SISTEM ROBOT OTONOM

Section 7:

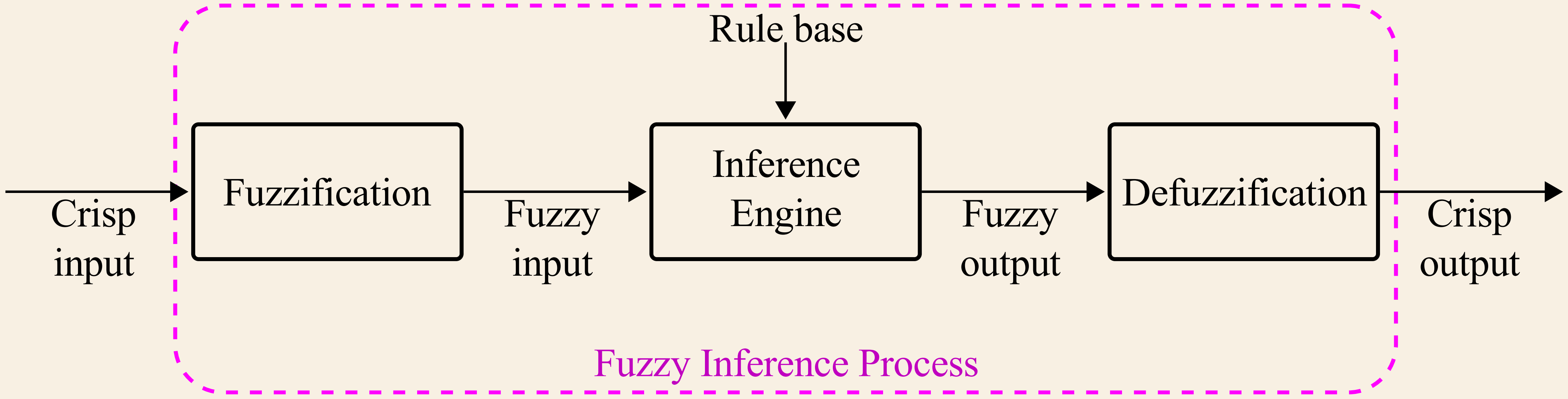
A brief introduction to fuzzy inference system part 2

