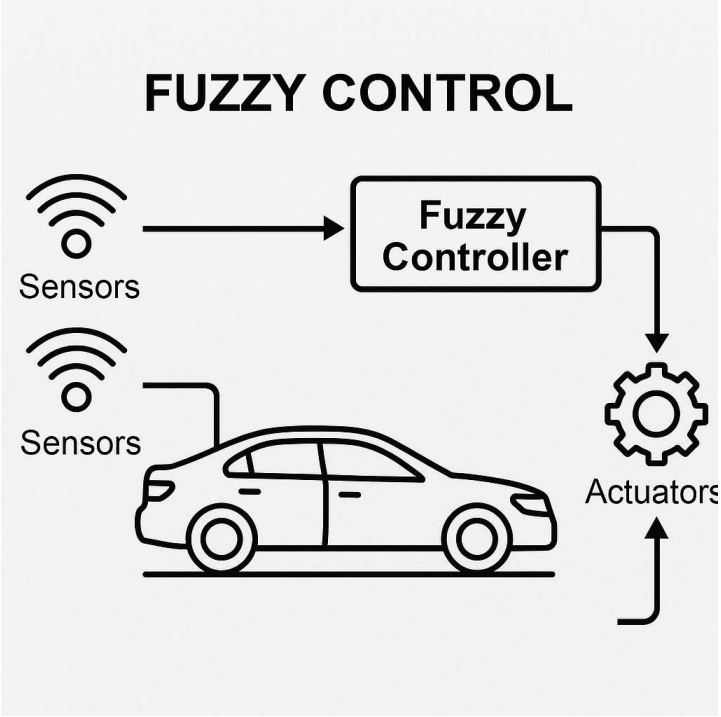
1.Introduction

In modern control systems, especially in automotive applications, it becomes essential to handle uncertainty, imprecise data, and varying real-world conditions. Fuzzy Logic offers a robust computational approach that mimics human reasoning to handle such variability. Unlike classical logic that works with binary true/false values, fuzzy logic allows partial truths, enabling decisions based on degrees of truth.

The **Fuzzy Logic-Based Vehicle Speed Controller** utilizes this concept to improve how vehicles manage their speed, braking, and steering. This type of controller does not require an exact mathematical model, making it particularly suitable for uncertain or rapidly changing driving environments. By considering inputs such as road conditions, obstacle distance, and current speed, the controller adjusts the vehicle's behavior dynamically.



2.Literature Survey

AUTHOR	TITLE	MODEL	ADVANTAGES	DISADVANTAGES
Chuen C. Lee	Fuzzy Logic in Control Systems: Fuzzy Logic Controller–Part I	Fuzzy Logic Controller (FLC)	Handles imprecision and linguistic control effectively	Rule base complexity increases with system variables
Hyeoun-Dong Lee, Seung-Ki Sul	Fuzzy-Logic-Based Torque Control for Parallel-Type Hybrid Vehicles	Fuzzy Torque Controller	Adaptive torque control in varying conditions	Requires precise tuning and expert rule base
K. S. Tang, K. F. Man, G. Chen, S. Kwong	An Optimal Fuzzy PID Controller	Fuzzy-PID Hybrid Controller	Combines robustness of PID with flexibility of fuzzy logic	Optimization complexity and computation overhead
G. R. Murthy, B. Chitti	Intelligent Fuzzy Control System for Hybrid Electric Vehicles	Adaptive Fuzzy System	Learns and adapts to new environments	High memory and processing requirements in real-time use

