NAME	**Shreya Shetty**
UID	**2019140059**
CLASS	**TE IT**
BATCH	**B**
ACADEMIC YEAR	**2021-22**
SUBJECT	**SC (Soft Computing)**
COURSE CODE	**IT312**
EXPERIMENT NO.	**9**

Aim:

To design and implement Fuzzy Controller for a problem of your choice

Theory:

Fuzzy Logic Control

Fuzzy logic control (FLC) is the most active research area in the application of fuzzy sec theory, fuzzy reasoning and fuzzy logic. The application of FLC extends from industrial process control to biomedical instrumentation and securities. Compared to conventional control techniques, FLC has been best utilized in complex ill-defined problems, which can be controlled by efficient human operator without knowledge of their underlying dynamics.

Why Use Fuzzy Logic in Control Systems

- A control system is an arrangement of physical components designed to alter another physical system so that this system exhibits certain desired characteristics.
- There exist two types of control systems: open-loop and closed-loop control systems.
- In open-loop control systems, the input control action is independent of the physical system output.
- On the other hand, in closed-loop control system, the input control action depends on the physical system output.
- Closed-loop control systems are also known as feedback control systems.
- The first step toward controlling any physical variable is to measure it. A sensor measures the controlled signal.
- A plant is the physical system under control. In a closed-loop control system, forcing signals of the system - called inputs -- are determined by the output responses of the system.

- The basic control problem is given as follows: The output of the physical system under control is adjusted by the help of error signal. The difference between the actual response (calculated) of the plant and the desired response gives the error signal.
- For obtaining satisfactory responses and characteristics for the closed-loop control system, an additional system, called as compensator or controller, can be added to the loop. The basic block diagram of closed-loop control system is shown in below Figure.

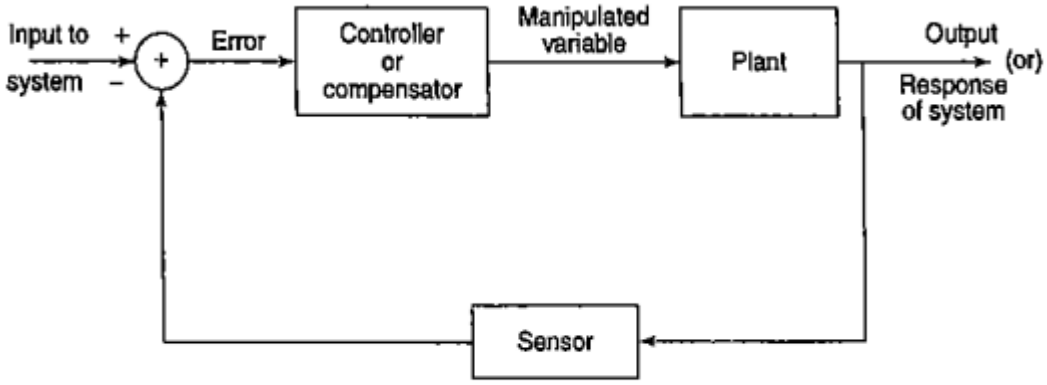

Figure 14-1 Block diagram of a closed-loop control system.

- The basic concept behind FLC is to utilize the expert knowledge and experience of a human operator for designing a controller for controlling an application process whose input-output relationship is given by a collection of fuzzy control rules using linguistic variables instead of a complicated dynamic model.
- The fuzzy control rules are basically IF-THEN rules. The linguistic variables, fuzzy control rules and fuzzy appropriate reasoning are best utilized for designing the controller.

Assumptions in Fuzzy Logic Control (FLC) Design

- The plant is observable and controllable − It must be assumed that the input, output as well as state variables are available for observation and controlling purpose.
- Existence of a knowledge body − It must be assumed that there exist a knowledge body having linguistic rules and a set of input-output data set from which rules can be extracted.
- Existence of solution − It must be assumed that there exists a solution.
- 'Good enough' solution is enough − The control engineering must look for 'good enough' solution rather than an optimum one.
- Range of precision − Fuzzy logic controller must be designed within an acceptable range of precision.

