Project Based Learning Report

On

Fuzzy Logic Controller for Temperature Regulation

Submitted in the partial fulfilment of the requirement

For the Project Based Learning in **Fuzzy Logic, Neural Network & Genetic Algorithms**

In

Electronics & Communication Engineering

By

2214110416 Harshit Agrawal

2214110425 Rahul Raj

2214110429 Vardan Rastogi

Under the guidance of **V .P. Kaduskar**

Department of Electronics & Communication Engineering

Bharati Vidyapeeth
(Deemed to be University)
College Of Engineering
Pune – 410043

Academic Year : 2024-25

Bharati Vidyapeeth
(Deemed to be University)
College of Engineering,
Pune – 411043

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

CERTIFICATE

Certified that the Project Based Learning report entitled, **"Fuzzy Logic Controller for Temperature Regulation"**

is work done by

2214110416 Harshit Agrawal

2214110425 Rahul Raj

2214110429 Vardan Rastogi

in partial fulfilment of the requirements for the award of credits for Project Based Learning (PBL) in of **Fuzzy Logic, Neural Network & Genetic Algorithm** Bachelor of Technology Semester IV, in Electronics & Communication Engineering.

Date:

Prof. V.P. Kaduskar
Course In-charge

Dr. Arundhati A. Shinde
Professor & Head

Index

Introduction

In modern control systems, maintaining a stable and desired temperature is crucial across various industries, including HVAC systems, industrial processing, and consumer electronics. Traditional control methods like Proportional-Integral-Derivative (PID) controllers often struggle in environments with uncertain, non-linear dynamics or where precise mathematical models are unavailable. To overcome these challenges, fuzzy logic controllers (FLCs) offer a robust alternative by simulating human-like reasoning to handle imprecise or uncertain data.

A Fuzzy Logic Controller for temperature regulation leverages fuzzy logic principles to manage temperature variations by making decisions based on linguistic variables such as "cold," "warm," or "hot," rather than relying on rigid numerical thresholds. The controller continuously adjusts the system based on real-time feedback from sensors, providing smooth, adaptive control. This flexibility allows FLCs to perform better than conventional controllers in complex or dynamic environments.

The implementation of a fuzzy logic controller for temperature regulation involves:

- Fuzzification of temperature inputs (converting precise sensor data into fuzzy values like "low," "medium," or "high").
- Rule evaluation through a knowledge base of "if-then" rules that define the controller's response to various temperature states.
- Defuzzification to convert fuzzy outputs back into precise control actions, such as adjusting a heater, fan speed, or cooling system.

The advantage of using fuzzy logic for temperature regulation lies in its ability to handle non-linearities, provide smoother transitions between states, and respond effectively to changes in the environment without requiring detailed mathematical models. This makes fuzzy logic controllers ideal for applications where conditions are constantly shifting, such as smart home systems, greenhouses, or industrial machinery.

In this project, the design and implementation of a fuzzy logic controller will be explored for the task of regulating temperature, with a focus on developing a system that is adaptable, efficient, and easy to maintain.

Problem Statement

Create a Fuzzy Logic Controller for temperature Regulation **Objective**:

Using fuzzy logic design a controller that can help in temperature regulation, slightest change in room temperature should affect the controller to show change.

Fuzzy Logic Controller

A Fuzzy Logic Controller (FLC) is a type of control system that uses fuzzy logic to make decisions. Instead of relying on precise inputs and mathematical models, like traditional control systems, a fuzzy logic controller processes vague or imprecise inputs and simulates human decision-making. This makes it particularly useful in situations where systems are difficult to model mathematically or where the relationships between variables are not linear.

Key Components of a Fuzzy Logic Controller:

1. Fuzzification Interface: Converts crisp numerical inputs (e.g., temperature, speed) into fuzzy sets using membership functions. This step transforms precise data into a form that fuzzy logic can work with.
2. Fuzzy Rule Base: Contains a set of "if-then" rules that describe the relationship between input variables and output actions. These rules are derived from expert knowledge or empirical data. For example:

- "If the temperature is high and the humidity is low, then increase the fan speed."
- "If the car is near the obstacle, then decelerate."