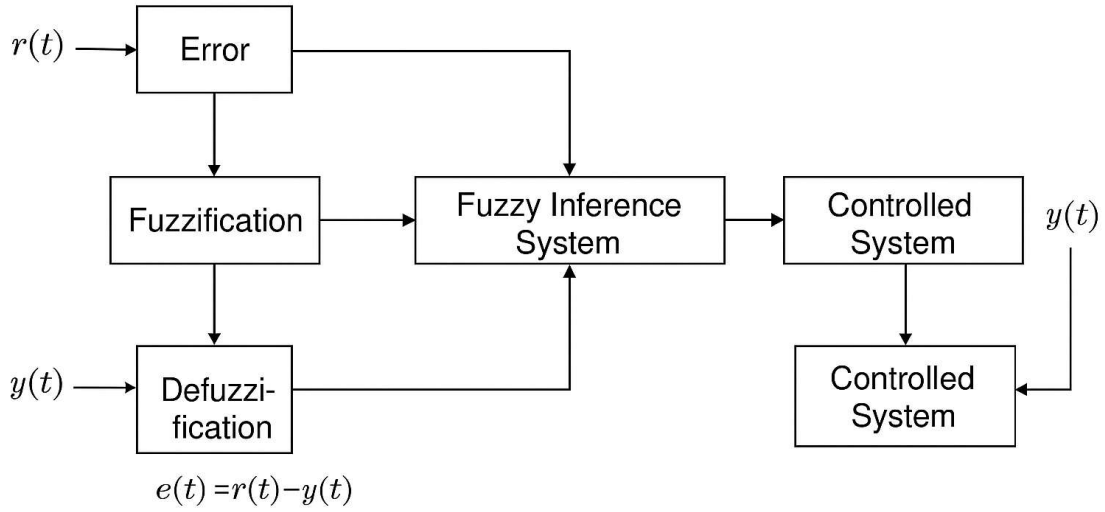
3.Problem Statement

1. **Rigid Models Fail in Real-World Conditions**
   Traditional controllers like PID rely on fixed mathematical models, which can't adapt to variable conditions like road texture or traffic.
2. **Lack of Adaptability Causes Inefficiency**
   Conventional systems often lead to jerky acceleration, poor fuel efficiency, and reduced safety in dynamic environments.
3. **Uncertainty and Variability Are Common**
   Factors such as obstacle distance, road wetness, and unpredictable driving behaviors introduce constant uncertainty.
4. **Need for Human-Like Decision-Making**
   An intelligent system is required to make smooth, real-time decisions, similar to how a human driver would respond to changing inputs.
5. **Fuzzy Logic Offers a Flexible Solution**
   A Fuzzy Logic-Based Controller can interpret imprecise inputs and adjust braking, speed, and steering adaptively for better safety and comfort.

