores the controller structure itself, demonstrating how fuzzy systems can be dynamically adjusted rather than merely optimized offline.

Although the implementation is written in C#, the same object-oriented concept can be applied inside MATLAB as well. MATLAB already supports OOP, meaning that a heuristic version of this approach could be embedded directly within it. This would allow the controller to readjust its membership function values dynamically, potentially reducing steady state error and even enabling an FLC auto-tuner similar to the one available for PID controllers.

The framework also enables direct integration with simulation environments such as Unity, which is typically difficult when using MATLAB-based lookup tables [21]. Unlike exported tables, which are static and limited in customizability, the presented architecture allows complete scripting control. This makes it possible to combine fuzzy logic with advanced learning or decision-making systems such as hybrid Fuzzy-PPO (Proximal Policy Optimization) controllers using Unity ML Agents package, SM-Fuzzy (State Machine) models for state-dependent control, or a metaheuristic SM-Fuzzy-PPO framework that merges the state machine patterns with fuzzy reasoning and reinforcement learning.

Beyond simulation, the proposed design offers advantages for IoT and embedded applications, where direct source code access within the hardware may simplify debugging and support the implementation of more unique control techniques. Having full transparency over every fuzzy parameter on the device can also accelerate development and improve maintainability over time.

While the current version lacks some of the visualization and tuning tools found in industrial-grade software, it successfully demonstrates that a complete and adaptable fuzzy controller can be implemented with minimal dependencies. Its open-source nature encourages experimentation, modification, and integration across various platforms. Future developments may include broader support for membership function types, visual tuning tools, a rule reader, automated embedded deployment, and many more features. Overall, this work lays a foundation for accessible and extensible fuzzy control systems that combine interpretability with flexibility, bridging the gap between classical fuzzy control and modern adaptive methods.

In addition to this, recent events have highlighted the risks of relying exclusively on proprietary software for control system development. On May 18, 2025, MathWorks confirmed a ransomware attack that disrupted access to MATLAB, licensing services, and cloud-based toolboxes. Later, users of MATLAB R2025b reported that the formerly bundled Simulink Specialized Power Systems (SPS) library was removed from the standard installation and its commercialization rights were transferred to OPAL-RT Technologies.

While MATLAB remains a powerful platform for simulation and validation, these incidents demonstrate how dependence on a single vendor can limit access, functionality, and continuity of workflows. This reinforces the importance of adopting open-source or modular alternatives that provide greater resilience and developer control [19, 20].

8. Conclusion

In conclusion, this paper presented a MOO-FLC (Modular Object-Oriented Fuzzy Logic Controller) developed with C#, built from the ground up by translating each functional component of the fuzzy logic system into separate scripts. This approach emphasized clarity, reusability, and open-source accessibility while maintaining compatibility with object-oriented principles.

Following the development phase, a validation process was conducted to ensure correct operation. The first stage involved a single-input single-output (SISO) test case designed to confirm that the proposed controller performed all fuzzy inference steps as expected. After confirming proper functionality, the same structure was extended to a multi-input single-output (MISO) configuration to assess scalability under additional input dimensions. Both configurations operated using identical rule base structures and membership function definitions.

To verify accuracy, the proposed controller was benchmarked against MATLAB's Fuzzy Logic Toolbox. Using identical membership functions, universes of discourse, and randomly generated input sequences, both systems were evaluated under matching test conditions. In the SISO case, the root mean square error (RMSE) was 3.0869×10^{-6} with a maximum deviation of 7.6294×10^{-6}. In the MISO case, the RMSE was 3.9661×10^{-6} and the maximum deviation was 9.0599×10^{-6}. The minimal differences between results demonstrate that the proposed implementation accurately replicates MATLAB's fuzzy inference behavior.

