**Part 1: Design and Implementation of the Fuzzy Logic Controller (FLC)**

**1.1: Define the Problem and Determine Inputs/Outputs**

For the intelligent assistive care environment, we will design a Fuzzy Logic Controller (FLC) to regulate the ambient temperature, lighting, and humidity of a room. The inputs will be:

* **Temperature (°C)**: Measured by a temperature sensor.
* **Humidity (%)**: Measured by a humidity sensor.
* **Light Level (lux)**: Measured by a light sensor.

The outputs will be:

* **Heater Power (%)**: Control signal for the heater.
* **Fan Speed (%)**: Control signal for the cooling fan.
* **Blind Position (%)**: Control signal for the window blinds.

**1.2: Design Fuzzy Sets and Membership Functions**

We will use triangular membership functions for simplicity. For each input and output, we define the fuzzy sets as follows:

* **Temperature**: Cold, Warm, Hot
* **Humidity**: Dry, Comfortable, Humid
* **Light Level**: Dark, Dim, Bright

Outputs:

* **Heater Power**: Off, Low, Medium, High
* **Fan Speed**: Off, Low, Medium, High
* **Blind Position**: Closed, Half-open, Open

**1.3: Define Fuzzy Rules**

Here is an example set of rules for the FLC:

* If (Temperature is Cold) and (Humidity is Dry) then (Heater Power is High) and (Fan Speed is Off)
* If (Temperature is Warm) and (Light Level is Dark) then (Blind Position is Open)
* If (Temperature is Hot) and (Humidity is Humid) then (Fan Speed is High) and (Heater Power is Off)

**1.4: Implement FLC in MATLAB**

We will use the MATLAB Fuzzy Logic Toolbox to implement the FLC.

% Initialize Fuzzy Inference System  
fis = mamfis('Name', 'AssistiveCareFLC');  
  
% Define Input: Temperature  
fis = addInput(fis, [0 40], 'Name', 'Temperature');  
fis = addMF(fis, 'Temperature', 'trimf', [0 0 20], 'Name', 'Cold');  
fis = addMF(fis, 'Temperature', 'trimf', [10 20 30], 'Name', 'Warm');  
fis = addMF(fis, 'Temperature', 'trimf', [20 40 40], 'Name', 'Hot');  
  
% Define Input: Humidity  
fis = addInput(fis, [0 100], 'Name', 'Humidity');  
fis = addMF(fis, 'Humidity', 'trimf', [0 0 50], 'Name', 'Dry');  
fis = addMF(fis, 'Humidity', 'trimf', [30 50 70], 'Name', 'Comfortable');  
fis = addMF(fis, 'Humidity', 'trimf', [50 100 100], 'Name', 'Humid');  
  
% Define Input: Light Level  
fis = addInput(fis, [0 1000], 'Name', 'LightLevel');  
fis = addMF(fis, 'LightLevel', 'trimf', [0 0 500], 'Name', 'Dark');  
fis = addMF(fis, 'LightLevel', 'trimf', [250 500 750], 'Name', 'Dim');  
fis = addMF(fis, 'LightLevel', 'trimf', [500 1000 1000], 'Name', 'Bright');  
  
% Define Output: Heater Power  
fis = addOutput(fis, [0 100], 'Name', 'HeaterPower');  
fis = addMF(fis, 'HeaterPower', 'trimf', [0 0 25], 'Name', 'Off');  
fis = addMF(fis, 'HeaterPower', 'trimf', [0 25 50], 'Name', 'Low');  
fis = addMF(fis, 'HeaterPower', 'trimf', [25 50 75], 'Name', 'Medium');  
fis = addMF(fis, 'HeaterPower', 'trimf', [50 100 100], 'Name', 'High');  
  
% Define Output: Fan Speed  
fis = addOutput(fis, [0 100], 'Name', 'FanSpeed');  
fis = addMF(fis, 'FanSpeed', 'trimf', [0 0 25], 'Name', 'Off');  
fis = addMF(fis, 'FanSpeed', 'trimf', [0 25 50], 'Name', 'Low');  
fis = addMF(fis, 'FanSpeed', 'trimf', [25 50 75], 'Name', 'Medium');  
fis = addMF(fis, 'FanSpeed', 'trimf', [50 100 100], 'Name', 'High');  
  
% Define Output: Blind Position  
fis = addOutput(fis, [0 100], 'Name', 'BlindPosition');  
fis = addMF(fis, 'BlindPosition', 'trimf', [0 0 50], 'Name', 'Closed');  
fis = addMF(fis, 'BlindPosition', 'trimf', [0 50 100], 'Name', 'Half-open');  
fis = addMF(fis, 'BlindPosition', 'trimf', [50 100 100], 'Name', 'Open');  
  
