INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

LABORATORIO FLC

5.4

Alumno

Christian Omar Payán Torróntegui – A01742658

Unidad de Formación

Implementación de Robótica Inteligente

Profesor

Gildardo Sánchez Ante

CAMPUS GUADALAJARA 12/04/2025

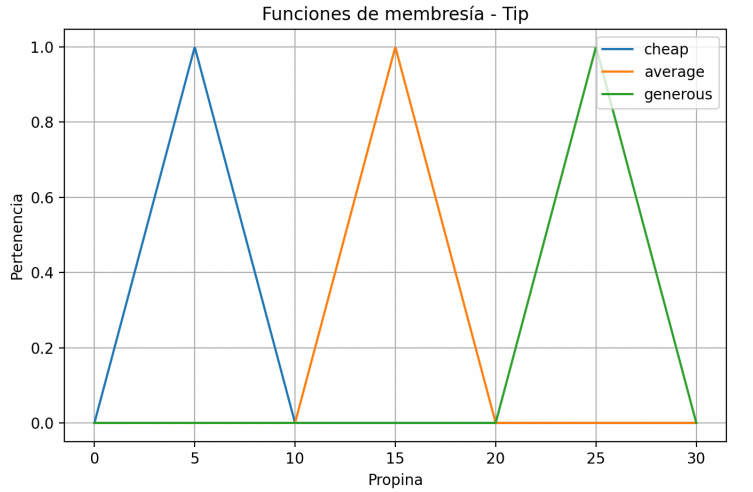
Link al repositorio de Github:
https://github.com/opyntorr/FLC

Utilizando pyfuzzylite en conjunto con la librería matplotlib, se pueden graficar las funciones membresía:

```
import fuzzylite as fl
import matplotlib.pyplot as plt
import numpy as np

# === Definición del motor difuso ===
engine = fl.Engine(
    name="TipCalculator",
    input_variables=[
        fl.InputVariable(
            name="service",
            minimum=0.0,
            maximum=10.0,
            lock_range=False,
            terms=[
                fl.Trapezoid("poor", 0.0, 0.0, 2.5, 5.0),
                fl.Triangle("good", 2.5, 5.0, 7.5),
                fl.Trapezoid("excellent", 5.0, 7.5, 10.0, 10.0),
            ],
        ),
        fl.InputVariable(
            name="food",
            minimum=0.0,
            maximum=10.0,
            lock_range=False,
            terms=[
                fl.Trapezoid("rancid", 0.0, 0.0, 1.0, 3.0),
                fl.Trapezoid("delicious", 7.0, 9.0, 10.0, 10.0),
            ],
        )
    ],
    output_variables=[
        fl.OutputVariable(
            name="tip",
            minimum=0.0,
            maximum=30.0,
            lock_range=False,
            lock_previous=False,
            default_value=fl.nan,
            aggregation=fl.Maximum(),
            defuzzifier=fl.Centroid(100),
```

```
            terms=[
                fl.Triangle("cheap", 0.0, 5.0, 10.0),
                fl.Triangle("average", 10.0, 15.0, 20.0),
                fl.Triangle("generous", 20.0, 25.0, 30.0),
            ],
        )
    ],
    rule_blocks=[
        fl.RuleBlock(
            name="mamdani",
            conjunction=fl.AlgebraicProduct(),
            disjunction=fl.AlgebraicSum(),
            implication=fl.AlgebraicProduct(),
            activation=fl.General(),
            rules=[
                fl.Rule.create("if service is poor or food is rancid then tip is
cheap"),
                fl.Rule.create("if service is good then tip is average"),
                fl.Rule.create("if service is excellent or food is delicious then
tip is generous"),
            ],
        )
    ]
)

# === Probar el motor con valores específicos ===
engine.input_variable("service").value = 2.0
engine.input_variable("food").value = 7.0
engine.process()

print("y =", engine.output_variable("tip").value)             # valor numérico
print("ỹ =", engine.output_variable("tip").fuzzy_value())     # valor difuso
(porcentajes de pertenencia)

# === Graficar las funciones de membresía de 'tip' ===
tip = engine.output_variable("tip")
x = np.linspace(tip.minimum, tip.maximum, 1000)

plt.figure(figsize=(8, 5))
for term in tip.terms:
    y = [term.membership(xi) for xi in x]
    plt.plot(x, y, label=term.name)

plt.title("Funciones de membresía - Tip")
plt.xlabel("Propina")
plt.ylabel("Pertenencia")
plt.legend()
plt.grid(True)
```

```
plt.show()
```



Utilizando el código proporcionado, se obtiene la siguiente propina y sus valores de membresía:

```
y= 4.998950209958008
ỹ = 1.000/cheap + 0.000/average + 0.000/generous
(base) opyntorr@MacBook-Air-de-Omar Python %
```