Djoko Purwanto; M. Q. Zaman

djoko@its.ac.id; muhammad.zaman@its.ac.id

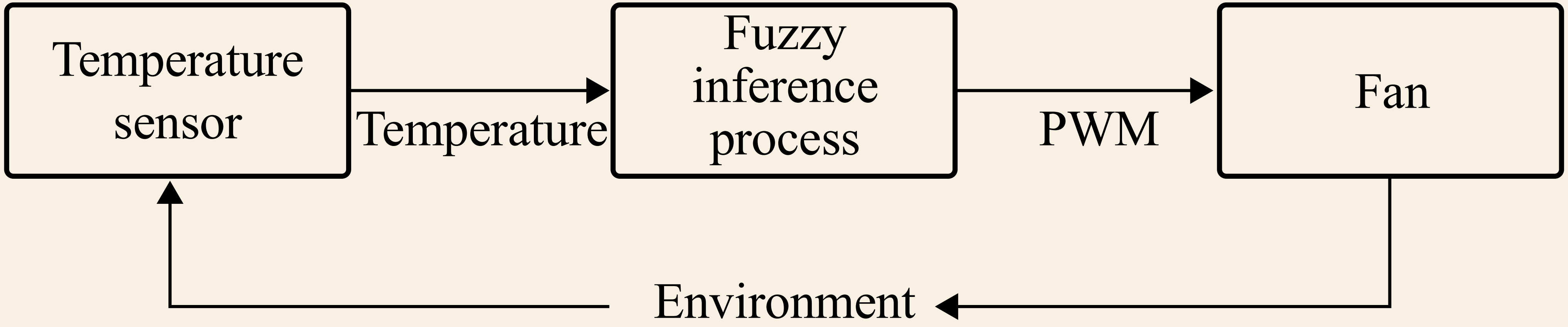
FUZZY INFERENCE SYSTEM

THE OVERALL INFERENCE PROCESS



What we learned in previous section is called Fuzzification process

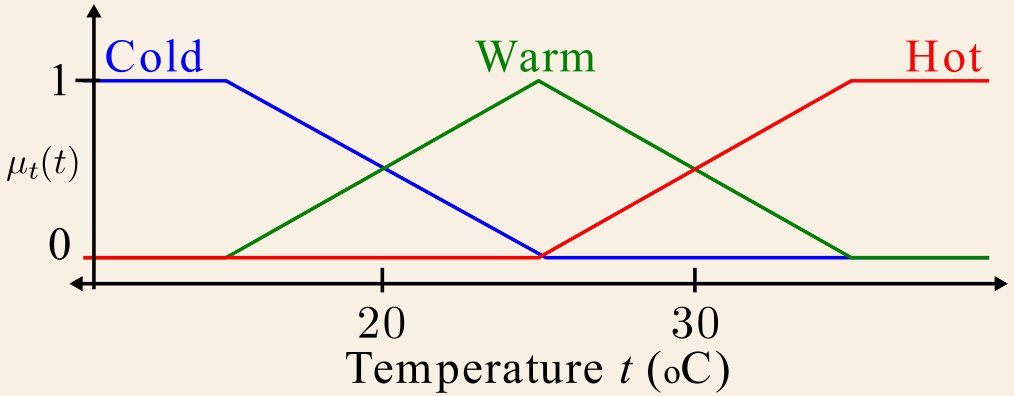
THE OVERALL FEEDBACK PROCESS



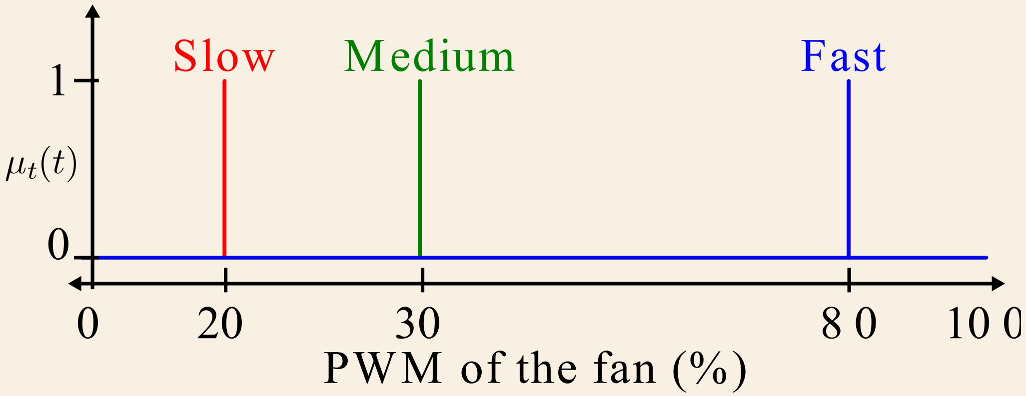
Please note that, for now, there is no temperature set point