While the framework is lightweight in terms of software development, its runtime execution is slightly slower compared to precompiled lookup-table-based approaches. However, this trade-off provides valuable flexibility, allowing real-time modification of membership functions, a capability typically unavailable in fixed-table systems. Since MATLAB also supports object-oriented programming, this architecture could be incorporated within MATLAB as a heuristic fuzzy controller capable of readjusting membership parameters dynamically at runtime to reduce steady-state error or support fuzzy auto-tuning processes.

Overall, the results demonstrate that the proposed MOO-FLC offers an accurate, extensible, and transparent alternative to license-based fuzzy control platforms. Its flexibility and open-source nature make it a promising option for educational use, embedded systems, and integration with simulation environments such as Unity or IoT-based control frameworks.

References

[1] Abdrakhmanov R. et al.: Intelligent Fuzzy-PID Temperature Control System for Ensuring Comfortable Microclimate in an Intelligent Building. IJACSA 15(3), 2024 [https://doi.org/10.14569/IJACSA.2024.0150331].

[2] Aliyeva K.: Estimation of Renewable Energy Sources under Uncertainty Using Fuzzy AHP Method. Informatyka, Automatyka, Pomiary w Gospodarce i Ochronie Środowiska 15(2), 104–109, 2025 [https://doi.org/10.35784/iapgos.7191].

[3] Bakirova L., Yusubov E.: Design and Simulation of the Auto-Tuning TS-Fuzzy PID Controller for the DC-DC ZETA Converter. In CEUR Workshop Proceedings, 238–243, 2021 [https://elibrary.ru/item.asp?id=49020698].

[4] Belman-Flores, J. M. et al.: A Review on Applications of Fuzzy Logic Control for Refrigeration Systems. Applied Sciences 12(3), 1302, 2022 [https://doi.org/10.3390/app12031302].

[5] Brucal S. G. E., Africa A. D. M., de Jesus L. C. M.: Optimizing Air Conditioning Unit Power Consumption in an Educational Building: A Rough Set Theory and Fuzzy Logic-Based Approach. Applied System Innovation 8(2), 2025, 32 [https://doi.org/10.3390/asi8020032].

[6] Boll A. et al.: Characteristics, potentials, and limitations of open-source Simulink projects for empirical research. Software and Systems Modeling 20(6), 2021, 2111–2130 [https://doi.org/10.1007/s10270-021-00883-0].

[7] Boulkroune A. et al.: Output-Feedback Controller Based Projective Lag-Synchronization of Uncertain Chaotic Systems in the Presence of Input Nonlinearities. Mathematical Problems in Engineering 2017(1), 8045803 [https://doi.org/10.1155/2017/8045803].

[8] Boulkroune A., Zouari F., Boubellouta A.: Adaptive Fuzzy Control for Practical Fixed-Time Synchronization of Fractional-Order Chaotic Systems. Journal of Vibration and Control 10775463251320258, 2025 [https://doi.org/10.1177/10775463251320258].

[9] Boll A., Vieregg N., Kehrer T.: Replicability of experimental tool evaluations in model-based software and systems engineering with MATLAB/Simulink. Innovations in Systems and Software Engineering 20(3), 2024, 209–224 [https://doi.org/10.1007/s11334-022-00442-w].

[10] Chen Y., Huang L.: Object-Oriented Programming. Chen Y., Huang L. (eds): MATLAB Roadmap to Applications: Volume I Fundamental, Springer Nature, Singapore, 2025, 399–439 [https://doi.org/10.1007/978-981-97-8788-3_10].

[11] Chojecki A., Ambroziak A., Borkowski P.: Fuzzy Controllers Instead of Classical PIDs in HVAC Equipment: Dusting Off a Well-Known Technology and Today's Implementation for Better Energy Efficiency and User Comfort. Energies 16(7), 2023, 2967 [https://doi.org/10.3390/en16072967].

[12] Dumitrescu C., Ciotirnae P., Vizitiu C.: Fuzzy Logic for Intelligent Control System Using Soft Computing Applications. Sensors 21(8), 2021, 2617 [https://doi.org/10.3390/s21082617].