3. Inference Engine: Processes the fuzzy input using the fuzzy rule base. The inference engine evaluates which rules apply and how much influence each rule has on the system's behavior, combining them to form fuzzy output sets.
4. Defuzzification Interface: Converts the fuzzy output sets back into precise numerical values (crisp outputs). The system uses these outputs to control a process. Various defuzzification methods can be used, such as the centroid method or the max-membership principle.

How It Works:

1. Input Variables: The controller receives inputs from sensors (e.g., temperature, pressure, speed). These inputs are usually real-world, continuous values.
2. Fuzzification: The inputs are fuzzified into linguistic variables such as "low," "medium," or "high," and assigned membership values.
3. Rule Evaluation: The fuzzy rules are applied based on the fuzzified inputs. For example, multiple rules might apply simultaneously with varying degrees of truth.
4. Aggregation and Output: The outputs from the fuzzy rule evaluations are aggregated into a single fuzzy output.
5. Defuzzification: The fuzzy output is defuzzified into a specific, actionable value that is sent to the system (e.g., motor speed, temperature adjustment).

Example:

In a washing machine, a fuzzy logic controller might receive inputs such as the weight of the laundry, the dirtiness level, and the water temperature. It then uses fuzzy rules to determine how much water to use, how long to wash, and the optimal agitation speed. This allows the washing machine to adjust its settings dynamically, even with vague input data.

Advantages of Fuzzy Logic Controllers:

- No Need for Precise Mathematical Models: Unlike traditional control systems, FLCs don't require precise system modeling, which is beneficial for systems with non-linear dynamics or uncertain parameters.
- Handles Uncertainty and Vagueness: FLCs are ideal for environments with uncertainty, vagueness, or where human-like reasoning is preferred.
- Simplicity in Complex Systems: FLCs can handle complex systems in a simple, human-readable format using intuitive "if-then" rules.

Applications:

- Industrial Process Control: Controlling processes like chemical plants, where parameters change unpredictably or are hard to model.
- Consumer Electronics: Used in air conditioners, refrigerators, and cameras for automated control based on ambiguous inputs.
- Robotics: Fuzzy logic controllers are used in robotic systems for navigation, path planning, and obstacle avoidance.
- Automotive Systems: In adaptive cruise control, anti-lock braking systems (ABS), and traction control, where the system must adjust to changing conditions.

By mimicking human reasoning and decision-making, fuzzy logic controllers are especially useful when precision isn't the priority, and systems need to adapt smoothly to complex or uncertain conditions.

Temperature Regulator using Fuzzy logic

A Fuzzy Logic Controller (FLC) for temperature regulation operates by simulating human-like decision-making to control the temperature in an environment, such as a room or industrial process, based on imprecise or uncertain data. Instead of working with strict numerical boundaries (like traditional controllers), an FLC interprets inputs (such as temperature readings) using linguistic terms like "cold," "warm," or "hot." It then adjusts the temperature control mechanism (e.g., heater, cooler, fan) accordingly.

The Process:
The fuzzy logic controller works through a series of steps:

1. **Input Acquisition (Sensing)**:

- Sensors measure the actual temperature in the system. In some cases, additional variables like humidity or external environmental factors can be considered.
- These inputs are crisp (precise) values, such as 24°C or 30°C.

2. **Fuzzification**:

- The crisp sensor inputs are converted into fuzzy values using membership functions. These functions map each input to linguistic variables such as "cold," "cool," "warm," and "hot."
- For example, a temperature of 24°C could partially belong to the fuzzy sets for "cool" (with a membership of 0.6) and "warm" (with a membership of 0.4).
- This step allows the FLC to handle imprecise or vague information.

Example of membership functions:

- If the temperature is below 20°C, it might be classified as "cold."
- If the temperature is between 20°C and 25°C, it might be "cool."
- If the temperature is between 25°C and 30°C, it might be "warm."
- If the temperature is above 30°C, it might be "hot."

3. **Rule Evaluation (Inference Engine)**:

- The fuzzy logic controller uses a set of if-then rules to decide what actions should be taken based on the fuzzy input values. These rules are created based on expert knowledge or practical experience.
- Each rule defines how the system should respond when the inputs are in certain fuzzy states. For example:
  - Rule 1: If the temperature is "cold," then increase the heater.
  - Rule 2: If the temperature is "warm," then reduce the heating or turn on the cooling system.
  - Rule 3: If the temperature is "hot," then turn off the heater and increase the cooling.
- The inference engine evaluates all applicable rules and combines their outputs to generate a fuzzy control output.