FUZZY OUTPUT

SINGLETON FUZZY OUTPUT



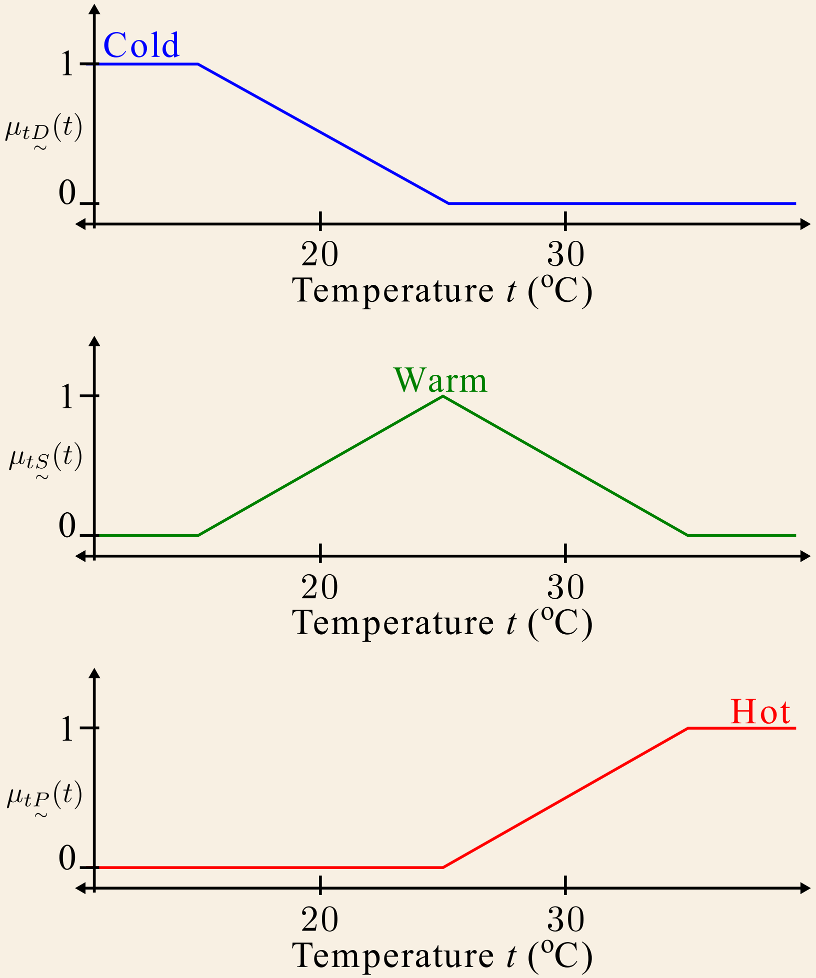
Fuzzy membership functions of input



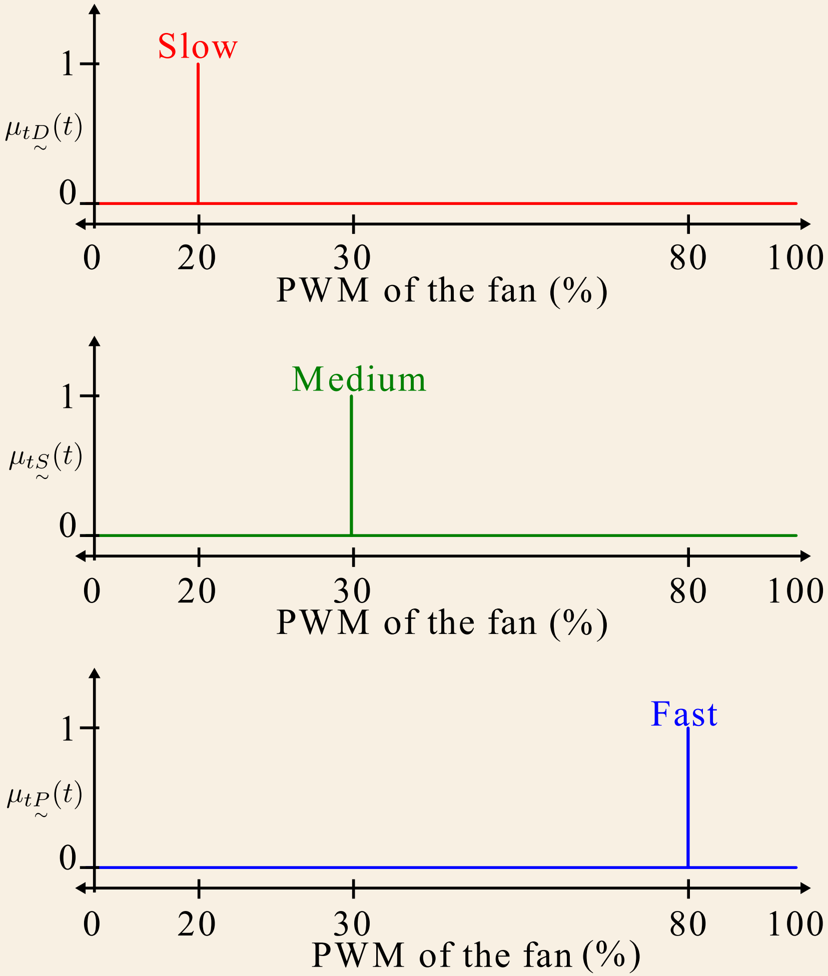
Singleton fuzzy membership output

Input membership function for comparison

SINGLETON FUZZY OUTPUT



Fuzzy membership functions of input



Singleton fuzzy membership output

Input membership function for comparison

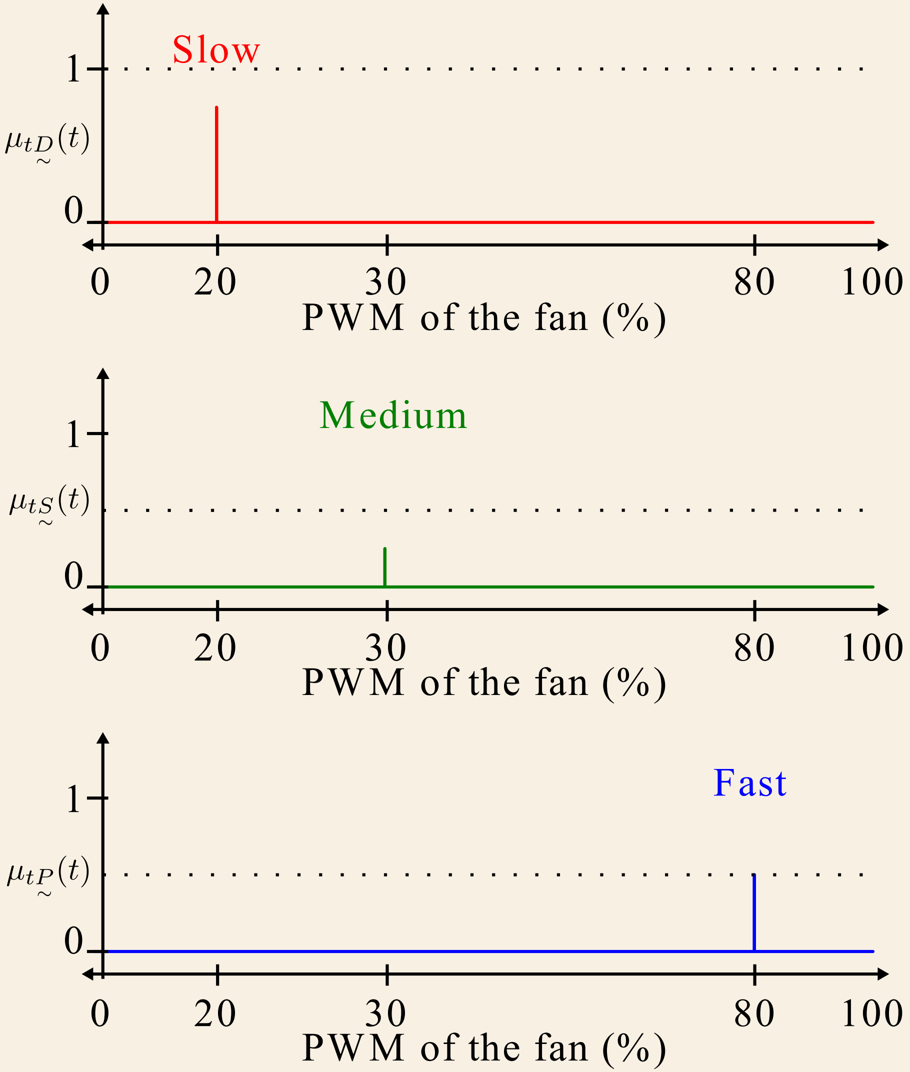
SINGLETON FUZZY OUTPUT

Singleton fuzzy output also has varying membership degree, for example:

- 0.75 of "slow", or 0.75 of "20"
- 0.3 of "medium", or 0.3 of "30"
- 0.5 of "fast", or 0.5 of "80"

Fuzzy output must be converted to crisp output, such as:

- 25% of PWM output
- 70% of PWM output



Singleton fuzzy membership output

RULE-BASED INFERENCE ENGINE

1 input and 1 output

THE RULE

1. **IF** Temp. is *Cold*, **THEN** PWM output is *Slow*
2. **IF** Temp. is *Warm*, **THEN** PWM output is *Medium*
3. **IF** Temp. is *Hot*, **THEN** PWM output is *Fast*