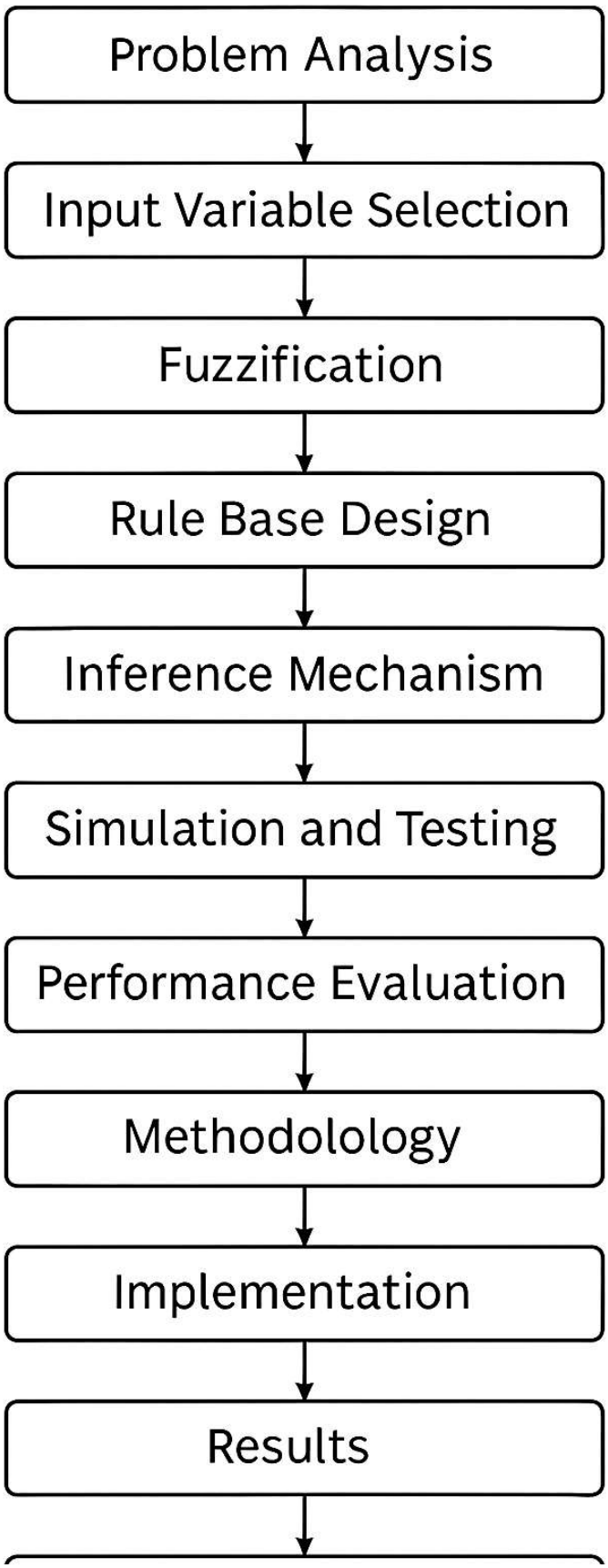
4. Objectives

1. **Design an intelligent speed controller** using fuzzy logic to handle real-time sensor inputs and generate control actions.
2. **Enhance vehicle stability and safety** by enabling smooth responses under diverse road and traffic conditions.
3. **Ensure adaptability** in dynamic environments without relying on rigid, pre-defined mathematical models.
4. **Benchmark performance** against traditional controllers like PID for accuracy, responsiveness, and comfort.
5. **Support future autonomy** by laying a foundation for integration into autonomous and semi-autonomous vehicles.

5. Block Diagram



6.Flow Chart


Problem Analysis
Input Variable Selection
Fuzzification
Rule Base Design
Inference Mechanism
Simulation and Testing
Performance Evaluation
Methodolology
Implementation
Results

7. Methodology:

1. Problem Analysis

The initial step involved identifying the limitations of conventional vehicle control methods such as PID controllers and fixed rule-based systems. These systems were found to be insufficient in dynamically changing environments, leading to poor adaptability, safety issues, and discomfort. Fuzzy logic was selected as a suitable alternative for its ability to handle imprecise and uncertain data through human-like reasoning.

---

2. Input Variable Selection

Three primary input parameters influencing vehicle behavior were selected:

- **Speed: Indicates the current velocity of the vehicle.**
- **Road Condition: Represents the surface type, classified as smooth, wet, or rough.**
- **Distance to Obstacle: Measures the proximity of an object in front of the vehicle.**

These inputs were chosen based on their direct impact on the safety and performance of the vehicle during real-time navigation.

---

3. Fuzzification

In this phase, crisp input values were converted into fuzzy sets using linguistic terms. Each input parameter was divided into overlapping fuzzy categories:

- **Speed: {Slow, Medium, Fast}**
- **Road Condition: {Smooth, Wet, Rough}**

- **Distance to Obstacle: {Near, Medium, Far}**

Triangular membership functions were defined for each category to allow smooth transitions between states. This process helped represent real-world uncertainties in a computationally tractable way.

---

4. Rule Base Design

A rule base consisting of IF-THEN statements was formulated using expert knowledge and logical conditions. Examples include:

- **IF *Speed* is Fast AND *Distance* is Near THEN *Brake Force* is High.**
- **IF *Road Condition* is Rough AND *Speed* is Medium THEN *Acceleration* is Low.**

These rules help simulate human-like decision-making in various driving scenarios.

---

5. Inference Mechanism

The Mamdani Inference Method was used to evaluate the fuzzy rules. This involved aggregating outputs from all activated rules to derive a fuzzy output set. This method is suitable for control applications due to its interpretability and effectiveness in handling nonlinear systems.