% Define Fuzzy Rules  
ruleList = [  
 "If (Temperature is Cold) and (Humidity is Dry) then (HeaterPower is High) (FanSpeed is Off)";  
 "If (Temperature is Warm) and (LightLevel is Dark) then (BlindPosition is Open)";  
 "If (Temperature is Hot) and (Humidity is Humid) then (FanSpeed is High) (HeaterPower is Off)";  
 % Add more rules as needed  
];  
  
for i = 1:length(ruleList)  
 fis = addRule(fis, ruleList(i));  
end  
  
% Display FIS structure  
disp(fis);  
  
% Plot Membership Functions  
figure;  
subplot(3,1,1);  
plotmf(fis, 'input', 1);  
title('Temperature Membership Functions');  
subplot(3,1,2);  
plotmf(fis, 'input', 2);  
title('Humidity Membership Functions');  
subplot(3,1,3);  
plotmf(fis, 'input', 3);  
title('Light Level Membership Functions');  
  
% Evaluate FIS  
temp = 25; % Example temperature input  
humidity = 60; % Example humidity input  
light = 300; % Example light level input  
  
output = evalfis(fis, [temp, humidity, light]);  
disp('FIS Output:');  
disp(output);  
  
% Plot Control Surface  
figure;  
gensurf(fis);  
title('Control Surface');

**1.5: Analysis and Justification**

**Choice of Fuzzy Sets**: Triangular membership functions were chosen for simplicity and ease of interpretation.

**Inference Mechanism**: Mamdani inference was selected due to its intuitive rule base and suitability for human-like reasoning.

**Defuzzification Method**: The centroid method was chosen for defuzzification due to its balance between precision and computational efficiency.

**Control Rules**: Rules were crafted based on common sense and expected user comfort levels, ensuring a practical approach to environmental control.

**Performance Analysis**: We can analyze the rule activation and control surfaces to verify that the FLC achieves the desired control behaviors. The gensurf function helps visualize how inputs affect the outputs, ensuring the rules are functioning as intended.

**Part 2: Comparison of Optimization Techniques on CEC’2005 Functions**

**2.1: Select Benchmark Functions**

We choose three functions from the CEC’2005 suite:

1. **Sphere Function (F1)**
2. **Rosenbrock Function (F2)**
3. **Rastrigin Function (F3)**

**2.2: Define Optimization Algorithms**

We will compare:

1. Genetic Algorithm (GA)
2. Particle Swarm Optimization (PSO)
3. Simulated Annealing (SA)

**2.3: Implement Optimization Algorithms in MATLAB**

% Sphere Function  
sphere = @(x) sum(x.^2);  
  
% Rosenbrock Function  
rosenbrock = @(x) sum(100\*(x(2:end)-(x(1:end-1).^2)).^2 + (x(1:end-1)-1).^2);  
  
% Rastrigin Function  
rastrigin = @(x) sum(x.^2 - 10\*cos(2\*pi\*x) + 10);  
  
% Define optimization parameters  
numRuns = 15;  
dim = [2, 10]; % Dimensions to test  
  
% Results storage  
results = struct();  
  
% Functions to test  
funcs = {sphere, rosenbrock, rastrigin};  
funcNames = {'Sphere', 'Rosenbrock', 'Rastrigin'};  
  
% Optimization algorithms  
algos = {'ga', 'pso', 'sa'};  
algoNames = {'Genetic Algorithm', 'Particle Swarm Optimization', 'Simulated Annealing'};  
  
% Run optimization  
for d = dim  
 for i = 1:length(funcs)  
 func = funcs{i};  
 fName = funcNames{i};  
   
 for j = 1:length(algos)  
 algo = algos{j};  
 aName = algoNames{j};  
   
 bestResults = zeros(1, numRuns);  
 for run = 1:numRuns  
 switch algo  
 case 'ga'  
 [x, fval] = ga(func, d);  
 case 'pso'  
 [x, fval  
  
] = particleswarm(func, d);  
 case 'sa'  
 [x, fval] = simulannealbnd(func, rand(1, d), -5\*ones(1, d), 5\*ones(1, d));  
 end  
 bestResults(run) = fval;  
 end  
   
 results.(fName).(aName).(['D', num2str(d)]) = struct(...  
 'Best', min(bestResults), ...  
 'Worst', max(bestResults), ...  
 'Mean', mean(bestResults), ...  
 'StdDev', std(bestResults) ...  
 );  
 end  
 end  
end  
  
% Display results  
disp(results);

**2.4: Analyze Results**

For each function and dimension, we report:

* Best result
* Worst result
* Mean performance
* Standard deviation

**Conclusion**

This approach provides a comprehensive comparison of different optimization algorithms on various benchmark functions, highlighting the strengths and weaknesses of each method. By running each algorithm multiple times, we ensure robust and reliable performance metrics, essential for evaluating and improving optimization techniques.