- Issues regarding stability and optimality − The issues of stability and optimality must be open in designing Fuzzy logic controller rather than addressed explicitly.

Architecture of Fuzzy Logic Control

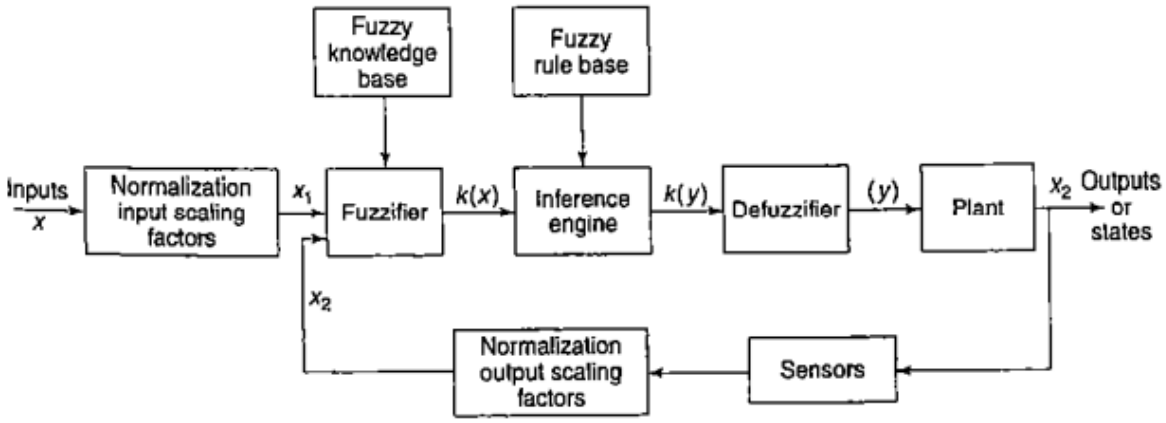

Figure 14-2 Basic architecture of an FLC system.

The basic architecture of a fuzzy logic controller is shown in the above Figure.

- The principal components of an FLC system are: a fuzzifier, a fuzzy rule base, a fuzzy knowledge base, an inference engine and a defuzzifier.
- It also includes parameters for normalization.
- When the output from the defuzzifier is not a control action for a plant, then the system is a fuzzy logic decision system.
- The fuzzifier present converts the crisp quantities into fuzzy quantities. The fuzzy rule base scores the knowledge about the operation of the process of domain expertise.
- The fuzzy knowledge base stores the knowledge about all the input-output fuzzy relationships. It includes the membership functions defining the input variables to the fuzzy rule base and the output variables to the plant under control.
- The inference engine is the kernel of an FLC system, and it possess the capability to simulate human decisions by performing approximate reasoning to achieve a desired control strategy.
- The defuzzifier converts the fuzzy quantities into crisp quantities from an inferred fuzzy control action by the inference engine.

Major Components of FLC

- **Fuzzifier** − The role of fuzzifier is to convert the crisp input values into fuzzy values.
- **Fuzzy Knowledge Base** − It stores the knowledge about all the input-output fuzzy relationships. It also has the membership function which defines the input variables to the fuzzy rule base and the output variables to the plant under control.
- **Fuzzy Rule Base** − It stores the knowledge about the operation of the process of domain.
- **Inference Engine** − It acts as a kernel of any FLC. Basically it simulates human decisions by performing approximate reasoning.
- **Defuzzifier** − The role of defuzzifier is to convert the fuzzy values into crisp values getting from fuzzy inference engine.

The various steps involved in designing a fuzzy logic controller are as follows:

Step 1: Locate the input, output and slate variables of the plane under consideration.

Step 2: Split the complete universe of discourse spanned by each variable into a number of fuzzy subsets, assigning each with a linguistic label. The subsets include all the elements in the universe.