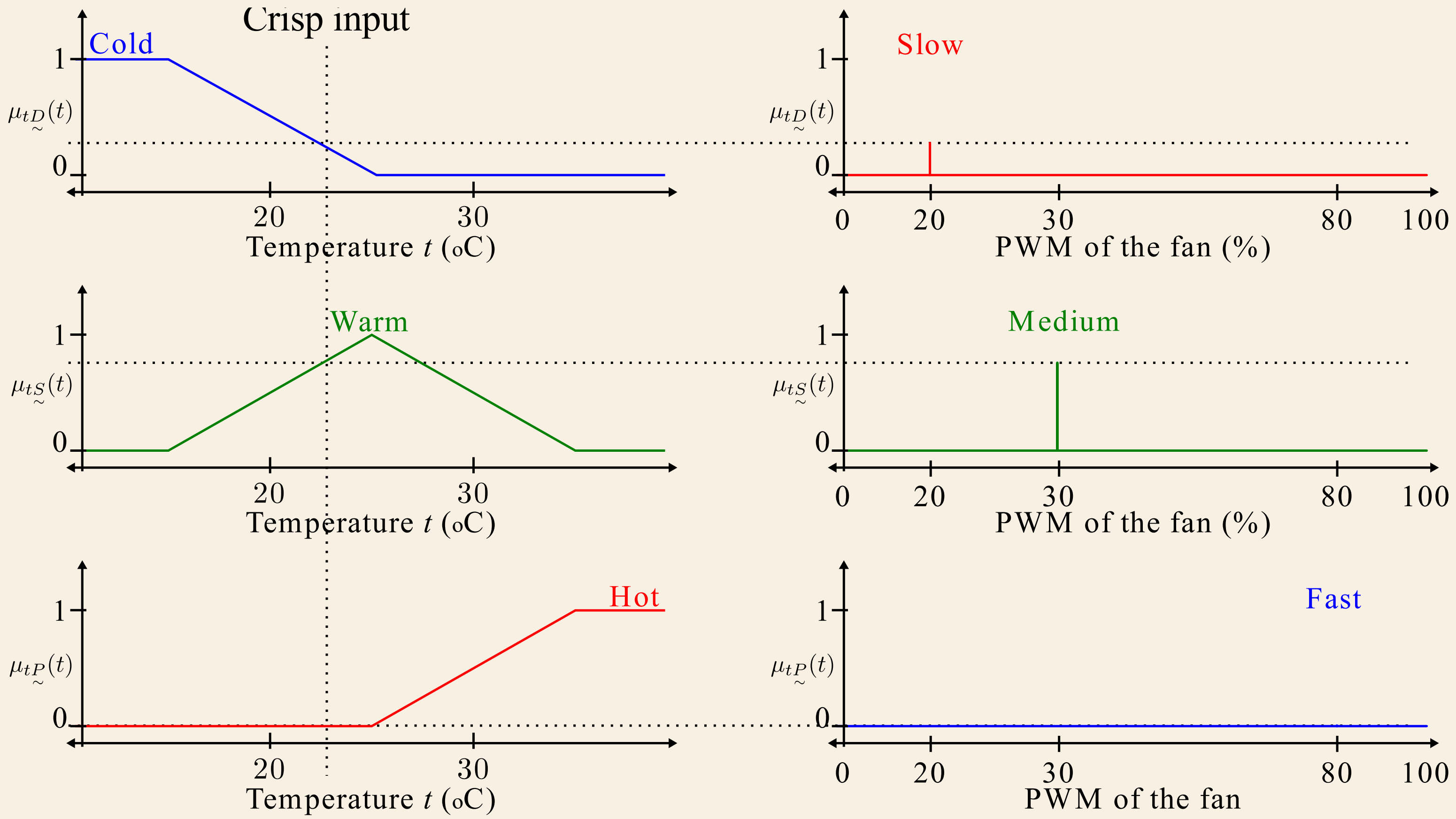
THE OUTPUT'S FIRING DEGREE

Assigning output's firing degree of 1 input fuzzy inference rule

1. **IF** Temp. is *(0.3) Cold*, **THEN** PWM output is *(0.3) Slow*
2. **IF** Temp. is *(0.7) Warm*, **THEN** PWM output is *(0.7) Medium*
3. **IF** Temp. is *(0.0) Hot*, **THEN** PWM output is *(0.0) Fast*