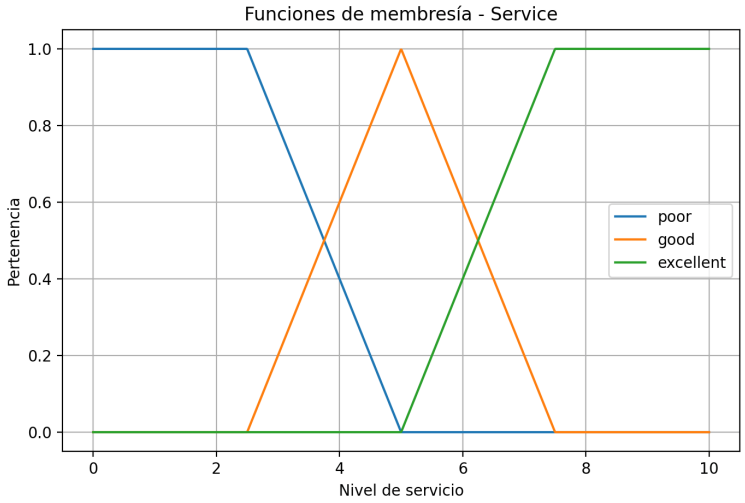
Ejercicio 1

Cambie la definición de términos para el caso del servicio por el siguiente:
term: poor Gaussian 0.000 1.500
term: good Gaussian 5.000 1.500
term: excellent Gaussian 10.000 1.500
Ejecute su Código. ¿Qué diferencia nota?

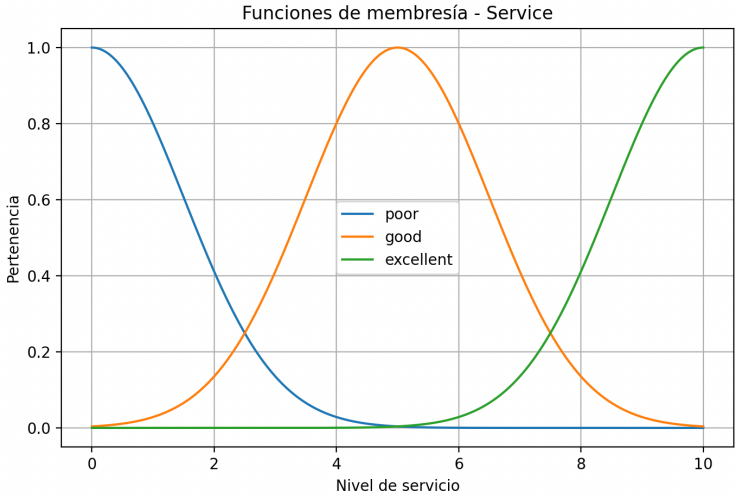
```
y = 7.4747515564301255
ỹ = 0.411/cheap + 0.135/average + 0.000/generous
```

Al ejecutar el código, se puede apreciar que la propina pasa de pertenecer simplemente a barato, para pertenecer ahora también a average. Lo que sí me parece un poco raro es que ahora la suma de las pertenencias no suma 1. Seguramente es porque exista una brecha entre poor y good, ya que el centro de por está en 0, mientras que el centro de good está en 5.

Antes las funciones membresía para service se veían así:



Mientras que ahora, se ven así:



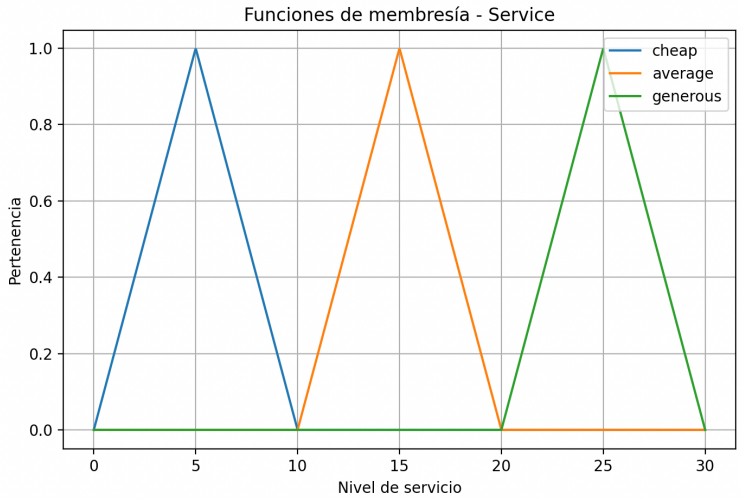
Diría que, de alguna manera, good está más extenido ahora en comparación con la función triángulo, a la par que poor y excelente se encuentran más alejados. Calificando el servicio con un 2, ahora el 2 se encuentra tanto bajo good como bajo poor, y en ninguna de las 2 hay una preferencia de 1, sino que se reparten y su suma no llega a 1. Anteriormente, con las funciones trapecio y triángulo, 2 se encontraba completamente bajo una preferencia de 1, sin tener ninguna preferencia dentro de Good.

