BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai – 400058-India
Department of Computer Engineering

Name	Rishabh Santosh Shenoy
UID no.	2023300222
Experiment No.	9

AIM:	To design and implement fuzzy logic controller for brake system
Program 1	
PROBLEM STATEMENT :	AMK37 Problem Design a fuzzy controller for a train approach or leaving a station. The i/p are distance from a station & speed of the train. The o/p is break power used. Use a) triangular membership function b) four descriptor for each variable c) five to six rules d) Approximate defuzzification method.

PROGRAM:

```
import numpy as np

def triangular(x, a, b, c):
    if x <= a or x >= c:
        return 0.0
    if a < x <= b:
        return (x - a) / (b - a) if (b - a) != 0 else 1.0
    return (c - x) / (c - b) if (c - b) != 0 else 1.0

def mu_VSD(x): return triangular(x, 0, 0, 100)
def mu_SD(x):  return triangular(x, 0, 100, 400)
def mu_LD(x):  return triangular(x, 100, 400, 500)
def mu_VLD(x): return triangular(x, 400, 500, 500)

def mu_VLS(y): return triangular(y, 0, 0, 10)
def mu_LS(y):  return triangular(y, 0, 10, 50)
def mu_HS(y):  return triangular(y, 10, 50, 60)
def mu_VHS(y): return triangular(y, 50, 60, 60)
```

```
bp_params = {
    'VLP': (0, 0, 20),
    'LP':  (0, 20, 80),
    'HP':  (20, 80, 100),
    'VHP': (80, 100, 100)
}

def inverse_z_for_label(label, mu):
    if mu <= 0:
        return None
    a,b,c = bp_params[label]
    if (b - a) != 0:
        return a + mu * (b - a)
    if (c - b) != 0:
        return c - mu * (c - b)
    return None

rule_base = {
    ('VSD', 'VLS'): 'HP', ('VSD', 'LS'): 'HP', ('VSD', 'HS'): 'VHP', 
('VSD','VHS'): 'VHP',
    ('SD',  'VLS'): 'LP', ('SD',  'LS'): 'LP', ('SD',  'HS'): 'HP',  ('SD','VHS'): 
'VHP',
    ('LD',  'VLS'): 'VLP',('LD',  'LS'): 'VLP',('LD',  'HS'): 'LP',  ('LD','VHS'): 
'HP',
    ('VLD', 'VLS'): 'VLP',('VLD', 'LS'): 'VLP',('VLD', 'HS'): 'LP', 
('VLD','VHS'): 'LP'
}

distance = 110
speed = 52

dist_vals = {
    'VSD': mu_VSD(distance),
    'SD': mu_SD(distance),
    'LD': mu_LD(distance),
    'VLD': mu_VLD(distance)
}
speed_vals = {
```

```
    'VLS': mu_VLS(speed),
    'LS': mu_LS(speed),
    'HS': mu_HS(speed),
    'VHS': mu_VHS(speed)
}

print("Fuzzification (distance & speed):")
for k,v in dist_vals.items(): print(f"  μ_{k}({distance}) = {v:.6f}")
for k,v in speed_vals.items(): print(f"  μ_{k}({speed}) = {v:.6f}")

fired = []
for dlab, dmu in dist_vals.items():
    for slab, smu in speed_vals.items():
        if (dlab, slab) in rule_base:
            out = rule_base[(dlab, slab)]
            strength = min(dmu, smu)
            if strength > 0:
                fired.append((dlab, slab, out, strength))

print("\nFired rules (d_label, s_label -> output, strength):")
for item in fired:
    print(" ", item)

agg = {}
for _,_,out,st in fired:
    agg[out] = max(agg.get(out, 0.0), st)

print("\nAggregated strengths per output (max rule strength):")
for out, s in agg.items():
    print(f"  {out}: {s:.6f}")

rep = []
for out, s in agg.items():
    z = inverse_z_for_label(out, s)
    if z is not None:
        rep.append((out, s, z))

print("\nRepresentative z-values (rising-side inverse):")
```

