# 1 Question 4

*A robot is navigating in an environment where there are obstacles it needs to avoid. The robot is equipped with sensors that can detect the obstacle. It also relays the distance and angle relative to the robot. Using these measurements, create a fuzzy inferencing system to decide the Speed and Steering.*

The following fuzzy quantities are defined, with their corresponding states:

**Antecedents**

- Distance from obstacle [D]
  - N=Near
  - F=Far
  - VF=Very Far
- Angle from Obstacle [A]
  - S=Small
  - M=Medium
  - L=Large

**Consequents**

- Speed [S]
  - SS= Slow Speed
  - MS=Medium Speed
  - FS=Fast Speed
  - MX= Maximum Speed
- Steering Turn [ST]
  - MST= Mild Turn
  - SST=Sharp Turn
  - VST= Very Sharp Turn

The universe of each of the quantities are defined below

| Fuzzy Quantity | Minimum Value | Maximum Value | Step |
|---|---|---|---|
| D | 0 | 10 | 0.5 |
| A | 0 | 90 | 1 |
| S | 0 | 5 | 0.2 |
| ST | 0 | 90 | 1 |

Table 1: Fuzzy membership values

1. Design a fuzzy system including: The problem was solved using the skfuzzy library in python.

   (a) The membership functions for each of the fuzzy quantities

      i. **Distance**: A trapezoidal membership function is used.

- Near :

$$\mu(x) = \begin{cases} 0 & x \leq 0 \\ \frac{x-0}{0.5} & 0 \leq x \leq 0.5 \\ 1 & 0.5 \leq x \leq 1 \\ \frac{1.5-x}{0.5} & 1 \leq x \leq 1.5 \end{cases}$$

- Far :

$$\mu(x) = \begin{cases} 0 & x \leq 1 \\ \frac{x-1}{0.5} & 1 \leq x \leq 1.5 \\ 1 & 1.5 \leq x \leq 6 \\ \frac{6.5-x}{0.5} & 6 \leq x \leq 6.5 \end{cases}$$

- Very Far :

$$\mu(x) = \begin{cases} 0 & x \leq 6 \\ \frac{x-6}{0.5} & 6 \leq x \leq 6.5 \\ 1 & x \geq 6.5 \end{cases}$$

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl

distance_obj = ctrl.Antecedent(np.arange(0, 10, 0.5), 'distance')
distance_obj['near'] = fuzz.trapmf(distance_obj.universe, [0, 0.5, 1, 1.5])
distance_obj['far'] = fuzz.trapmf(distance_obj.universe, [1, 1.5, 6, 6.5])
distance_obj['veryfar'] = fuzz.trapmf(distance_obj.universe, [6, 6.5, 10,
    10])
```

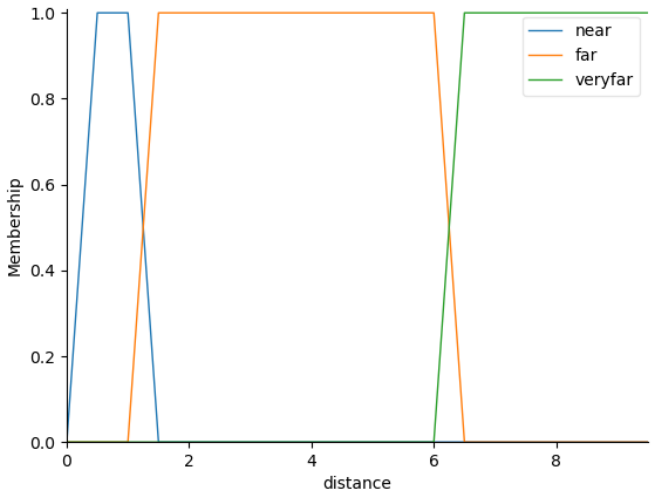


Figure 1: Membership function graph for distance antecedent

ii. **Angle**: A trapezoidal membership function is used.

- Small :

$$\mu(x) = \begin{cases} 0 & x \leq 0 \\ \frac{x-0}{5} & 0 \leq x \leq 5 \\ 1 & 5 \leq x \leq 10 \\ \frac{15-x}{5} & 10 \leq x \leq 15 \end{cases}$$

- Medium :

$$\mu(x) = \begin{cases} 0 & x \leq 10 \\ \frac{x-10}{5} & 10 \leq x \leq 15 \\ 1 & 15 \leq x \leq 45 \\ \frac{50-x}{5} & 45 \leq x \leq 50 \end{cases}$$

- Large :

$$\mu(x) = \begin{cases} 0 & x \leq 45 \\ \frac{x-45}{5} & 45 \leq x \leq 50 \\ 1 & x \geq 50 \end{cases}$$