Example: If the temperature is both "cool" and "warm" (to different degrees), the controller will consider both rules associated with these fuzzy states and apply a weighted combination of their outputs.

4. **Aggregation of Rule Outputs**:
- Once the rules are evaluated, the resulting fuzzy outputs are aggregated into a single fuzzy output. For example, if multiple rules are triggered, the controller combines their suggested actions to form a composite output.

5. **Defuzzification**:
- The fuzzy output (which could be a fuzzy variable like "increase heating slightly") is converted back into a crisp (precise) output value. This is done using defuzzification techniques like the centroid method, where the controller calculates the center of gravity of the fuzzy output set to determine the precise control action.
- This crisp output might be a value such as "increase the heater by 30%" or "reduce the cooling by 20%."

6. **Output Action**:
- The controller sends the crisp output as a control signal to the system's actuators (e.g., heating or cooling devices). This action adjusts the temperature by turning the heater, cooler, or fan on or off, or by varying their intensity.
- The system continuously monitors and adjusts based on updated sensor inputs.

Example Scenario:
Imagine a room with a fuzzy logic controller regulating the temperature. The system's goal is to maintain a temperature of around 25°C.

1. Input: The current temperature sensor reads 22°C.
2. Fuzzification: The temperature is fuzzified and mapped as "cool" with a membership value of 0.7 and "cold" with a membership value of 0.3.
3. Rule Evaluation:
- Rule 1: "If temperature is cool, then increase the heater slightly."
- Rule 2: "If temperature is cold, then increase the heater more."
- Both rules are partially activated due to the fuzzified input.
4. Aggregation: The outputs of both rules are combined, suggesting a moderate increase in heating.
5. Defuzzification: The combined fuzzy output is converted into a crisp output, which might instruct the heater to increase by 40%.
6. Output: The heater increases its power by 40%, raising the room's temperature.

Advantages of Fuzzy Logic in Temperature Regulation:
- Smooth and Adaptive Control: Unlike traditional controllers, FLCs avoid abrupt changes and overshooting by providing smooth, gradual responses to temperature changes.
- No Need for Precise Models: Fuzzy logic does not require an exact mathematical model of the system, making it ideal for non-linear and complex systems.
- Handling Uncertainty: FLCs can handle vague, incomplete, or uncertain information, making them well-suited for real-world environments where sensor data may be noisy or imprecise.

Python Code

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl

# Updated array which contains the room temperature (0 to 60°C)
temperature = np.arange(0, 61)

# Set the input
temp = ctrl.Antecedent(temperature, 'temp')

# Updated array to represent the possible AC temperatures (15 to 45°C)
ac_temperature = np.arange(15, 46)

# Set the output of the system
ac_temp = ctrl.Consequent(ac_temperature, 'ac_temp')

# Automatically generate the membership functions
temp.automf(5)
ac_temp.automf(5)

# Graphical representation of membership functions
temp.view()
ac_temp.view()

# Setting rules
rule1 = ctrl.Rule(temp['poor'], ac_temp['good'])
rule2 = ctrl.Rule(temp['average'], ac_temp['average'])
rule3 = ctrl.Rule(temp['good'], ac_temp['poor'])
rule4 = ctrl.Rule(temp['mediocre'], ac_temp['decent'])
rule5 = ctrl.Rule(temp['decent'], ac_temp['mediocre'])
```

```
# Creating control system
temperature_ctrl = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5])

detect_temp = ctrl.ControlSystemSimulation(temperature_ctrl)

# Testing for room temperature 60°C
detect_temp.input['temp'] = 60
detect_temp.compute()
temp_output = detect_temp.output['ac_temp']
print("\nWhen room temperature is 60°C:")
print("The AC temperature is adjusted to", temp_output, "°C")
ac_temp.view(sim=detect_temp)

# Testing for room temperature 30°C
detect_temp.input['temp'] = 41
detect_temp.compute()
temp_output = detect_temp.output['ac_temp']
print("\nWhen room temperature is 30°C:")
print("The AC temperature is adjusted to", temp_output, "°C")
ac_temp.view(sim=detect_temp)