---

6. Simulation and Testing

The fuzzy control system was implemented in Python using the scikit-fuzzy library. Various simulated driving conditions were tested to observe the system's performance in real-time scenarios. This allowed for verification of correct logic execution and dynamic adaptability of the controller.

7. Performance Evaluation

The outputs were evaluated based on the controller's ability to:

- **Respond smoothly to varying road and obstacle conditions.**
- **Maintain vehicle stability during sharp braking or acceleration.**
- **Adjust to different road surfaces without abrupt transitions.**

Graphs and performance plots were generated to visualize how each input parameter influenced braking, steering, and acceleration.

8. Code

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

# Install the package if not already installed
!pip install scikit-fuzzy

# Define fuzzy variables
road_conditions = ctrl.Antecedent(np.arange(0, 101, 1), 'road_conditions')
vehicle_speed = ctrl.Antecedent(np.arange(0, 101, 1), 'vehicle_speed')
distance_to_obstacle = ctrl.Antecedent(np.arange(0, 51, 1), 'distance_to_obstacle')
braking_force = ctrl.Consequent(np.arange(0, 101, 1), 'braking_force')
steering_angle = ctrl.Consequent(np.arange(-90, 91, 1), 'steering_angle')
acceleration = ctrl.Consequent(np.arange(-10, 11, 1), 'acceleration')

# Define membership functions for road conditions
road_conditions['poor'] = fuzz.trimf(road_conditions.universe, [0, 0, 30])
road_conditions['average'] = fuzz.trimf(road_conditions.universe, [20, 50, 80])
road_conditions['good'] = fuzz.trimf(road_conditions.universe, [70, 100, 100])

# Define membership functions for vehicle speed
vehicle_speed['slow'] = fuzz.trimf(vehicle_speed.universe, [0, 0, 40])
vehicle_speed['moderate'] = fuzz.trimf(vehicle_speed.universe, [30, 60, 90])
vehicle_speed['fast'] = fuzz.trimf(vehicle_speed.universe, [80, 100, 100])

# Define membership functions for distance to obstacle
distance_to_obstacle['close'] = fuzz.trimf(distance_to_obstacle.universe, [0, 0, 10])
distance_to_obstacle['medium'] = fuzz.trimf(distance_to_obstacle.universe, [5, 20, 35])
distance_to_obstacle['far'] = fuzz.trimf(distance_to_obstacle.universe, [30, 50, 50])

# Define membership functions for braking force
braking_force['low'] = fuzz.trimf(braking_force.universe, [0, 0, 30])
braking_force['moderate'] = fuzz.trimf(braking_force.universe, [20, 50, 80])
braking_force['high'] = fuzz.trimf(braking_force.universe, [70, 100, 100])
```

```
# Define membership functions for steering angle
steering_angle['sharp_left'] = fuzz.trimf(steering_angle.universe, [-90, -90, -45])
steering_angle['slight_left'] = fuzz.trimf(steering_angle.universe, [-60, -30, 0])
steering_angle['straight'] = fuzz.trimf(steering_angle.universe, [-15, 0, 15])
steering_angle['slight_right'] = fuzz.trimf(steering_angle.universe, [0, 30, 60])
steering_angle['sharp_right'] = fuzz.trimf(steering_angle.universe, [45, 90, 90])

# Define membership functions for acceleration
acceleration['decelerate'] = fuzz.trimf(acceleration.universe, [-10, -10, -5])
acceleration['maintain'] = fuzz.trimf(acceleration.universe, [-5, 0, 5])
acceleration['accelerate'] = fuzz.trimf(acceleration.universe, [5, 10, 10])