THE OUTPUT'S FIRING DEGREE

Assigning output's firing degree of 1 input fuzzy inference rule



Fuzzy membership functions of input

Singleton fuzzy membership output

Visualization of inference process

RULE-BASED INFERENCE ENGINE

2 input and 1 output

THE RULE

Example 2 fuzzy inputs and 1 fuzzy output rule:

- Input is fuzzified into 2 memberships.
- Output membership number is the same number with input combinations.

Rules:

1. **IF** left sensor is *near* and right sensor is *near*, **THEN** Left motor is -5
2. **IF** left sensor is *far* and right sensor is *near*, **THEN** Left motor is -5
3. **IF** left sensor is *near* and right sensor is *far*, **THEN** Left motor is 5
4. **IF** left sensor is *far* and right sensor is *far*, **THEN** Left motor is 5

THE OUTPUT'S FIRING DEGREE

Assigning output's firing degree of 2 inputs fuzzy inference rule.
In this case, memberships of input 1 is *0.3 near* or *0.7 far*
While memberships of input 2 is *0.1 near* or *0.9 far*

1. **IF** left sensor is *(0.3) near* and right sensor is *(0.1) near*, **THEN** Left motor is *(0.1)* -5
2. **IF** left sensor is *(0.7) far* and right sensor is *(0.1) near*, **THEN** Left motor is *(0.1)* -5
3. **IF** left sensor is *(0.3) near* and right sensor is *(0.9) far*, **THEN** Left motor is *(0.3)* 5
4. **IF** left sensor is *(0.7) far* and right sensor is *(0.9) far*, **THEN** Left motor is *(0.7)* 5