```
angle_obj = ctrl.Antecedent(np.arange(0, 90, 1), 'angle')

angle_obj['small'] = fuzz.trapmf(angle_obj.universe, [0, 5, 10, 15])
angle_obj['medium'] = fuzz.trapmf(angle_obj.universe, [10, 15, 45, 50])
angle_obj['large'] = fuzz.trapmf(angle_obj.universe, [45, 50, 90, 90])
```

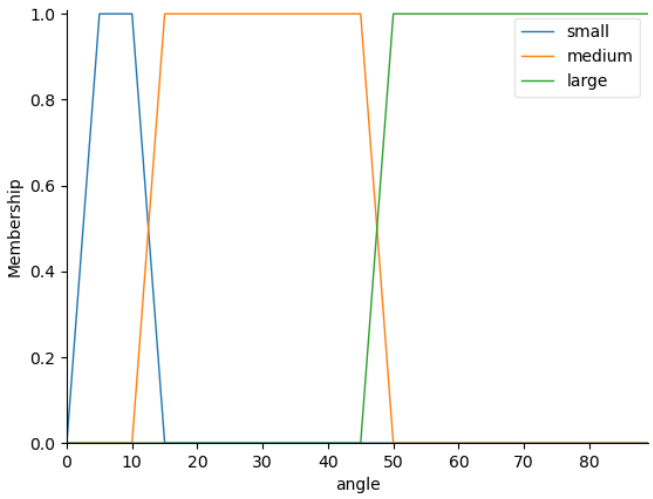


Figure 2: Membership function graph for angle antecedent

iii. **Speed**: A trapezoidal membership function is used for the consequent.

- Slow Speed :

$$\mu(x) = \begin{cases} 0 & x \leq 0 \\ \frac{x-0.2}{0.2} & 0.2 \leq x \leq 0.4 \\ 1 & 0.2 \leq x \leq 0.4 \\ \frac{0.6-x}{0.2} & 0.4 \leq x \leq 0.6 \end{cases}$$

- Medium Speed:

$$\mu(x) = \begin{cases} 0 & x \leq 0.4 \\ \frac{x-0.4}{0.2} & 0.4 \leq x \leq 0.6 \\ 1 & 0.6 \leq x \leq 2 \\ \frac{2.2-x}{0.2} & 2 \leq x \leq 2.2 \end{cases}$$

- Fast Speed :

$$\mu(x) = \begin{cases} 0 & x \leq 2 \\ \frac{x-2}{0.2} & 2 \leq x \leq 2.2 \\ 1 & 2.2 \leq x \leq 4 \\ \frac{4.2-x}{0.2} & 4 \leq x \leq 4.2 \end{cases}$$

- Max Speed :

$$\mu(x) = \begin{cases} 0 & x \leq 4 \\ \frac{x-4}{0.2} & 4 \leq x \leq 4.2 \\ 1 & x \geq 5 \end{cases}$$