# Testing for room temperature 0°C
detect_temp.input['temp'] = 39
detect_temp.compute()
temp_output = detect_temp.output['ac_temp']
print("\nWhen room temperature is 17°C:")
print("The AC temperature is adjusted to", temp_output, "°C")
ac_temp.view(sim=detect_temp)
```

Code Explanation

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
```

- **import numpy as np**: Imports the NumPy library and assigns it the alias np. NumPy is used for numerical operations, like creating arrays.
- **import skfuzzy as fuzz**: Imports the skfuzzy module from the scikit-fuzzy library for fuzzy logic operations and assigns it the alias fuzz.
- **from skfuzzy import control as ctrl**: Imports the control submodule from skfuzzy to work with fuzzy control systems and assigns it the alias ctrl.

```
# Updated array which contains the room temperature (0 to 60°C)
temperature = np.arange(0, 61)
```

- **temperature = np.arange(0, 61)**: Creates a NumPy array of integers from 0 to 60. This array represents possible room temperatures in degrees Celsius that the system will work with.

```
# Set the input
temp = ctrl.Antecedent(temperature, 'temp')
```

- **temp = ctrl.Antecedent(temperature, 'temp')**:
- Defines the input fuzzy variable called temp (short for temperature) using the array temperature.
- Antecedent means it is the input to the fuzzy system, where each value in the temperature array is fuzzyfied (mapped to fuzzy sets).
- 'temp' is the label used to represent the input variable.

```
# Updated array to represent the possible AC temperatures (15 to 45°C)
ac_temperature = np.arange(15, 46)
```

- **ac_temperature = np.arange(15, 46)**: Creates a NumPy array representing possible air conditioning (AC) temperature outputs, ranging from 15°C to 45°C. These are the temperatures that the AC can be set to based on the input room temperature.

```
# Set the output of the system
ac_temp = ctrl.Consequent(ac_temperature, 'ac_temp')
```

- **ac_temp = ctrl.Consequent(ac_temperature, 'ac_temp')**:
- Defines the output fuzzy variable called ac_temp (for AC temperature) using the ac_temperature array.
- Consequent means it is the output of the fuzzy system that will be computed based on the input temperature and the fuzzy rules.
- 'ac_temp' is the label used to represent this output variable.

```
# Automatically generate the membership functions
temp.automf(5)
ac_temp.automf(5)
```

- **temp.automf(5)**: Automatically generates 5 membership functions (fuzzy sets) for the input temp variable. These functions divide the input space into five linguistic categories such as 'poor', 'mediocre', 'average', 'decent', and 'good'.
- **ac_temp.automf(5)**: Similarly, automatically generates 5 membership functions for the output variable ac_temp. The categories represent different possible states for AC temperatures.

```
# Graphical representation of membership functions
temp.view()
ac_temp.view()
```

- **temp.view()**: Displays the membership functions for the input temp (room temperature) variable graphically.
- **ac_temp.view()**: Displays the membership functions for the output ac_temp (AC temperature) variable graphically.

```
# Setting rules
rule1 = ctrl.Rule(temp['poor'], ac_temp['good'])
rule2 = ctrl.Rule(temp['average'], ac_temp['average'])
rule3 = ctrl.Rule(temp['good'], ac_temp['poor'])
rule4 = ctrl.Rule(temp['mediocre'], ac_temp['decent'])
rule5 = ctrl.Rule(temp['decent'], ac_temp['mediocre'])
```

- **rule1 = ctrl.Rule(temp['poor'], ac_temp['good'])**:
  Defines a fuzzy rule. This rule says that if the room temperature is poor (low), the AC temperature should be good (high).
- **rule2 = ctrl.Rule(temp['average'], ac_temp['average'])**:
  If the room temperature is average, the AC temperature should also be average.
- **rule3 = ctrl.Rule(temp['good'], ac_temp['poor'])**:
  If the room temperature is good (high), the AC temperature should be poor (low).
- **rule4 = ctrl.Rule(temp['mediocre'], ac_temp['decent'])**:
  If the room temperature is mediocre, the AC temperature should be decent.
- **rule5 = ctrl.Rule(temp['decent'], ac_temp['mediocre'])**:
  If the room temperature is decent, the AC temperature should be mediocre.

```
# Creating control system
temperature_ctrl = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5])
```