[13] Ferrara P., Arceri V., Cortesi A.: Challenges of software verification: the past, the present, the future. International Journal on Software Tools for Technology Transfer 26(4), 2024, 421–430 [https://doi.org/10.1007/s10009-024-00765-y].

[14] García-Martínez J. R. et al.: A PID-Type Fuzzy Logic Controller-Based Approach for Motion Control Applications. Sensors 20(18), 2020, 5323 [https://doi.org/10.3390/s20185323].

[15] Haghrah A. A., Ghaemi S., Badamchizadeh M. A.: PyIT2FLS: An open-source Python framework for flexible and scalable development of type 1 and interval type 2 fuzzy logic models. SoftwareX 30, 2025, 102146 [https://doi.org/10.1016/j.softx.2025.102146].

[16] Hongli L., Peiyong D., Lei J.: A Novel Fuzzy Controller Design based-on PID Gains for HVAC Systems. 7th World Congress on Intelligent Control and Automation, 2008, 736–739 [https://doi.org/10.1109/WCICA.2008.4593014].

[17] Merazka L., Zouari F., Boulkroune A.: High-Gain Observer-Based Adaptive Fuzzy Control for a Class of Multivariable Nonlinear Systems. 6th International Conference on Systems and Control (ICSC), 96–102 [https://doi.org/10.1109/ICoSC.2017.7958728].

[18] Mammadov R.: Optimal Load Shedding for Power Systems Using the Binary Exhaustive Search Method. International Conference on Intelligent and Fuzzy Systems. Springer Nature Switzerland, Cham 2025, 175–183 [https://doi.org/10.1007/978-3-031-98304-7_20].

[19] Mammadzada R.: A Script-Based Approach for Automating Fuzzy-PID Rule Assignment from 7×7 Tables to FIS Files. 6th International Conference on Problems of Cybernetics and Informatics (PCI), Baku, Azerbaijan, 2025, 1–4 [https://doi.org/10.1109/PCI66488.2025.11219882].

[20] Mammadzada R., Aliyeva K.: Modulation For Information Measurement System In Context Of Software Development Using Node Based Graphical Editor. International Research, Education & Training Center, 24(3), 2023, 143–151 [https://doi.org/10.36962/piretc24032023-143].

[21] Mammadzada R.: Virtual Temperature Control in Unity Environment with ML Agents. Azerbaijan Higher Technical Educational Institutions 26(3-4), 2024, 51–55 [https://doi.org/10.36962/PAHTEI149032024-51].

[22] Matos V. S. et al.: Embedded predictive controller based on fuzzy linear parameter-varying model: A hardware-in-the-loop application to an ESP-lifted oil well system. Digital Chemical Engineering 5, 2022, 100054 [https://doi.org/10.1016/j.dche.2022.100054].

[23] Pérez-Juárez J. G. et al.: Kinematic Fuzzy Logic-Based Controller for Trajectory Tracking of Wheeled Mobile Robots in Virtual Environments. Symmetry 17(2), 2025, 301 [https://doi.org/10.3390/sym17020301].

[24] Rajeswari Subramaniam K., Cheng C. T., Pang T. Y.: Fuzzy Logic Controlled Simulation in Regulating Thermal Comfort and Indoor Air Quality Using a Vehicle Heating, Ventilation, and Air-Conditioning System. Sensors 23(3), 1395, 2023 [https://doi.org/10.3390/s23031395].

[25] Rigatos G. et al.: Nonlinear Optimal Control for a Gas Compressor Driven by an Induction Motor. Results in Control and Optimization 11, 100226, 2023 [https://doi.org/10.1016/j.rico.2023.100226].

[26] Saatchi R.: Fuzzy Logic Concepts, Developments and Implementation. Information 15(10), 2024, 656 [https://doi.org/10.3390/info15100656].