Step 3: Obtain the membership function for each fuzzy subset.

Step 4: Assign the fuzzy relationships between the inputs or states of fuzzy subsets on one side and the outputs of fuzzy subsets on other side, thereby forming the rule base.

Step 5: Choose appropriate scaling factors for the input and output variables for normalizing the variables between [0, 1] and [-1, I] interval.

Step 6: Carry out the fuzzification process.

Step 7: Identify the output contributed from each rule using fuzzy approximate reasoning.

Step 8: Combine the fuzzy outputs obtained from each rule.

Step 9: Finally, apply defuzzification to form a crisp output.

The above steps are performed and executed for a simple FLC system. The following design elements are adopted for designing a general FLC system:

1. Fuzzification strategies and the interpretation of a fuzzifier.
2. Fuzzy knowledge base:

a. normalization of the parameters involved;
b. partitioning of input and output spaces;
c. selection of membership functions of a primary fuzzy set.

3. Fuzzy rule base:
   a. selection of input and output variables;
   b. source from which fuzzy control rules are w be derived;
   c. types of fuzzy control rules;
   d. completeness of fuzzy control rules.
4. Decision· making logic:
   a. proper definition of fuzzy implication;
   b. interpretation of connective "and";
   c. interpretation of connective "or";
   d. inference engine.
5. Defuzzification strategies and the interpretation of a defuzzifier.

When all the above five design parameters are fixed, the FLC system is simple. Based on all this, the features of a simple FLC system are as follows:

fixed and uniform input and output scaling factors for normalization; fixed and noninteractive rules; fixed membership functions; only limited number of rules, which increases exponentially with the number of input variables;