	`` for out,s,z in rep: print(f" {out}: strength={s:.6f} => z_rep={z:.6f}") rep_sorted = sorted(rep, key=lambda x: x[1], reverse=True) top2 = rep_sorted[:2] print("\nTop 2 outputs by strength (for mean-of-max):") for out,s,z in top2: print(f" {out}: strength={s:.6f}, z_rep={z:.6f}") if len(top2) == 0: mean_of_max = None elif len(top2) == 1: mean_of_max = top2[0][2] else: mean_of_max = (top2[0][2] + top2[1][2]) / 2.0 print("\nFinal (mean-of-max) defuzzified brake power =", f"{mean_of_max:.6f} %" if mean_of_max is not None else "N/A") ```
PROGRAM WITH LIBRARY:	``` !pip install scikit-fuzzy import numpy as np import skfuzzy as fuzz from skfuzzy import control as ctrl distance = ctrl.Antecedent(np.arange(0, 501, 1), 'distance') speed = ctrl.Antecedent(np.arange(0, 61, 1), 'speed') brake = ctrl.Consequent(np.arange(0, 101, 1), 'brake') distance['VSD'] = fuzz.trimf(distance.universe, [0, 0, 100]) distance['SD'] = fuzz.trimf(distance.universe, [0, 100, 400]) distance['LD'] = fuzz.trimf(distance.universe, [100, 400, 500]) distance['VLD'] = fuzz.trimf(distance.universe, [400, 500, 500]) speed['VLS'] = fuzz.trimf(speed.universe, [0, 0, 10]) speed['LS'] = fuzz.trimf(speed.universe, [0, 10, 50]) speed['HS'] = fuzz.trimf(speed.universe, [10, 50, 60]) ```

	`` speed['VHS'] = fuzz.trimf(speed.universe, [50, 60, 60]) brake['VLP'] = fuzz.trimf(brake.universe, [0, 0, 20]) brake['LP'] = fuzz.trimf(brake.universe, [0, 20, 80]) brake['HP'] = fuzz.trimf(brake.universe, [20, 80, 100]) brake['VHP'] = fuzz.trimf(brake.universe, [80, 100, 100]) rule1 = ctrl.Rule(distance['VSD'] & speed['VLS'], brake['HP']) rule2 = ctrl.Rule(distance['VSD'] & speed['LS'], brake['HP']) rule3 = ctrl.Rule(distance['VSD'] & speed['HS'], brake['VHP']) rule4 = ctrl.Rule(distance['VSD'] & speed['VHS'], brake['VHP']) rule5 = ctrl.Rule(distance['SD'] & speed['VLS'], brake['LP']) rule6 = ctrl.Rule(distance['SD'] & speed['LS'], brake['LP']) rule7 = ctrl.Rule(distance['SD'] & speed['HS'], brake['HP']) rule8 = ctrl.Rule(distance['SD'] & speed['VHS'], brake['VHP']) rule9 = ctrl.Rule(distance['LD'] & speed['VLS'], brake['VLP']) rule10 = ctrl.Rule(distance['LD'] & speed['LS'], brake['VLP']) rule11 = ctrl.Rule(distance['LD'] & speed['HS'], brake['LP']) rule12 = ctrl.Rule(distance['LD'] & speed['VHS'], brake['HP']) rule13 = ctrl.Rule(distance['VLD'] & speed['VLS'], brake['VLP']) rule14 = ctrl.Rule(distance['VLD'] & speed['LS'], brake['VLP']) rule15 = ctrl.Rule(distance['VLD'] & speed['HS'], brake['LP']) rule16 = ctrl.Rule(distance['VLD'] & speed['VHS'], brake['LP']) brake_ctrl = ctrl.ControlSystem([rule1,rule2,rule3,rule4, rule5,rule6,rule7,rule8, rule9,rule10,rule11,rule12, rule13,rule14,rule15,rule16]) brake_sys = ctrl.ControlSystemSimulation(brake_ctrl) brake_sys.input['distance'] = 110 brake_sys.input['speed'] = 52 brake_sys.compute() ```

```
print("\nBrake Power Output =", round(brake_sys.output['brake'], 2), "%")

brake.view(sim=brake_sys)
```

RESULT:

```
psipl@psipl-OptiPlex-SFF-7010:~/Desktop/Rishabh_Shenoy$ /usr/bin/python3 /home/psipl/Desktop/Rishabh_Shenoy/Exp09/part2.py
Fuzzification (distance & speed):
  μ_VSD(110) = 0.000000
  μ_SD(110) = 0.966667
  μ_LD(110) = 0.033333
  μ_VLD(110) = 0.000000
  μ_VLS(52) = 0.000000
  μ_LS(52) = 0.000000
  μ_HS(52) = 0.800000
  μ_VHS(52) = 0.200000

Fired rules (d_label, s_label -> output, strength):
  ('SD', 'HS', 'HP', 0.8)
  ('SD', 'VHS', 'VHP', 0.2)
  ('LD', 'HS', 'LP', 0.03333333333333333)
  ('LD', 'VHS', 'HP', 0.03333333333333333)

Aggregated strengths per output (max rule strength):
  HP: 0.800000
  VHP: 0.200000
  LP: 0.033333

Representative z-values (rising-side inverse):
  HP: strength=0.800000 => z_rep=68.000000
```

```
Representative z-values (rising-side inverse):
  HP: strength=0.800000 => z_rep=68.000000
  VHP: strength=0.200000 => z_rep=84.000000
  LP: strength=0.033333 => z_rep=0.666667

Top 2 outputs by strength (for mean-of-max):
  HP: strength=0.800000, z_rep=68.000000
  VHP: strength=0.200000, z_rep=84.000000

Final (mean-of-max) defuzzified brake power = 76.000000 %
psipl@psipl-OptiPlex-SFF-7010:~/Desktop/Rishabh_Shenoy$ 
0
```