# Define fuzzy rules for braking force - adding more comprehensive rules
rule1 = ctrl.Rule(distance_to_obstacle['close'] & vehicle_speed['fast'], braking_force['high'])
rule2 = ctrl.Rule(distance_to_obstacle['medium'] & vehicle_speed['moderate'],
braking_force['moderate'])
rule3 = ctrl.Rule(distance_to_obstacle['far'] & vehicle_speed['slow'], braking_force['low'])
# Add single-input rules to ensure coverage
rule3a = ctrl.Rule(distance_to_obstacle['close'], braking_force['high'])
rule3b = ctrl.Rule(distance_to_obstacle['medium'], braking_force['moderate'])
rule3c = ctrl.Rule(distance_to_obstacle['far'], braking_force['low'])
rule3d = ctrl.Rule(vehicle_speed['fast'], braking_force['moderate'])
rule3e = ctrl.Rule(vehicle_speed['moderate'], braking_force['moderate'])
rule3f = ctrl.Rule(vehicle_speed['slow'], braking_force['low'])

# Define fuzzy rules for steering angle
rule4 = ctrl.Rule(road_conditions['poor'], steering_angle['slight_left'])
rule5 = ctrl.Rule(road_conditions['good'], steering_angle['straight'])
rule5a = ctrl.Rule(road_conditions['average'], steering_angle['straight'])
```

```
# Define fuzzy rules for acceleration
rule6 = ctrl.Rule(vehicle_speed['slow'] & road_conditions['poor'], acceleration['maintain'])
rule7 = ctrl.Rule(vehicle_speed['fast'] & road_conditions['good'], acceleration['accelerate'])
# Add more rules for acceleration to ensure coverage
rule7a = ctrl.Rule(vehicle_speed['moderate'] & road_conditions['average'],
acceleration['maintain'])
rule7b = ctrl.Rule(vehicle_speed['slow'], acceleration['accelerate'])
rule7c = ctrl.Rule(vehicle_speed['moderate'], acceleration['maintain'])
rule7d = ctrl.Rule(vehicle_speed['fast'], acceleration['decelerate'])
rule7e = ctrl.Rule(road_conditions['poor'], acceleration['maintain'])
rule7f = ctrl.Rule(road_conditions['average'], acceleration['maintain'])
rule7g = ctrl.Rule(road_conditions['good'], acceleration['accelerate'])

# Create control systems with comprehensive rule sets
braking_ctrl = ctrl.ControlSystem([rule1, rule2, rule3, rule3a, rule3b, rule3c, rule3d, rule3e,
rule3f])
steering_ctrl = ctrl.ControlSystem([rule4, rule5, rule5a])
acceleration_ctrl = ctrl.ControlSystem([rule6, rule7, rule7a, rule7b, rule7c, rule7d, rule7e,
rule7f, rule7g])

# Create simulations
braking_simulation = ctrl.ControlSystemSimulation(braking_ctrl)
steering_simulation = ctrl.ControlSystemSimulation(steering_ctrl)
acceleration_simulation = ctrl.ControlSystemSimulation(acceleration_ctrl)