[27] Sheng T., Luo H., Wu M.: Design and Simulation of a Multi-Channel Biomass Hot Air Furnace with an Intelligent Temperature Control System. Agriculture 14(3), 2024, 419 [https://doi.org/10.3390/agriculture14030419].

[28] Shrestha S. L., Chowdhury S. A., Csallner C.: SLNET: A Redistributable Corpus of 3rd-party Simulink Models. IEEE/ACM 19th International Conference on Mining Software Repositories (MSR), 2022, 1–5 [https://doi.org/10.1145/3524842.3528001].

[29] Spolaor S. et al.: Simpful: A User-Friendly Python Library for Fuzzy Logic. International Journal of Computational Intelligence Systems 13(1), 2020, 1687–1698 [https://doi.org/10.2991/ijcis.d.201012.002].

[30] Sun W. et al.: Fuzzy Control Algorithm Applied on Constant Airflow Controlling of Fans. Energies 16(11), 4425, 2023 [https://doi.org/10.3390/en16114425].

[31] Tang H. H., Ahmad N. S.: Fuzzy logic approach for controlling uncertain and nonlinear systems: a comprehensive review of applications and advances. Systems Science & Control Engineering 12(1), 2024, 2394429 [https://doi.org/10.1080/21642583.2024.2394429].

[32] Xakimovich S. I., Maxamadjonovna U. D.: Fuzzy-Logical Control Models of Nonlinear Dynamic Objects. Advances in Science, Technology and Engineering Systems 5(4), 419–423, 2020 [https://doi.org/10.25046/aj050449].

[33] Yusubov E., Bakirova L.: A Self-Tuning Fuzzy PID Controller for SEPIC Based on Takagi-Sugeno Inference System. International Conference Automatics and Informatics (ICAI), IEEE, 2021, 54–57 [https://doi.org/10.1109/ICAI52893.2021.9639804].

[34] Zadeh L. A.: Fuzzy sets. Information and Control 8, 1965, 338–353.

[35] Zhang D., Chen T.: Scikit-ANFIS: A Scikit-Learn Compatible Python Implementation for Adaptive Neuro-Fuzzy Inference System. International Journal of Fuzzy Systems 26(6), 2024, 2039–2057 [https://doi.org/10.1007/s40815-024-01697-0].

[36] Zhang S., Li T., Chen L.: Fuzzy Logic Control of External Heating System for Electric Vehicle Batteries at Low Temperature. WEVJ 14(4), 2023, 99 [https://doi.org/10.3390/wevj14040099].

[37] Zhang W. et al.: An empirical study of manual abstraction between class diagrams and code of open-source systems. Software and Systems Modeling, 2025 [https://doi.org/10.1007/s10270-025-01289-y].

[38] Zouari F., Ben Saad K., Benrejeb M.: Adaptive Backstepping Control for a Class of Uncertain Single Input Single Output Nonlinear Systems. 10th International Multi-Conferences on Systems, Signals & Devices 2013 (SSD13), 1–6 [https://doi.org/10.1109/SSD.2013.6564134].

[39] Zouari F., Saad K. B., Benrejeb M.: Robust Neural Adaptive Control for a Class of Uncertain Nonlinear Complex Dynamical Multivariable Systems. International Review on Modelling and Simulations 5(5), 2075–2103, 2012 [https://www.scopus.com/pages/publications/84873265173].

M.Sc. Rahim Mammadzada
e-mail: rahim.mammadzada02@gmail.com

Rahim Mammadzada, born in Azerbaijan on September 17, 2000, obtained a B.Sc. degree in "Instrumentation Engineering" in 2021, and an M.Sc. degree in "Computerized-Information Measurement Technologies" with distinction in 2023. Currently, he is a second-year Ph.D. student in "Information-Measurement and Control Systems" at the Azerbaijan State Oil and Industry University. Rahim's research interests encompass control theory, temperature control, fuzzy logic, and object-oriented programming.

https://orcid.org/0009-0008-7552-2790