Ejercicio 2

Cambie la definición de términos de la propina, siga usando triángulos, pero ahora haga que se de un cierto traslape entre los términos.

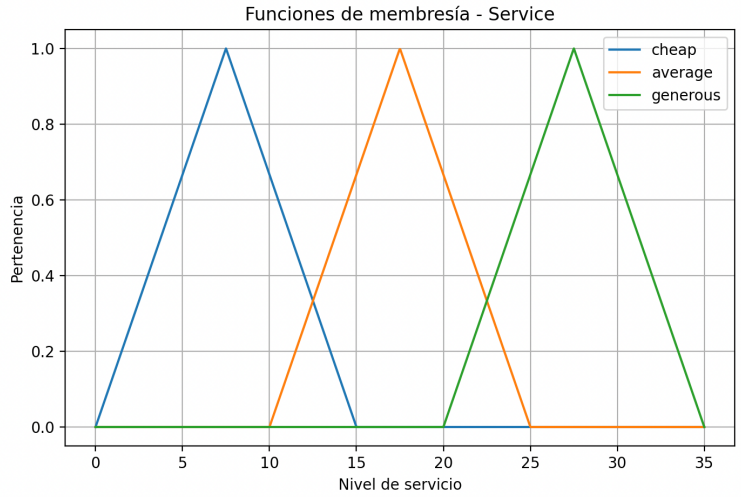
Corra de nuevo su código y comente los resultados.

Actualmente, las funciones de membresía para propina se ven de la siguiente manera:



Ahora, probando con los siguientes valores:

```
fl.Triangle("cheap", 0.0, 7.5, 15.0),
fl.Triangle("average", 10.0, 17.5, 25.0),
fl.Triangle("generous", 20.0, 27.5, 35.0),
```



Y, siguiendo usando las funciones gaussianas para calificar el servicio, obtenemos los siguientes resultados:

```
y = 9.847520321864693
ỹ = 0.411/cheap + 0.135/average + 0.000/generous
```

El valor crisp (y) aumentó ligeramente debido a que antes el máximo era 30, mientras que ahora, para poder hacer los traslapes, aumenté el valor máximo a 35; pero las reglas, entradas y resultados difusos no cambiaron porque la lógica es la misma. Al estar las funciones ahora más separadas y abiertas, el promedio de la figura formada termina más a la derecha (un valor más grande).

Ejercicio 3

Revise cómo este mismo ejercicio se puede resolver usando la librería scikit-fuzzy (skfuzzy), a través de este link:
https://pythonhosted.org/scikit-fuzzy/auto_examples/plot_tipping_problem_newapi.html
Note que en este caso se hace uso del API, lo que permite automatizar muchas partes del proceso, aunque también se pueden hacer manualmente (consulte por ejemplo este otro link:
https://pythonhosted.org/scikit-fuzzy/auto_examples/plot_tipping_problem.html
Ahora en este caso, cambie los términos para la propina haciéndolos gaussianos, siguiendo estas definiciones:

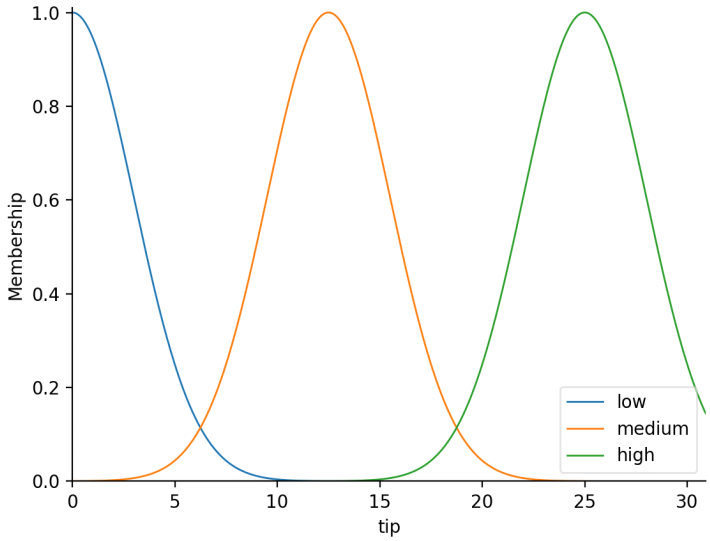
- *term: low Gaussian 0.000 3.000*
- *term: medium Gaussian 12.5 3.00*
- *term: high Gaussian 25.000 3.00*

Pruebe nuevamente y comente los resultados.