```
speed_obj = ctrl.Consequent(np.arange(0, 5, 0.2), 'speed', 'centroid')

speed_obj['slowspeed'] = fuzz.trapmf(speed_obj.universe, [0, 0.2, 0.4,
    0.6])
speed_obj['mediumspeed'] = fuzz.trapmf(speed_obj.universe, [0.4, 0.6, 2,
    2.2])
speed_obj['fastspeed'] = fuzz.trapmf(speed_obj.universe, [2, 2.2, 4, 4.2])
speed_obj['maxspeed'] = fuzz.trapmf(speed_obj.universe, [4, 4.2, 5, 5])
```

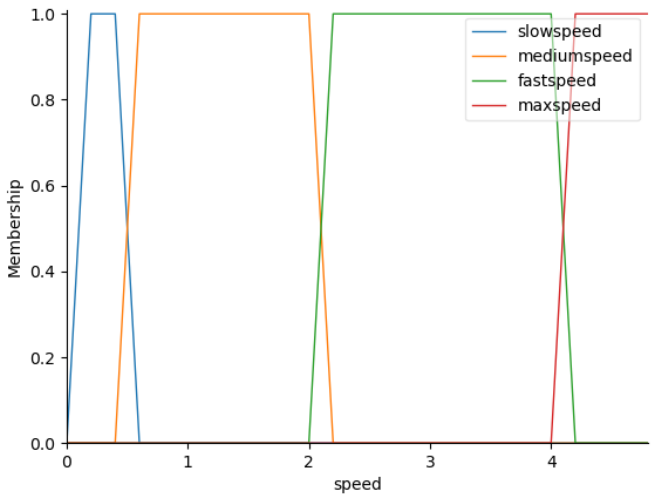


Figure 3: Membership function graph for speed consequent

iv. **Angle**: A gaussian membership function is used for the consequent.

- Mild Turn : c=0, $\mu = 10$
- Sharp Turn: c=45, $\mu = 10$
- Very Sharp Turn : c=90, $\mu = 10$

```
steering_obj = ctrl.Consequent(np.arange(0, 90, 1), 'steering', 'centroid')

sigma = 10
steering_obj['mildturn'] = fuzz.gaussmf(steering_obj.universe, 0, sigma)
steering_obj['sharpturn'] = fuzz.gaussmf(steering_obj.universe, 45, sigma)
steering_obj['verysharpturn'] = fuzz.gaussmf(
    steering_obj.universe, 90, sigma)
```

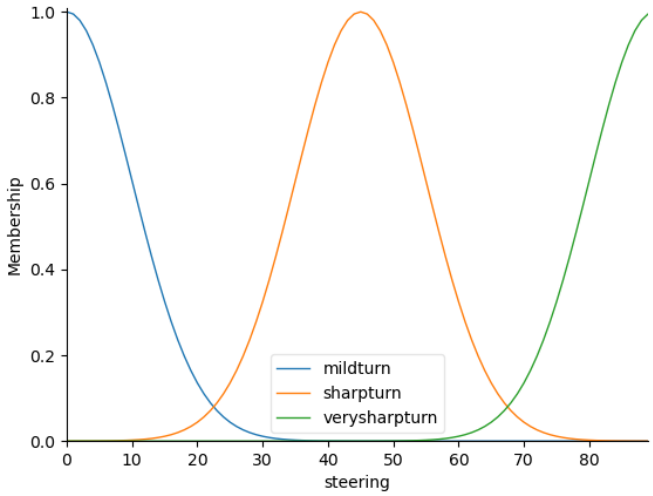


Figure 4: Membership function graph for speed consequent

## 2 Rules

Rules in fuzzy control system help in connecting antecedents to consequents.

1. **Rule One**:

   -*if* distance is NEAR and the angle is SMALL *then* speed is SLOW SPEED and the steering turn is VERY SHARP TURN.

2. **Rule Two**:

   -*if* distance is NEAR and the angle is MEDIUM *then* speed is MEDIUM SPEED and the steering turn is SHARP TURN.

3. **Rule Three**:

   -*if* distance is NEAR and the angle is LARGE *then* speed is MEDIUM SPEED and the steering turn is MILD TURN.

4. **Rule Four**:

   -*if* distance is FAR and the angle is SMALL *then* speed is MEDIUM SPEED and the steering turn is SHARP TURN.

5. **Rule Five**:

   -*if* distance is FAR and the angle is MEDIUM *then* speed is MEDIUM SPEED and the steering turn is MAX TURN.

6. **Rule Six**:

   -*if* distance is FAR and the angle is LARGE *then* speed is MAX SPEED and the steering turn is MILD TURN.

7. **Rule Seven**:

   -*if* distance is VERY FAR and the angle is SMALL *then* speed is MAX SPEED and the steering turn is MILD TURN.

8. **Rule Eight**:

   -*if* distance is VERY FAR and the angle is MEDIUM *then* speed is MAX SPEED and the steering turn is MILD TURN.

9. **Rule Nine**:

   -*if* distance is VERY FAR and the angle is LARGE *then* speed is MAX SPEED and the steering turn is MILD TURN.

The above rules are implemented in python as below.

