Practical 8

Name: Sharvari Kishor More
Class: D20B
Roll No.: 35

Aim: To design a Fuzzy control system using Fuzzy tool/library.

Theory:
Traditional computers work on binary logic → inputs/outputs are either 0 (false) or 1 (true).
Fuzzy Logic extends this by allowing values between 0 and 1 to represent degrees of truth.
Example: instead of saying *"The distance is near"*, fuzzy logic allows *"The distance is 0.7 near and 0.3 medium"*.
This makes fuzzy logic useful for real-world problems where boundaries are not crisp (like braking, climate control, or washing machines).

scikit-fuzzy

- scikit-fuzzy (imported as skfuzzy) is a Python library that helps in building fuzzy logic systems.
- It is built on top of NumPy and SciPy.
- Features:
    1. Fuzzification (creating membership functions).
    2. Defining fuzzy rules (IF … THEN …).
    3. Performing inference (rule evaluation).
    4. Defuzzification (converting fuzzy output → crisp output).
    5. Visualization of membership functions.

Fuzzy Control System Stages

1. **Fuzzification** → Convert crisp inputs into fuzzy values (e.g., Distance = 30m → partly Near, partly Medium).
2. **Rule Base** → Define rules like:
    - IF Distance is Near AND Speed is Fast → Braking is Strong.
    - IF Distance is Far AND Speed is Slow → Braking is Light.
3. **Inference Engine** → Applies all rules and calculates fuzzy output.
4. **Defuzzification** → Converts fuzzy output back to a crisp value (e.g., Braking Force = 78%).

Code:

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
import matplotlib.pyplot as plt

# Step 1: Define Fuzzy Variables
```

```
distance = ctrl.Antecedent(np.arange(0, 101, 1), 'distance')  # in meters
speed = ctrl.Antecedent(np.arange(0, 121, 1), 'speed')        # in km/h
braking = ctrl.Consequent(np.arange(0, 101, 1), 'braking')    # braking force %

# Step 2: Define Membership Functions
distance['near']   = fuzz.trimf(distance.universe, [0, 0, 40])
distance['medium'] = fuzz.trimf(distance.universe, [20, 50, 80])
distance['far']    = fuzz.trimf(distance.universe, [60, 100, 100])

speed['slow']   = fuzz.trimf(speed.universe, [0, 0, 60])
speed['medium'] = fuzz.trimf(speed.universe, [30, 60, 90])
speed['fast']   = fuzz.trimf(speed.universe, [70, 120, 120])

braking['light']  = fuzz.trimf(braking.universe, [0, 0, 50])
braking['medium'] = fuzz.trimf(braking.universe, [25, 50, 75])
braking['strong'] = fuzz.trimf(braking.universe, [50, 100, 100])

# Step 3: Define Rules
rule1 = ctrl.Rule(distance['near'] & speed['fast'], braking['strong'])
rule2 = ctrl.Rule(distance['near'] & speed['medium'], braking['strong'])
rule3 = ctrl.Rule(distance['near'] & speed['slow'], braking['medium'])
rule4 = ctrl.Rule(distance['medium'] & speed['fast'], braking['strong'])
rule5 = ctrl.Rule(distance['medium'] & speed['medium'], braking['medium'])
rule6 = ctrl.Rule(distance['medium'] & speed['slow'], braking['light'])
rule7 = ctrl.Rule(distance['far'] & speed['fast'], braking['medium'])
rule8 = ctrl.Rule(distance['far'] & speed['medium'], braking['light'])
rule9 = ctrl.Rule(distance['far'] & speed['slow'], braking['light'])

# Step 4: Create Control System
brake_ctrl = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5, rule6, rule7, rule8, rule9])
brake = ctrl.ControlSystemSimulation(brake_ctrl)

# Step 5: Test with Input
brake.input['distance'] = 30   # meters
brake.input['speed'] = 90      # km/h
brake.compute()

print("Input Distance: 30m, Speed: 90km/h")
print("Braking Force Output: {:.2f}%".format(brake.output['braking']))

# Step 6: Custom Visualization (side-by-side)
fig, axs = plt.subplots(1, 3, figsize=(16, 4))

# Distance MF
```

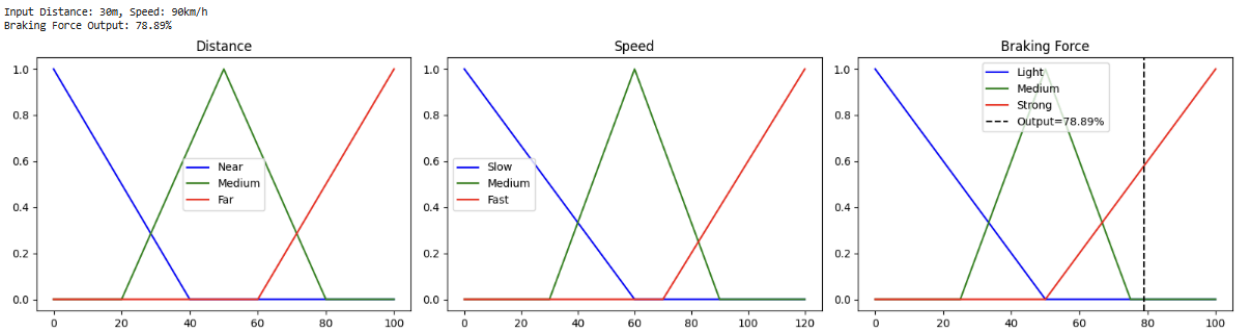
```
axs[0].plot(distance.universe, distance['near'].mf, 'b', label='Near')
axs[0].plot(distance.universe, distance['medium'].mf, 'g', label='Medium')
axs[0].plot(distance.universe, distance['far'].mf, 'r', label='Far')
axs[0].set_title('Distance')
axs[0].legend()

# Speed MF
axs[1].plot(speed.universe, speed['slow'].mf, 'b', label='Slow')
axs[1].plot(speed.universe, speed['medium'].mf, 'g', label='Medium')
axs[1].plot(speed.universe, speed['fast'].mf, 'r', label='Fast')
axs[1].set_title('Speed')
axs[1].legend()

# Braking MF
axs[2].plot(braking.universe, braking['light'].mf, 'b', label='Light')
axs[2].plot(braking.universe, braking['medium'].mf, 'g', label='Medium')
axs[2].plot(braking.universe, braking['strong'].mf, 'r', label='Strong')
axs[2].axvline(brake.output['braking'], color='k', linestyle='--',
label=f"Output={brake.output['braking']:.2f}%")
axs[2].set_title('Braking Force')
axs[2].legend()

plt.tight_layout()
plt.show()
```

Output:



Conclusion:

- We successfully designed and implemented a Fuzzy Braking System.
- Instead of abrupt ON/OFF braking, the fuzzy system provides smooth, human-like decision making.
- For Distance = 30m and Speed = 90 km/h, the system applies strong braking (~78%).
- This approach is very useful in autonomous vehicles, adaptive cruise control, and driver-assistance systems.

This is the Colab File