Program 2

PROBLEM STATEMENT :

AMK37

Problem Design a fuzzy controller for a train approach or leaving a station. The i/p are distance from a station & speed of the train. The o/p is break power used. Use
a) triangular membership function
b) Four descriptor for each variable
c) five to six rules
d) Approximate defuzzification method.

PROGRAM IN COLAB:

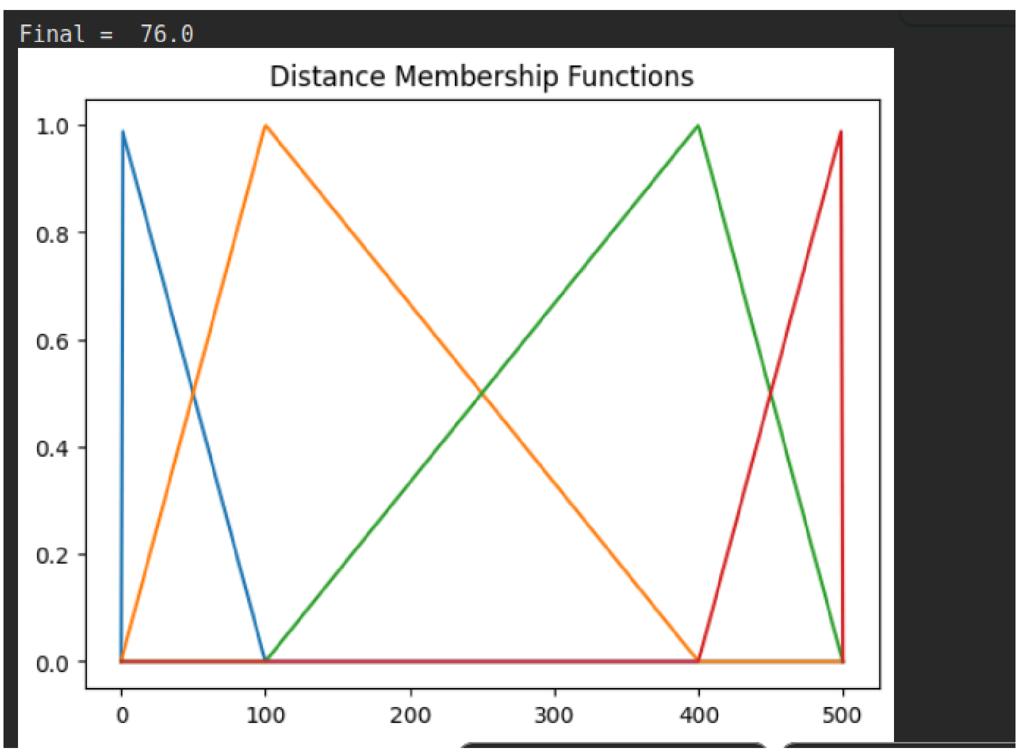
```
import numpy as np
import matplotlib.pyplot as plt

def triangular(x, a, b, c):
    if x <= a or x >= c:
        return 0.0
    if a < x <= b:
        return (x - a) / (b - a) if (b - a) != 0 else 1.0
    return (c - x) / (c - b) if (c - b) != 0 else 1.0

def mu_VSD(x): return triangular(x, 0, 0, 100)
def mu_SD(x):  return triangular(x, 0, 100, 400)
def mu_LD(x):  return triangular(x, 100, 400, 500)
def mu_VLD(x): return triangular(x, 400, 500, 500)

def mu_VLS(y): return triangular(y, 0, 0, 10)
def mu_LS(y):  return triangular(y, 0, 10, 50)
def mu_HS(y):  return triangular(y, 10, 50, 60)
def mu_VHS(y): return triangular(y, 50, 60, 60)

bp_params = {
    'VLP': (0, 0, 20),
    'LP':  (0, 20, 80),
    'HP':  (20, 80, 100),
    'VHP': (80, 100, 100)
}

def inverse_z_for_label(label, mu):
    if mu <= 0:
```