Code:

```
import sys

waitingTrafficFuzzySet = ['minimal', 'light', 'average', 'heavy', 'standstill']
oncomingTrafficFuzzySet = ['minimal', 'light', 'average', 'heavy', 'excess']

durationFuzzySet = ['short', 'medium', 'long']

fuzzyRules = [[['minimal', 'minimal'], 'short'], [['minimal', 'light'], 'short'],
[['minimal', 'average'], 'medium'],
                    [['minimal', 'heavy'], 'long'], [['minimal', 'excess'],
'long'],

                    [['light', 'minimal'], 'short'], [['light', 'light'], 'short'],
[['light', 'average'], 'medium'],
                    [['light', 'heavy'], 'medium'], [['light', 'excess'], 'long'],

                    [['average', 'minimal'], 'short'], [['average', 'light'],
'medium'], [['average', 'average'], 'medium'],
```

```
                    [['average', 'heavy'], 'long'], [['average', 'excess'],
'long'],

                    [['heavy', 'minimal'], 'medium'], [['heavy', 'light'],
'medium'], [['heavy', 'average'], 'long'],
                    [['heavy', 'heavy'], 'long'], [['heavy', 'excess'], 'long'],

                    [['standstill', 'minimal'], 'medium'], [['standstill',
'light'], 'long'], [['standstill', 'average'], 'long'],
                    [['standstill', 'heavy'], 'long'], [['standstill', 'excess'],
'long']]


def carWaitingFunction(cars):
    linguisticCarsWaiting = []

    if cars >= 0 and cars <= 15:
        linguisticCarsWaiting.append(waitingTrafficFuzzySet[0]) # Minimal load of
cars waiting
    if cars >= 10 and cars <= 25:
        linguisticCarsWaiting.append(waitingTrafficFuzzySet[1]) # Light load of cars
waiting
    if cars >= 20 and cars <= 35:
        linguisticCarsWaiting.append(waitingTrafficFuzzySet[2]) # Average load of
cars waiting
    if cars >= 30 and cars <= 45:
        linguisticCarsWaiting.append(waitingTrafficFuzzySet[3]) # Heavy load of cars
waiting
    if cars >= 40:
        linguisticCarsWaiting.append(waitingTrafficFuzzySet[4]) # Standstill load of
cars waiting


    #Determine the overall wait time
    valueOfCarsWaiting = []

    if len(linguisticCarsWaiting) > 1:
        if linguisticCarsWaiting[0] == waitingTrafficFuzzySet[0] and
linguisticCarsWaiting[1] == waitingTrafficFuzzySet[1]:
            #Minimal and Light traffic wating (10 : 15)
            minimal = - (cars - 15) / (15 - 10)
            valueOfCarsWaiting.append([linguisticCarsWaiting[0], minimal])

            light = - (cars - 25) / (15 - 10)
            valueOfCarsWaiting.append([linguisticCarsWaiting[1], light])

            print('Fuzzy Set Minimal & Light: ', valueOfCarsWaiting)

            return valueOfCarsWaiting

        elif linguisticCarsWaiting[0] == waitingTrafficFuzzySet[1] and
linguisticCarsWaiting[1] == waitingTrafficFuzzySet[2]:
            #Light and Average traffic waiting
```

```
            light = - (cars - 25) / (25 - 20)
            valueOfCarsWaiting.append([linguisticCarsWaiting[0], light])

            average = (cars - 20) / (25 - 20)
            valueOfCarsWaiting.append([linguisticCarsWaiting[1], average])

            print('Fuzzy Set Light & Average: ', valueOfCarsWaiting)

            return valueOfCarsWaiting

        elif linguisticCarsWaiting[0] == waitingTrafficFuzzySet[2] and
linguisticCarsWaiting[1] == waitingTrafficFuzzySet[3]:
            #Average and Heavy traffic waiting
            average = - (cars - 35) / (35 - 30)
            valueOfCarsWaiting.append([linguisticCarsWaiting[0], average])

            heavy = (cars - 30) / (35 - 30)
            valueOfCarsWaiting.append([linguisticCarsWaiting[1], heavy])

            print('Fuzzy Set Average & Heavy: ', valueOfCarsWaiting)

            return valueOfCarsWaiting

        elif linguisticCarsWaiting[0] == waitingTrafficFuzzySet[3] and
linguisticCarsWaiting[1] == waitingTrafficFuzzySet[4]:
            #Average and Heavy traffic waiting
            heavy = - (cars - 45) / (45 - 40)
            valueOfCarsWaiting.append([linguisticCarsWaiting[0], heavy])

            standstill = (cars - 40) / (45 - 40)
            valueOfCarsWaiting.append([linguisticCarsWaiting[1], standstill])