`min()` function is used to implement `and` logic

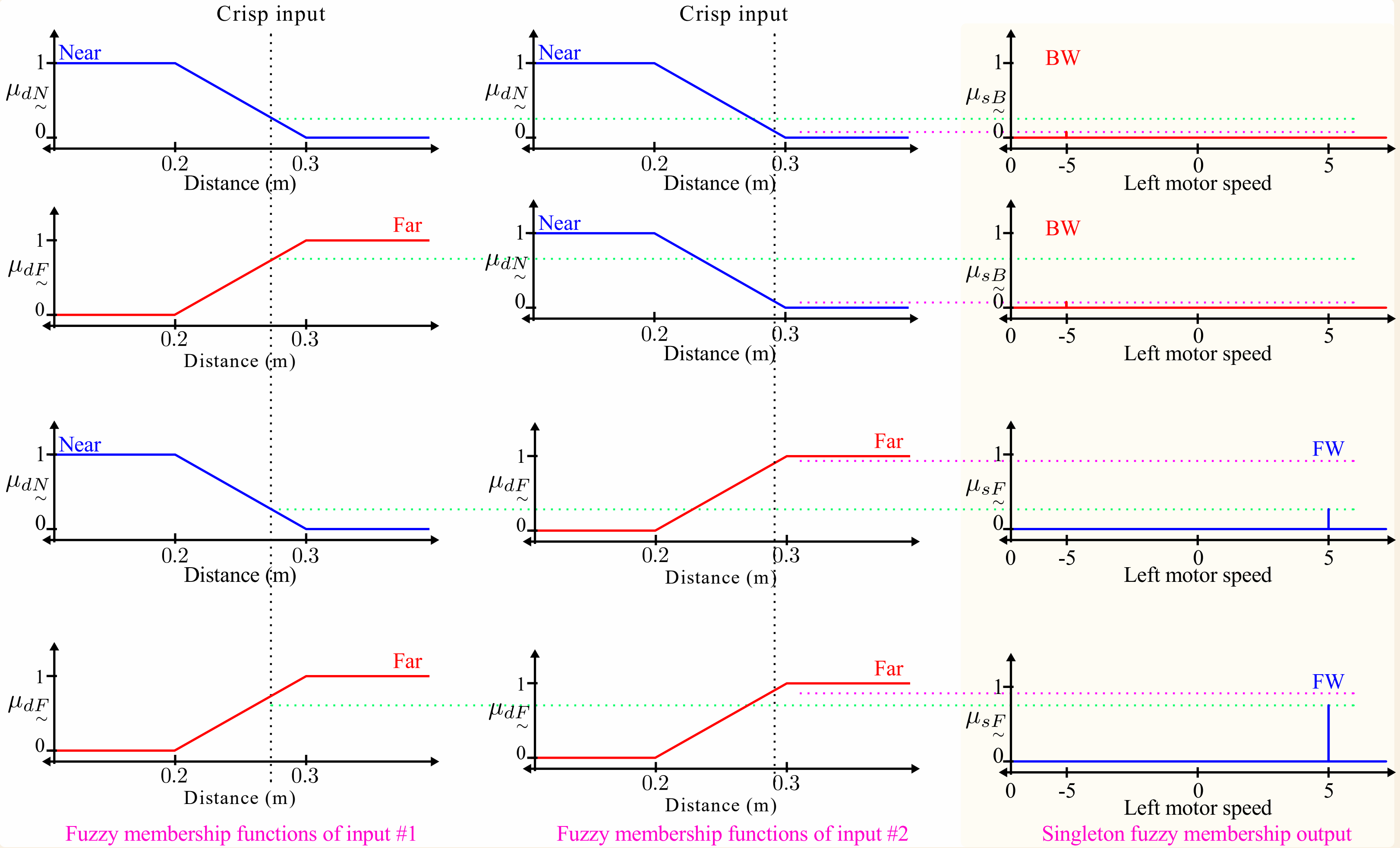
THE OUTPUT'S FIRING DEGREE

Assigning output's firing degree of 2 inputs fuzzy inference rule.

In this case, memberships of input 1 is *0.3 near* or *0.7 far*

While memberships of input 2 is *0.1 near* or *0.9 far*

Visualization of inference Process:



THE OUTPUT'S FIRING DEGREE

Assigning output's firing degree of 2 inputs fuzzy inference rule.
In this case, memberships of input 1 is *0.3 near* or *0.7 far*
While memberships of input 2 is *0.1 near* or *0.9 far*

Left motor output		Left sensor	
		0.3 near	**0.7 far**
Right sensor	**0.1 near**	*(0.1)* -5	*(0.1)* -5
	0.9 far	*(0.3)* 5	*(0.7)* 5

The rule in a table form

DEFUZZIFICATION

Using singleton fuzzy output

WEIGHTED AVERAGE

r-*th* singleton output of the rule

$$z^* = \frac{\sum_r (f_r \cdot z_r)}{\sum_r f_r}$$

Crisp output

r-*th* firing degree of the rule

WEIGHTED AVERAGE

1. **IF** Temp. is (*0.3*) *Cold*, **THEN** PWM output is (*0.3*) *20*
2. **IF** Temp. is (*0.7*) *Warm*, **THEN** PWM output is (*0.7*) *30*
3. **IF** Temp. is (*0.0*) *Hot*, **THEN** PWM output is (*0.0*) *80*

Firing degree of the rule

Singleton output of the rule

$$z^* = \frac{\Sigma_r (f_r \cdot z_r)}{\Sigma_r f_r} = \frac{(0.3 \cdot 20) + (0.7 \cdot 80) + (0.0 \cdot 30)}{0.3 + 0.7 + 0.0} = 27$$

WEIGHTED AVERAGE

Left motor output		Left sensor	
		0.3 **near**	**0.7** **far**
Right sensor	**0.1** **near**	*(0.1)* -5	*(0.1)* -5
	0.9 **far**	*(0.3)* 5	*(0.7)* 5

Firing degree of the rule

Singleton output of the rule

$$
\begin{aligned}
z^* &= \frac{\Sigma_r (f_r \cdot z_r)}{\Sigma_r f_r} \\
&= \frac{(0.1 \cdot -5) + (0.1 \cdot -5) + (0.3 \cdot 5) + (0.7 \cdot 5)}{0.1 + 0.1 + 0.3 + 0.7} \\
&= 3.33
\end{aligned}
$$