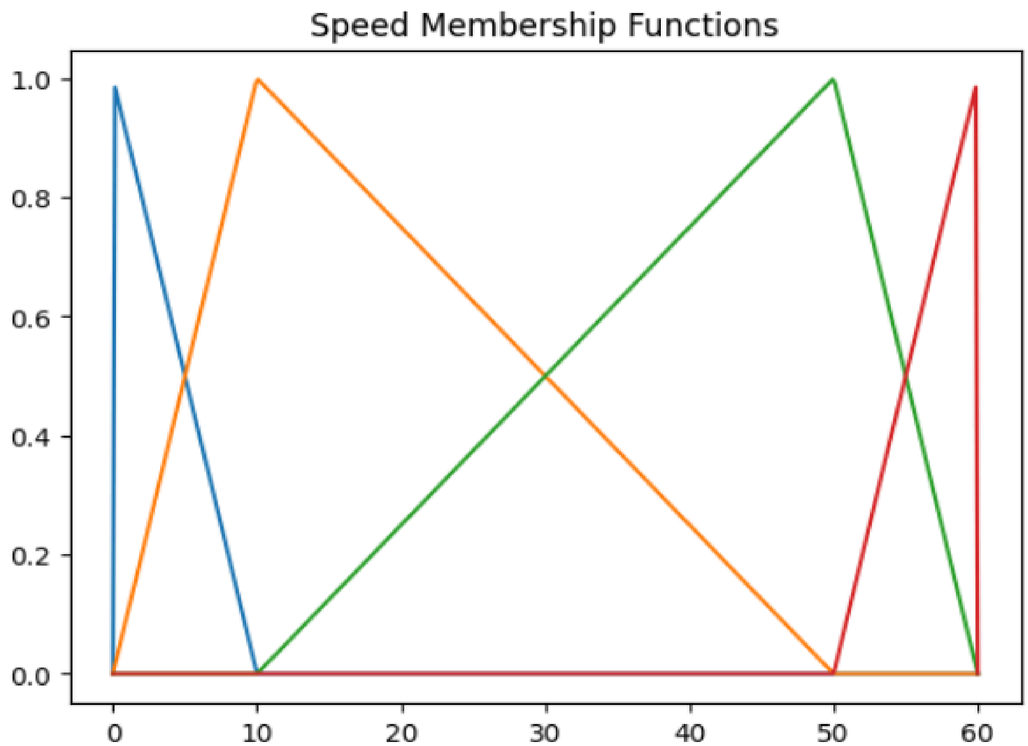
```
        return None
    a,b,c = bp_params[label]
    if (b - a) != 0:
        return a + mu * (b - a)
    if (c - b) != 0:
        return c - mu * (c - b)
    return None

rule_base = {
    ('VSD', 'VLS'): 'HP', ('VSD', 'LS'): 'HP', ('VSD', 'HS'): 'VHP',
('VSD','VHS'): 'VHP',
    ('SD',  'VLS'): 'LP', ('SD',  'LS'): 'LP', ('SD',  'HS'): 'HP',  ('SD','VHS'):
'VHP',
    ('LD',  'VLS'): 'VLP',('LD',  'LS'): 'VLP',('LD',  'HS'): 'LP',  ('LD','VHS'):
'HP',
    ('VLD', 'VLS'): 'VLP',('VLD', 'LS'): 'VLP',('VLD', 'HS'): 'LP',
('VLD','VHS'): 'LP'
}

distance = 110
speed = 52

dist_vals = {
    'VSD': mu_VSD(distance),
    'SD': mu_SD(distance),
    'LD': mu_LD(distance),
    'VLD': mu_VLD(distance)
}
speed_vals = {
    'VLS': mu_VLS(speed),
    'LS': mu_LS(speed),
    'HS': mu_HS(speed),
    'VHS': mu_VHS(speed)
}

fired = []
for dlab, dmu in dist_vals.items():
    for slab, smu in speed_vals.items():
```