            print('Fuzzy Set Heavy & Standstill: ', valueOfCarsWaiting)

            return valueOfCarsWaiting
        else:
            return valueOfCarsWaiting.append([linguisticCarsWaiting[0],1])



def carIncomingFunction(cars):
    linguisticCarsIncoming = []

    if cars >= 0 and cars <= 15:
        linguisticCarsIncoming.append(oncomingTrafficFuzzySet[0]) # Minimal load of
cars coming into the intersection
    if cars >= 10 and cars <= 25:
        linguisticCarsIncoming.append(oncomingTrafficFuzzySet[1]) # Light load of
cars coming into the intersection
    if cars >= 20 and cars <= 35:
        linguisticCarsIncoming.append(oncomingTrafficFuzzySet[2]) # Average load of
cars coming into the intersection
    if cars >= 30 and cars <= 45:
```

```
        linguisticCarsIncoming.append(oncomingTrafficFuzzySet[3]) # Heavy load of
cars coming into the intersection
    if cars >= 40:
        linguisticCarsIncoming.append(oncomingTrafficFuzzySet[4]) # Standstill load
of cars coming into the intersection

    print('\nINCOMING CARS FUZZY: ', linguisticCarsIncoming, '\n')

    valueOfCarsIncoming = []

    if len(linguisticCarsIncoming) > 1:
        if linguisticCarsIncoming[0] == oncomingTrafficFuzzySet[0] and
linguisticCarsIncoming[1] == oncomingTrafficFuzzySet[1]:
            #Minimal and Light traffic wating (10 : 15)
            minimal = - (cars - 15) / (15 - 10)
            valueOfCarsIncoming.append([linguisticCarsIncoming[0], minimal])

            light = - (cars - 25) / (15 - 10)
            valueOfCarsIncoming.append([linguisticCarsIncoming[1], light])

            print('Fuzzy Set Minimal & Light: ', valueOfCarsIncoming)

            return valueOfCarsIncoming

        elif linguisticCarsIncoming[0] == oncomingTrafficFuzzySet[1] and
linguisticCarsIncoming[1] == oncomingTrafficFuzzySet[2]:
            #Light and Average traffic waiting
            light = - (cars - 25) / (25 - 20)
            valueOfCarsIncoming.append([linguisticCarsIncoming[0], light])

            average = (cars - 20) / (25 - 20)
            valueOfCarsIncoming.append([linguisticCarsIncoming[1], average])

            print('Fuzzy Set Light & Average: ', valueOfCarsIncoming)

            return valueOfCarsIncoming

        elif linguisticCarsIncoming[0] == oncomingTrafficFuzzySet[2] and
linguisticCarsIncoming[1] == oncomingTrafficFuzzySet[3]:
            #Average and Heavy traffic waiting
            average = - (cars - 35) / (35 - 30)
            valueOfCarsIncoming.append([linguisticCarsIncoming[0], average])

            heavy = (cars - 30) / (35 - 30)
            valueOfCarsIncoming.append([linguisticCarsIncoming[1], heavy])

            print('Fuzzy Set Average & Heavy: ', valueOfCarsIncoming)

            return valueOfCarsIncoming

        elif linguisticCarsIncoming[0] == oncomingTrafficFuzzySet[3] and
linguisticCarsIncoming[1] == oncomingTrafficFuzzySet[4]:
            #Average and Heavy traffic waiting
```

```
            heavy = - (cars - 45) / (45 - 40)
            valueOfCarsIncoming.append([linguisticCarsIncoming[0], heavy])

            excess = (cars - 40) / (45 - 40)
            valueOfCarsIncoming.append([linguisticCarsIncoming[1], excess])

            print('Fuzzy Set Heavy & Excess: ', valueOfCarsIncoming)

            return valueOfCarsIncoming

        else:
            return valueOfCarsIncoming.append([linguisticCarsIncoming[0],1])


def check(x):
    for i in x:
        if x[0] != True:
            return False
    return True


def infer(waiting, oncoming, rules):
    A = []
    P = []

    for i in waiting:
       A.append(i)
    for i in oncoming:
        A.append(i)
    # print("A",A)
    while A:
        x = A.pop(0)
        for rule in rules:
            # print("rule:",rule)
            for j, k in enumerate(rule[0]):
                # print(j,k,x[0])
                if k == x[0]:
                    rule[0][j] = [True, rule[0][j], x[1]]
                    # print('yes1')
            if check(rule[0]):
            # if 1<2:
                # print("yes2")
                result = rule[1]
                P.append(rule)
                print("P",P)