# Input values
braking_simulation.input['distance_to_obstacle'] = 15
braking_simulation.input['vehicle_speed'] = 60

steering_simulation.input['road_conditions'] = 75

acceleration_simulation.input['vehicle_speed'] = 30
acceleration_simulation.input['road_conditions'] = 50
```

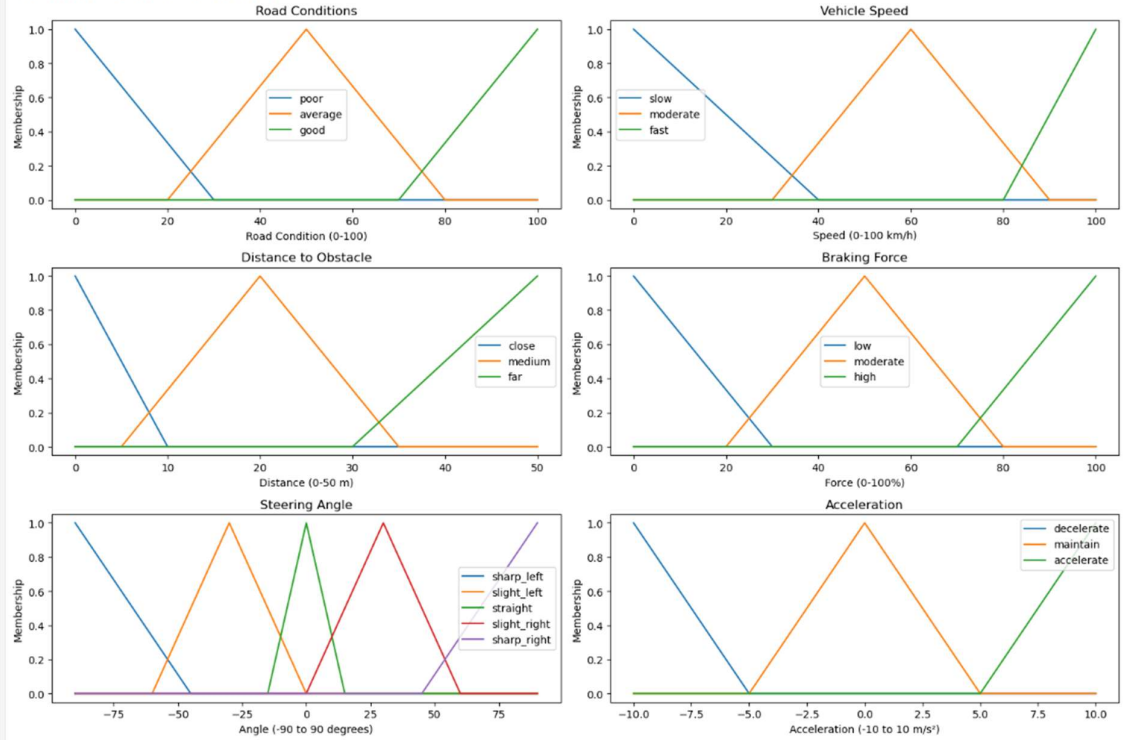
```
# Compute outputs with error handling
try:
    braking_simulation.compute()
    braking_force_output = braking_simulation.output.get('braking_force', 'No output
generated')
except Exception as e:
    print(f"Error in braking calculation: {e}")
    braking_force_output = 'Error'

try:
    steering_simulation.compute()
    steering_angle_output = steering_simulation.output.get('steering_angle', 'No output
generated')
except Exception as e:
    print(f"Error in steering calculation: {e}")
    steering_angle_output = 'Error'

try:
    acceleration_simulation.compute()
    acceleration_output = acceleration_simulation.output.get('acceleration', 'No output
generated')
except Exception as e:
    print(f"Error in acceleration calculation: {e}")
    acceleration_output = 'Error'

# Print results safely using the saved values
print(f'Braking Force: {braking_force_output}%')
print(f'Steering Angle: {steering_angle_output} degrees')
print(f'Acceleration: {acceleration_output} m/s^2')