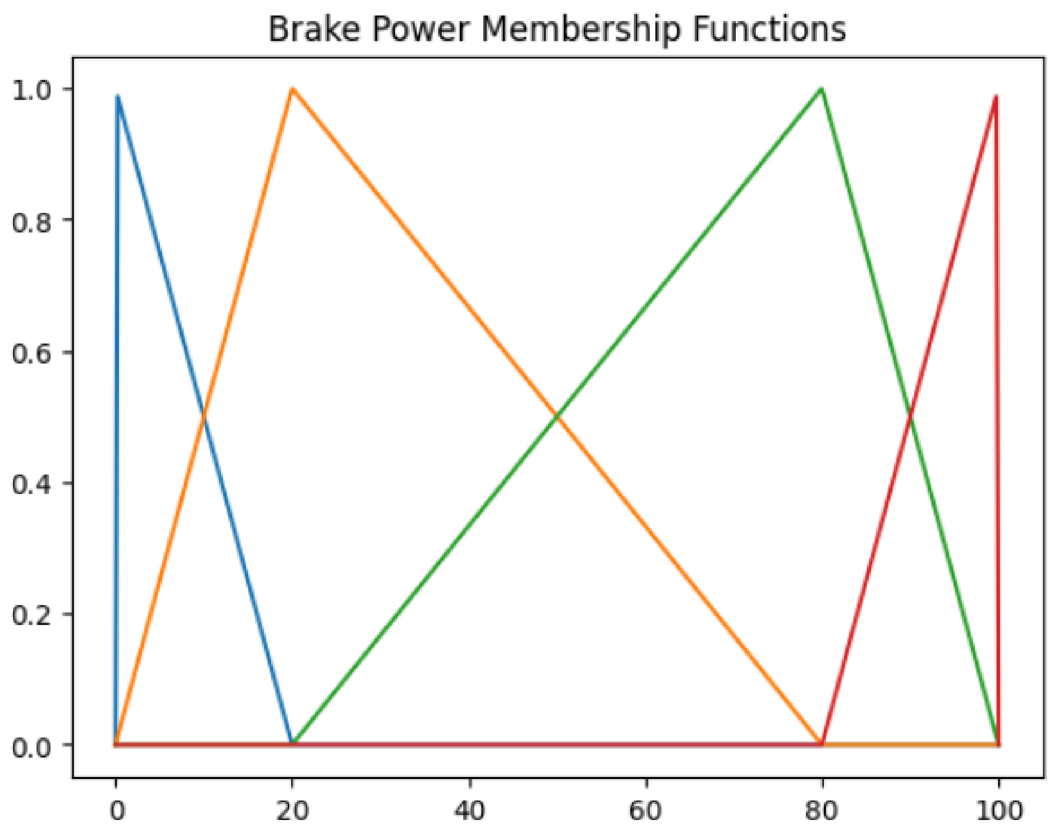
```
        if (dlab, slab) in rule_base:
            out = rule_base[(dlab, slab)]
            strength = min(dmu, smu)
            if strength > 0:
                fired.append((dlab, slab, out, strength))

agg = {}
for _,_,out,st in fired:
    agg[out] = max(agg.get(out, 0.0), st)

rep = []
for out, s in agg.items():
    z = inverse_z_for_label(out, s)
    if z is not None:
        rep.append((out, s, z))

rep_sorted = sorted(rep, key=lambda x: x[1], reverse=True)
top2 = rep_sorted[:2]

if len(top2) == 0:
    mean_of_max = None
elif len(top2) == 1:
    mean_of_max = top2[0][2]
else:
    mean_of_max = (top2[0][2] + top2[1][2]) / 2.0

print("Final = ", mean_of_max)

x1 = np.linspace(0,500,400)
plt.figure()
plt.plot(x1, [mu_VSD(t) for t in x1])
plt.plot(x1, [mu_SD(t) for t in x1])
plt.plot(x1, [mu_LD(t) for t in x1])
plt.plot(x1, [mu_VLD(t) for t in x1])
plt.title("Distance Membership Functions")
plt.show()

x2 = np.linspace(0,60,400)
```

```
plt.figure()
plt.plot(x2, [mu_VLS(t) for t in x2])
plt.plot(x2, [mu_LS(t) for t in x2])
plt.plot(x2, [mu_HS(t) for t in x2])
plt.plot(x2, [mu_VHS(t) for t in x2])
plt.title("Speed Membership Functions")
plt.show()

def mu_BP(label,z):
    a,b,c = bp_params[label]
    return triangular(z,a,b,c)

x3 = np.linspace(0,100,400)
plt.figure()

for label in bp_params:
    plt.plot(x3, [mu_BP(label,t) for t in x3])
plt.title("Brake Power Membership Functions")
plt.show()
```

RESULT:




Speed Membership Functions
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
60


Brake Power Membership Functions
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100

CONCLUSION:	From this experiment, I understood how fuzzy logic can be used to determine brake power based on distance and speed. First, the inputs were fuzzified into linguistic terms like Very Short Distance or High Speed. Then, the rule base was applied to find which conditions were fired. After that, the results were aggregated, and finally, defuzzification was done using the Mean of Max method. The final brake power obtained was around 76%, which means the vehicle needs strong braking in the given situation. Overall, I learned how fuzzy systems handle real-life uncertain conditions effectively.