                A.append(result)
                rule[0] = [rule[0],'processed']

    return P



def defuzzyfication(inputs):
```

```
    fuzzyValues = []
    currentFuzzyLinguistics = 'short'
    currentFuzzyLinguistics2 = 'short'
    currentFuzzyValue = 1.0
    fuzzyDifference = 0

    for i in inputs:
        for j in i:
            if j == 'short':
                if i[1] < currentFuzzyValue:
                    currentFuzzyValue = i[1]
            if j == 'medium':
                if i[1] < currentFuzzyValue:
                    currentFuzzyValue = i[1]
                    currentFuzzyLinguistics = j
            if j == 'long':
                if i[1] < currentFuzzyValue:
                    currentFuzzyValue = i[1]
                    currentFuzzyLinguistics = j
    if currentFuzzyLinguistics == 'medium':
        currentFuzzyLinguistics2 = 'short'
        fuzzyDifference = 1.0 - currentFuzzyValue
    if currentFuzzyLinguistics == 'long':
        currentFuzzyLinguistics2 = 'medium'
        fuzzyDifference = 1.0 - currentFuzzyValue
    else:
        fuzzyDifference = 1.0 - currentFuzzyValue


    fuzzyValues.append(currentFuzzyLinguistics)
    fuzzyValues.append(currentFuzzyValue)
    fuzzyValues.append(currentFuzzyLinguistics2)
    fuzzyValues.append(fuzzyDifference)

    return fuzzyValues


# Finding time
def secs(values):
    seconds = 0
    shortX = 50
    mediumX = 90
    longX = 125

    condition1 = values[0]
    value1 = values[1]
    condition2 = values[2]
    value2 = values[3]

    if condition1 == 'medium' and condition2 == 'short':
        seconds = (value1 * mediumX + value2 * shortX)
    if condition1 == 'medium' and condition2 == 'long':
        seconds = (value1 * mediumX + value2 * longX)
```

```
    if condition1 == 'long' and condition2 == 'medium':
        seconds = (value1 * longX + value2 * mediumX)
    if condition1 == 'short' and condition2 == 'medium':
        seconds = (value1 * shortX + value2 * mediumX)
    if condition1 == 'long':
        seconds = 150
    else:
        seconds = 45
    return seconds


def main():
    carsWaiting = 10
    carsIncoming = 20
    # carsWaiting = 10
    # carsIncoming = 20
    print('No of cars waiting: ', carsWaiting)
    print('No of cars incoming: ', carsIncoming)
    print('')
    #Returns a two dimensional array in the form [['Light', 1.0], ['Average', 0.0]]
    waitingTraffic = carWaitingFunction(carsWaiting)
    incomingTraffic = carIncomingFunction(carsIncoming)

    print('\nCARS WAITING: ', waitingTraffic, '   CARS INCOMING: ', incomingTraffic)


    possibleWait = infer(waitingTraffic, incomingTraffic, fuzzyRules)
    result = []
    resultMax = {}

    for i in possibleWait:
        print(i[0][0][0][1], i[0][0][0][2], [0][0][1][1], i[0][0][1][2], i[1])
        minimum = min(i[0][0][0][2], i[0][0][1][2])
        result.append([i[1],minimum])

    for i in resultMax:
        if i[0] in resultMax:
            resultMax[i[0]].add(i[1])
        else:
            resultMax[i[0]] = set([i[1]])

    inference = []
    for key, value in resultMax.items():
        inference.append([key,max(value)])
    # print('\nPOSSIBLE:',possibleWait,result,resultMax,inference)
    print('\nDeffuzzyfication')
    finalValue = defuzzyfication(inference)
    print('\nFUZZY SET: ', finalValue)
    seconds = secs(finalValue)
    print('\nGreen lights will be activated for ', seconds, ' seconds!')

if __name__ == '__main__':
    main()
```

Output:

```
No of cars waiting:  10
No of cars incoming:  20

Fuzzy Set Minimal & Light:  [['minimal', 1.0], ['light', 3.0]]

INCOMING CARS FUZZY:  ['light', 'average']

Fuzzy Set Light & Average:  [['light', 1.0], ['average', 0.0]]

CARS WAITING:  [['minimal', 1.0], ['light', 3.0]]    CARS INCOMING:  [['light', 1.0], ['average', 0.0]]

Deffuzzyfication

FUZZY SET:  ['short', 1.0, 'short', 0.0]

Green lights will be activated for  45  seconds!
```

Conclusion:

In this experiment, I have implemented Fuzzy controller for the traffic system. I learnt the working of fuzzy inference system from fuzzification of input, checking the rule base and defuzzification.