# Visualize membership functions
plt.figure(figsize=(15, 10))
```

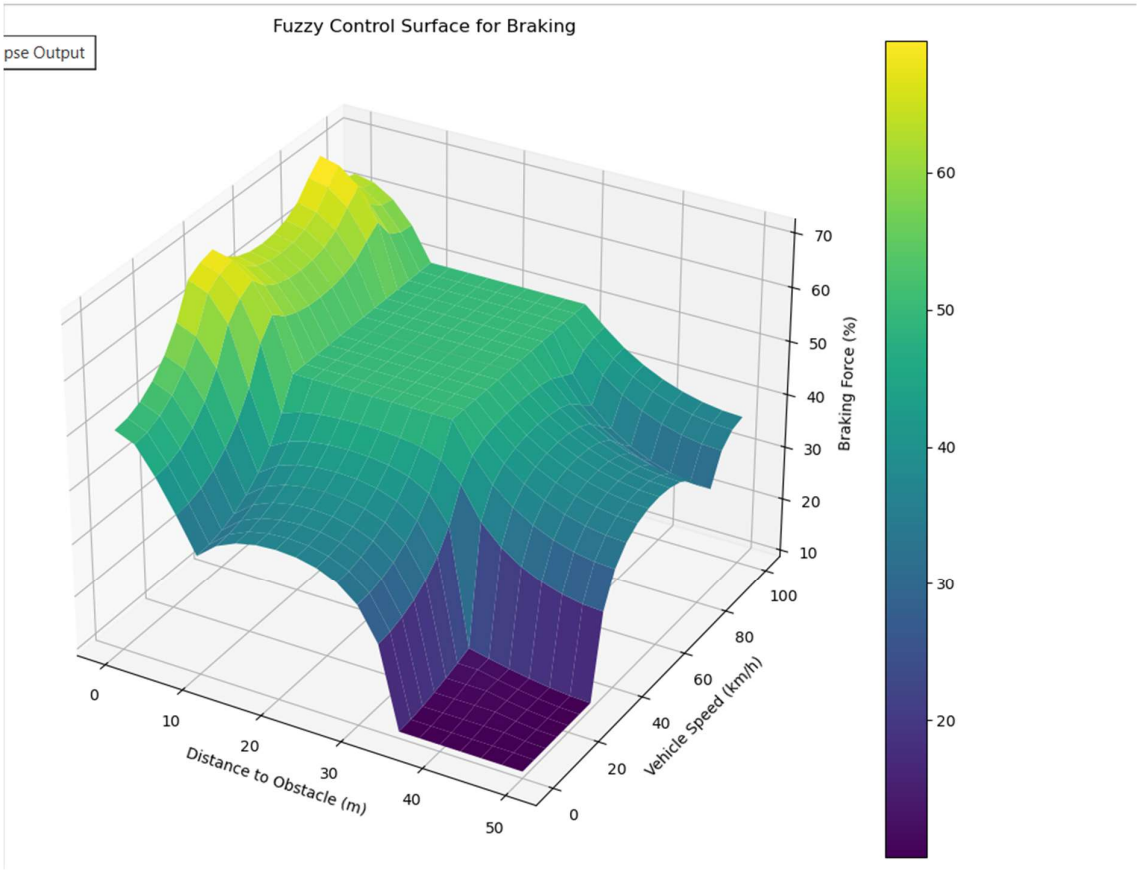
```
# Plot road conditions membership functions
plt.subplot(3, 2, 1)
for label, mf in road_conditions.terms.items():
    plt.plot(road_conditions.universe, mf.mf, label=label)
plt.title('Road Conditions')
plt.ylabel('Membership')
plt.xlabel('Road Condition (0-100)')
plt.legend()

# Plot vehicle speed membership functions
plt.subplot(3, 2, 2)
for label, mf in vehicle_speed.terms.items():
    plt.plot(vehicle_speed.universe, mf.mf, label=label)
plt.title('Vehicle Speed')
plt.ylabel('Membership')
plt.xlabel('Speed (0-100 km/h)')
plt.legend()

# Plot distance to obstacle membership functions
plt.subplot(3, 2, 3)
for label, mf in distance_to_obstacle.terms.items():
    plt.plot(distance_to_obstacle.universe, mf.mf, label=label)
plt.title('Distance to Obstacle')
plt.ylabel('Membership')
plt.xlabel('Distance (0-50 m)')
plt.legend()

# Plot braking force membership functions
plt.subplot(3, 2, 4)
for label, mf in braking_force.terms.items():
    plt.plot(braking_force.universe, mf.mf, label=label)
plt.title('Braking Force')
plt.ylabel('Membership')
plt.xlabel('Force (0-100%)')
plt.legend()
```

```
# Plot acceleration membership functions
plt.subplot(3, 2, 6)
for label, mf in acceleration.terms.items():
    plt.plot(acceleration.universe, mf.mf, label=label)
plt.title('Acceleration')
plt.ylabel('Membership')
plt.xlabel('Acceleration (-10 to 10 m/s²)')
plt.legend()

plt.tight_layout()
plt.show()