```
# defining the rules
rule_one = ctrl.Rule(distance_obj['near'] & angle_obj['small'],
                     (speed_obj['slowspeed'], steering_obj['verysharpturn']))
rule_two = ctrl.Rule(distance_obj['near'] & angle_obj['medium'],
                     (speed_obj['mediumspeed'], steering_obj['sharpturn']))
rule_three = ctrl.Rule(distance_obj['near'] & angle_obj['large'],
                       (speed_obj['mediumspeed'], steering_obj['mildturn']))
rule_four = ctrl.Rule(distance_obj['far'] & angle_obj['small'],
                      (speed_obj['mediumspeed'], steering_obj['sharpturn']))
rule_five = ctrl.Rule(distance_obj['far'] & angle_obj['medium'],
                      (speed_obj['mediumspeed'], steering_obj['mildturn']))
rule_six = ctrl.Rule(distance_obj['far'] & angle_obj['large'],
                     (speed_obj['maxspeed'], steering_obj['mildturn']))
rule_seven = ctrl.Rule(distance_obj['veryfar'] & angle_obj['small'],
                       (speed_obj['maxspeed'], steering_obj['mildturn']))
rule_eight = ctrl.Rule(distance_obj['veryfar'] & angle_obj['medium'],
                       (speed_obj['maxspeed'], steering_obj['mildturn']))
rule_nine = ctrl.Rule(distance_obj['veryfar'] & angle_obj['large'],
                      (speed_obj['maxspeed'], steering_obj['mildturn']))
```

## 3 Results

The Mamdani Fuzzy inference system is implemented in the skfuzzy library by the function *compute_rule* in the ControlSystemSimulation class. It performs the mamdani inference in 3 steps:

- **Aggregation** : This finds the net accomplishment of the antecedent by AND-ing or OR-ing together all the membership values of the terms that make up the accomplishment condition.
- **Activation** : The degree of membership of the consequence is determined by the degree of accomplishment of the antecedent, which is what we determined in step 1. The only difference would be if the consequent has a weight.
- **Accumulation** : Apply the activation to each consequent, accumulating multiple rule firings into a single membership value.

```
obstacle_ctrl = ctrl.ControlSystem(
    [rule_one, rule_two, rule_three, rule_four, rule_five, rule_six, rule_seven,
        rule_eight, rule_nine])

obstacle_avoidance = ctrl.ControlSystemSimulation(obstacle_ctrl)
obstacle_avoidance.input['distance'] = 1
obstacle_avoidance.input['angle'] = 90
obstacle_avoidance.compute()

print(obstacle_avoidance.output['speed'])
speed_obj.view(sim=obstacle_avoidance)

print(obstacle_avoidance.output['steering'])
steering_obj.view(sim=obstacle_avoidance)
```

The Defuzzification method used to generate the crisp output is centroid and is defined in the parameters of the consequent.

1. **Example 1**:

   Distance of the obstacle from the robot is near value but the angle is large enough to avoid the object:

```
obstacle_avoidance.input['distance'] = 1
obstacle_avoidance.input['angle'] = 90

obstacle_avoidance.compute()
```

   (a) **Speed**: Using the compute method in the ControlSystemSimulation class on its object we get the speed to be **1.3** from the Fuzzy control system.

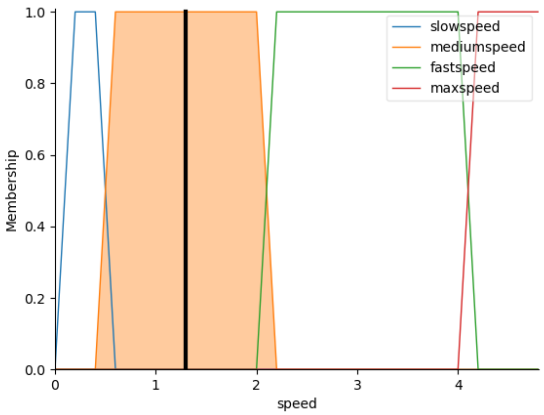


Figure 5: Defuzzufied value of the speed

   (b) **Steering Turn**: The steering value computed by the fuzzy system is **7.98 deg**.

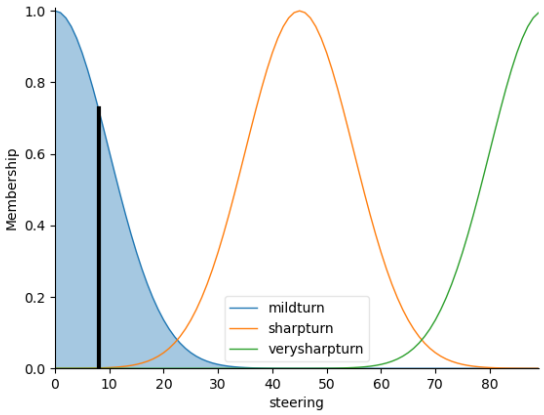


Figure 6: Defuzzufied value of the steering turn

2. **Example 2**:

   Distance of the obstacle from the robot is a very far value but the angle is small enough to hit the object:

```
obstacle_avoidance.input['distance'] = 7
obstacle_avoidance.input['angle'] = 1

obstacle_avoidance.compute()
```

   (a) **Speed**: Using the compute method in the ControlSystemSimulation class on its object we get the speed to be **4.4** from the Fuzzy control system.

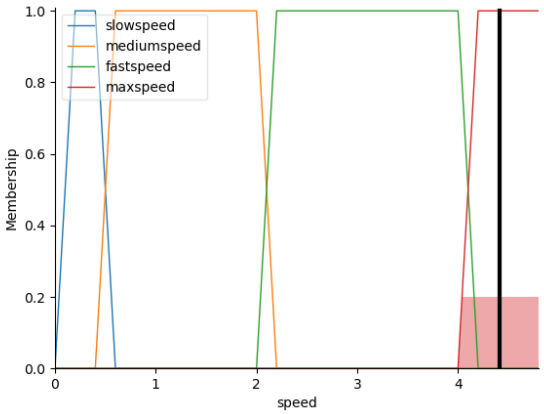


Figure 7: Defuzzufied value of the speed

   (b) **Steering Turn**: The steering value computed by the fuzzy system is **11.6 deg**.

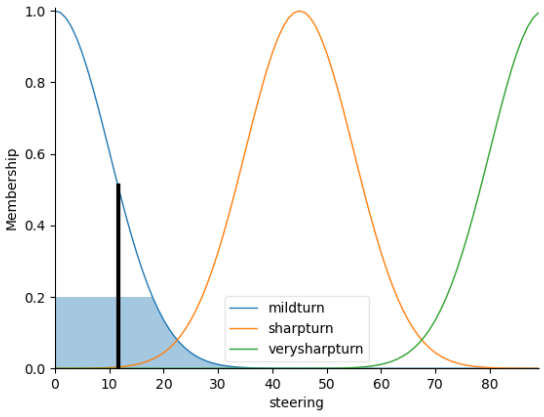


Figure 8: Defuzzufied value of the steering turn