PYTHON IMPLENTATION

TRIANGULAR MEMBERSHIP FUNCTION

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

def triangleMF(x, a, b, c, FD0, FD2):
    if x < a:
        FD = FD0
    elif x >= a and x < b:
        FD = (x-a)/(b-a)
    elif x >= b and x < c:
        FD = (c-x)/(c-b)
    elif x >= c:
        FD = FD2
    return FD
```

FUZZIFICATION FUNCTION

```python
# input membership fuzzification
def dis_MF(x):
    near = triangleMF(x, 0.2, 0.2, 0.3, 1, 0)
    far = triangleMF(x, 0.2, 0.3, 0.3, 0, 1)
    y = np.array([[near],
                  [far]],
                  dtype=float)
    return y
lt = ["near","far"]
```

```python
# singleton membership fuzzification of output
singleton_PWM_outputs = np.array([
    [-5],
    [5]
    ],dtype=float)

#  singleton firing degree of output
# initiated with ones
# later, the firing degree will be assigned using min() function
FD_PWM_outputs = np.ones(shape=singleton_PWM_outputs.shape, dtype=float)
```

FUZZIFICATION PROCESS

```
# Fuzzification of input 1 (column)
x1 = 0.27
y1 = dis_MF(x1);
print(f"Left sensor (input#1) {x1} Firing Degree vector is [{y1[0][0]:.2f},{y1
[1][0]:.2f}]")

# Fuzzification of input 2 (row)
x2 = 0.29
y2 = dis_MF(x2);
print(f"Right sensor (input#2) {x2} Firing Degree vector is [{y2[0][0]:.2f},
{y2[1][0]:.2f}]")
```

```
Left sensor (input#1) 0.27 Firing Degree vector is [0.30,0.70]
Right sensor (input#2) 0.29 Firing Degree vector is [0.10,0.90]
```

RULE TABLE

```python
# rule table
# list of indices pointing out to the singleton membership function output index
# here, index 0 is -5 and index 1 is 5
rule_table = np.array(
    [
        [0, 0],
        [1, 1]
        ],
    dtype=int)
```

OUTPUT FIRING DEGREE AND DEFUZZIFICATION

```
# r-th rule firing degree calculation
# and defuzzification
num = 0;
den = 0;
for r, rval in enumerate(y2):
    for c,cval in enumerate(y1):
        tab_idx = rule_table[r][c]

        fd1andfd2 = min(cval,rval)

        FD_PWM_outputs[tab_idx] = fd1andfd2

        print(f"IF input 1 {lt[c]} FD = {cval} AND
        input 2 {lt[r]} FD = {rval}, THEN FD of
        {singleton_PWM_outputs[tab_idx][0]} (MF idx
        {tab_idx}) = {fd1andfd2}")

        num = num + (fd1andfd2*singleton_PWM_outputs
        [tab_idx][0])
        den = den + (fd1andfd2)

crisp_out = num/den
```

```
IF input 1 near FD = [0.3] AND input 2 near FD = [0.1], THEN FD of -5.0 (MF idx 0) = [0.1]
IF input 1 far FD = [0.7] AND input 2 near FD = [0.1], THEN FD of -5.0 (MF idx 0) = [0.1]
IF input 1 near FD = [0.3] AND input 2 far FD = [0.9], THEN FD of 5.0 (MF idx 1) = [0.3]
IF input 1 far FD = [0.7] AND input 2 far FD = [0.9], THEN FD of 5.0 (MF idx 1) = [0.7]
```

```
print(f"Left motor vel output is {crisp_out[0]:.1f}")
```

```
Left motor vel output is 3.3
```

ASSIGNMENT

- Create a membership function for ultrasound reading in CoppeliaSim.
- The number of used sensors is by your design.
- The ultrasound reading is fuzzified into some number of membership functions by your design.
- Put an obstacle in CoppeliaSim.
- Design left and right wheel velocities based on fuzzy inference system to avoid obstacles.
- All the code is done in Python language using only NumPy and Matplotlib libraries.

In the report (PDF file):

- - Plot your fuzzification design of sensors
- - Show your fuzzy rule design in table style

Files to be submitted:

- CoppeliaSim *.ttt file
- Python code *.py
- Report in *.pdf
- Demo video (youtube link)