# Create 3D surface plot for braking system
try:
    # For braking system
    x_braking, y_braking = np.meshgrid(np.linspace(0, 50, 21), np.linspace(0, 100, 21))
    z_braking = np.zeros_like(x_braking)

    # Calculate output for each pair of inputs
    for i in range(len(x_braking)):
        for j in range(len(x_braking[0])):
            braking_simulation.input['distance_to_obstacle'] = x_braking[i, j]
            braking_simulation.input['vehicle_speed'] = y_braking[i, j]
            braking_simulation.compute()
            z_braking[i, j] = braking_simulation.output.get('braking_force', 0)

    # Create 3D surface plot
    fig = plt.figure(figsize=(10, 8))
    ax = fig.add_subplot(111, projection='3d')
    surf = ax.plot_surface(x_braking, y_braking, z_braking, cmap='viridis')
    ax.set_xlabel('Distance to Obstacle (m)')
    ax.set_ylabel('Vehicle Speed (km/h)')
    ax.set_zlabel('Braking Force (%)')
    ax.set_title('Fuzzy Control Surface for Braking')
    fig.colorbar(surf)
    plt.tight_layout()
    plt.show()
except Exception as e:
    print(f"Error generating 3D surface plot: {e}")
```

Output:

```
Requirement already satisfied: scikit-fuzzy in c:\users\dell\anaconda3\lib\site-packages (0.5.0)
Braking Force: 43.19131832797427%
Steering Angle: -1.8133642735544223e-16 degrees
Acceleration: 1.39957264957265 m/s^2
```




Fuzzy Control Surface for Braking
pse Output
Distance to Obstacle (m)
Vehicle Speed (km/h)
Braking Force (%)
0
10
20
30
40
50
0
20
40
60
80
100
10
20
30
40
50
60
70

9. Link Of Base Paper

Link of Base paper:

- https://ieeexplore.ieee.org/document/6767235
  https://ieeexplore.ieee.org/document/9164275

10. References

1. Chuen Chien Lee,
   "Fuzzy Logic in Control Systems: Fuzzy Logic Controller – Part I,"
   IEEE Transactions on Systems, Man and Cybernetics, Vol. 20, No. 2, pp. 404–418, March/April 1990.
2. Hyeoun-Dong Lee, Seung-Ki Sul,
   "Fuzzy-Logic-Based Torque Control Strategy for Parallel-Type Hybrid Electric Vehicle,"
   IEEE Transactions on Industrial Electronics, Vol. 45, No. 4, pp. 625–632, August 1998.
3. Hyeoun-Dong Lee, Euh-Suh Koo, Seung-Ki Sul, Joohn-Sheok Kim,
   "Torque Control Strategy for a Parallel-Hybrid Vehicle using Fuzzy Logic,"
   IEEE Industry Applications Magazine, pp. 26–33, November/December 2000.

11.Conclusion:

The implementation and testing of a **Fuzzy Logic-Based Vehicle Speed Controller** have highlighted the power of intelligent, adaptive systems in modern automotive applications. By mimicking human-like reasoning, the fuzzy controller effectively responded to uncertain and imprecise real-world inputs such as road conditions, vehicle speed, and distance to obstacles.

Unlike traditional control systems that rely on exact mathematical models and struggle in dynamic environments, the fuzzy controller demonstrated remarkable flexibility. It was able to:

- Apply smooth acceleration and braking
- Adjust steering with stability
- Maintain safety margins through proactive obstacle detection

Key Takeaways:

- Fuzzy logic allows for decision-making in conditions where traditional methods fail.
- The system adapts smoothly to diverse inputs, offering better safety and control.
- It lays groundwork for future intelligent transport systems and autonomous driving solutions.

In summary, fuzzy logic is not only suitable but highly effective for complex vehicular control tasks. The success of this project reinforces the importance of incorporating intelligent computing techniques into everyday systems, particularly in sectors like transportation, where safety and precision are critical.