- **temperature_ctrl = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5])**:
  Creates a fuzzy control system based on the five rules we defined.
  The control system will use these rules to adjust the output (AC temperature) based on the input (room temperature).

```
detect_temp = ctrl.ControlSystemSimulation(temperature_ctrl)
```

- **detect_temp = ctrl.ControlSystemSimulation(temperature_ctrl)**:
  Creates a simulation environment for the fuzzy control system temperature_ctrl.
  The detect_temp object is where we can input room temperatures and get the corresponding AC temperature outputs.

```
# Testing for room temperature 60°C
detect_temp.input['temp'] = 60
detect_temp.compute()
temp_output = detect_temp.output['ac_temp']
print("\nWhen room temperature is 60°C:")
print("The AC temperature is adjusted to", temp_output, "°C")
ac_temp.view(sim=detect_temp)
```

- **detect_temp.input['temp'] = 60**: Inputs a room temperature of 60°C into the fuzzy control system.
- **detect_temp.compute()**: Performs fuzzy computations using the input temperature (60°C) and the rules defined earlier.
- **temp_output = detect_temp.output['ac_temp']**: Retrieves the computed AC temperature from the fuzzy control system.
- **print()**: Prints the result to the console.
- **ac_temp.view(sim=detect_temp)**: Displays a graphical representation of the output membership function along with the computed AC temperature.

The same procedure is repeated for testing room temperatures of 30°C, 0°C, and 17°C in the following blocks.

Code Output

Calculated ac temperature to be set for different room temperature :

```
When room temperature is 60°C:
The AC temperature is adjusted to 17.52212389380531 °C

When room temperature is 30°C:
The AC temperature is adjusted to 24.80983302411873 °C

When room temperature is 17°C:
The AC temperature is adjusted to 25.66636851520572 °C
```

Figure 1:

The membership is assigned automatically to each value each ranging from
0 to 1 for poor temp. maximum membership at 0°C and least at 15°C similarly for mediocre category max membership at 15°C and least at 0°C and 30°C, for average max membership at 30°C least at 15°C and 45°C, for decent max membership at 45°C and least at 30°C and 60°C.
Membership Function graph for room temperature ranging from 0-60°C :

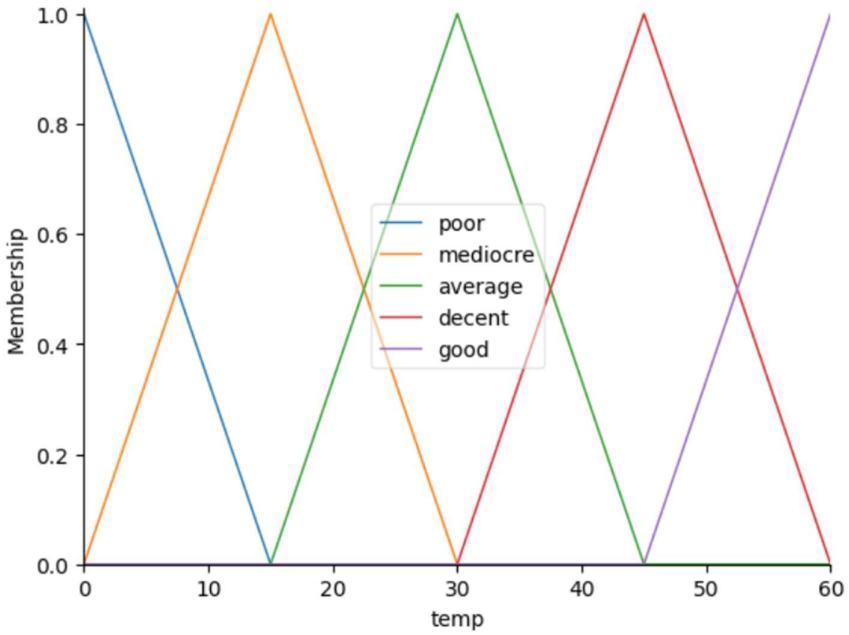

Figure 2:

The membership is assigned automatically to each value each ranging from
0 to 1 for poor temp. maximum membership at 15°C and least at 22.5°C similarly for mediocre category max membership at 22.5°C and least at 15°C and 30°C, for average max membership at 30°C least at 22.5°C and 37.5°C, for decent max membership at 37.5°C and least at 30°C and 45°C.
Membership Function graph for ac temperature ranging from 15-45°C:

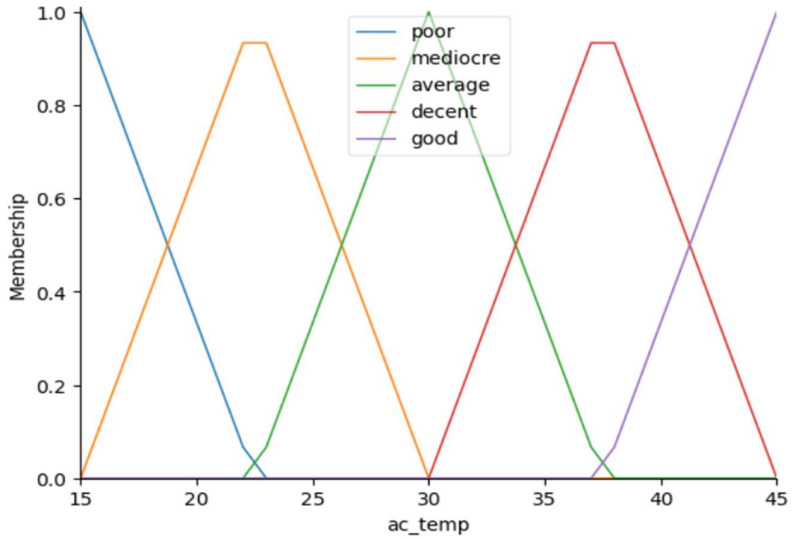

Figure 3:

After considering the set rules the control system comes with the value that has max membership related to the input value, the blue region here shows that the temperature to be set lies in poor category and has max membership 17.5°C.
The temperature at which ac is to be adjusted for 60°C room temperature i.e, around 17.5°C:

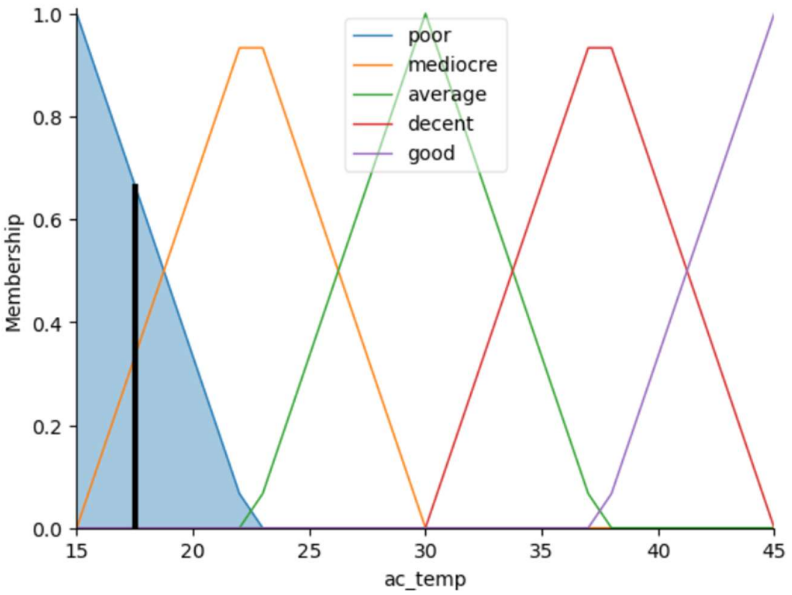

Figure 4:

After considering the set rules the control system comes with the value that has max membership related to the input value, the highlighted region here shows that the temperature to be set lies in mediocre category and has max membership at 24.8°C.
The temperature at which ac is to be adjusted for 30°C room temperature i.e, around 24.8°C:

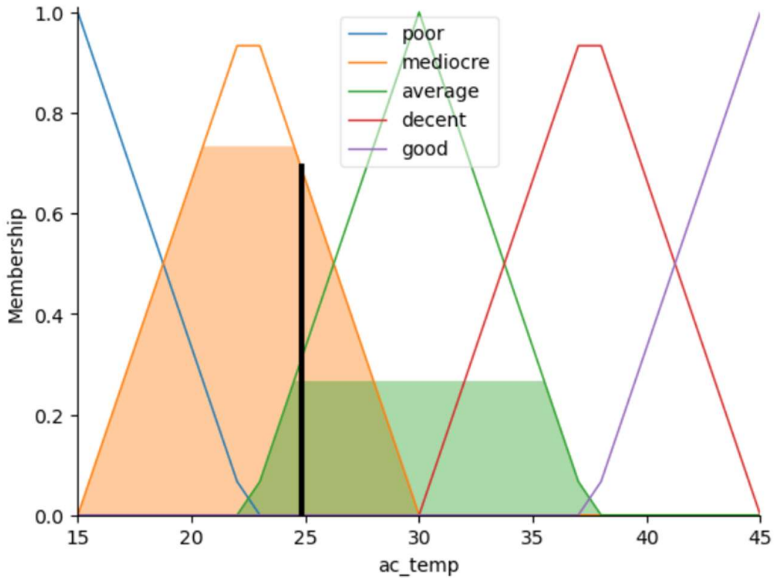

Figure 5:

After considering the set rules the control system comes with the value that has max membership related to the input value, the highlighted region here shows that the temperature to be set lies in mediocre category as well as average and has max membership at 25.6°C.
The temperature at which ac is to be adjusted for 17°C room temperature i.e, around 25.6°C:

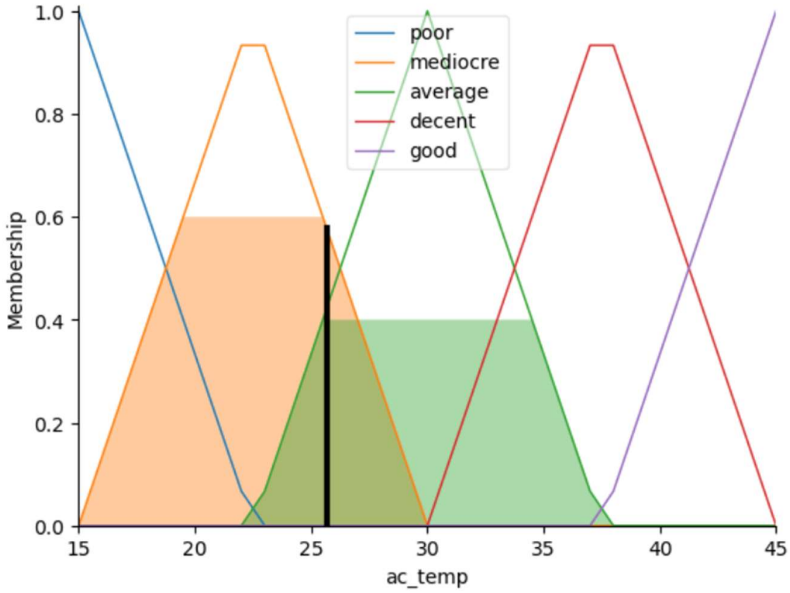

Figure 6:

Conclusion

The implementation of a Fuzzy Logic Controller (FLC) for temperature regulation offers a practical and effective solution for managing temperature in environments where traditional control methods face limitations. Unlike conventional controllers that rely on precise mathematical models and rigid control parameters, fuzzy logic controllers excel in handling uncertainties, nonlinear dynamics, and imprecise data. By mimicking human-like decision-making, the FLC can adapt to changing conditions, providing smoother transitions, reducing overshoot, and improving energy efficiency.

Through this project, we have demonstrated that a fuzzy logic controller can successfully regulate temperature by:

- Handling uncertain and vague data using fuzzification and linguistic variables, allowing for flexible control strategies.
- Employing simple "if-then" rules to guide decision-making, based on expert knowledge or empirical data.
- Providing adaptive, real-time responses to temperature fluctuations without the need for complex mathematical models.
- Achieving energy-efficient operation by optimizing heating and cooling actions in response to varying conditions.

The FLC's adaptability makes it highly suitable for a wide range of applications, from household HVAC systems to industrial processes, greenhouses, and smart building management. Its ability to deal with real-world complexities and uncertain inputs represents a significant advantage over traditional controllers, particularly in environments with dynamic conditions.

In conclusion, the fuzzy logic controller for temperature regulation proves to be a versatile, efficient, and robust solution for modern control systems, contributing to both improved performance and user comfort in temperature-sensitive environments. This approach opens up further possibilities for integrating fuzzy logic into other control systems where flexibility and adaptability are crucial.