En lugar de escribir todo manualmente, se utilizó la API para facilitar la creación y simulación del sistema, con la misma lógica de programación orientada a objetos. No se necesita caluclar área bajo la curva manualmente ni nada.

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

# === Variables de entrada ===
service = ctrl.Antecedent(np.arange(0, 11, 0.1), 'service')
food = ctrl.Antecedent(np.arange(0, 11, 0.1), 'food')

# === Variable de salida (con funciones gaussianas) ===
tip = ctrl.Consequent(np.arange(0, 31, 0.1), 'tip')

# === Funciones de membresía para inputs ===
service['poor'] = fuzz.trimf(service.universe, [0, 0, 5])
service['good'] = fuzz.trimf(service.universe, [0, 5, 10])
service['excellent'] = fuzz.trimf(service.universe, [5, 10, 10])

food['rancid'] = fuzz.trimf(food.universe, [0, 0, 5])
food['delicious'] = fuzz.trimf(food.universe, [5, 10, 10])
```

```python
# === Funciones gaussianas para la salida tip ===
tip['low'] = fuzz.gaussmf(tip.universe, 0.0, 3.0)
tip['medium'] = fuzz.gaussmf(tip.universe, 12.5, 3.0)
tip['high'] = fuzz.gaussmf(tip.universe, 25.0, 3.0)

# === Reglas ===
rule1 = ctrl.Rule(service['poor'] | food['rancid'], tip['low'])
rule2 = ctrl.Rule(service['good'], tip['medium'])
rule3 = ctrl.Rule(service['excellent'] | food['delicious'], tip['high'])

# === Sistema de control ===
tipping_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])
tipping = ctrl.ControlSystemSimulation(tipping_ctrl)

# === Probar con valores específicos ===
tipping.input['service'] = 2.0
tipping.input['food'] = 7.0

# === Procesar ===
tipping.compute()

# === Resultado ===
print("y =", tipping.output['tip'])

# Obtener valor de salida
y = tipping.output['tip']

# Calcular pertenencias a cada término
μ_low = fuzz.gaussmf(y, 0.0, 3.0)
μ_medium = fuzz.gaussmf(y, 12.5, 3.0)
μ_high = fuzz.gaussmf(y, 25.0, 3.0)

# Imprimir resultados
print(f"y = {y:.2f}")
print(f"Pertenencia a 'low': {μ_low:.3f}")
print(f"Pertenencia a 'medium': {μ_medium:.3f}")
print(f"Pertenencia a 'high': {μ_high:.3f}")


# === Graficar las funciones de membresía de tip ===
tip.view()
plt.show()
```




```
y = 14.630759646059044
y = 14.63
Pertenencia a 'low': 0.000
Pertenencia a 'medium': 0.777
Pertenencia a 'high': 0.003
```

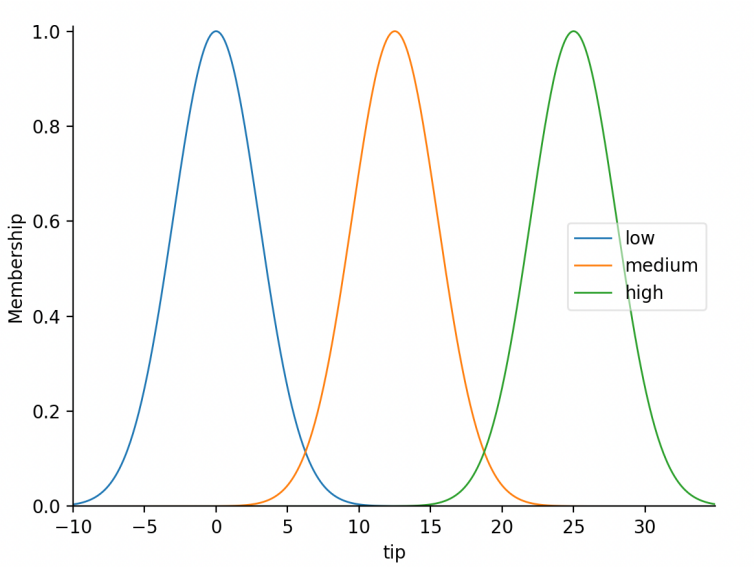
Ahora, fuera de su centro, las funciones gaussianas nunca son cero, lo que extiende y suaviza el área de influencia, aun así, parece ser que el resultado se muestra en 0 para low. Esto solo puede significar un error.

Al ser funciones gaussianas, hace que las gráficas se extiendan más allá de 35 y menos allá del 0, como fue el caso con las funciones triangulares solapadas.

Algo muy curioso al estar trabajado con esta librería recae en esta línea:

```
tip = ctrl.Consequent(np.arange(-10, 35, 0.1), 'tip')
```

Posteriormente, la tenía limitada de 0 a 31, y es curioso, pues hace que todos los cálculos solo se evalúen dentro de ese rango, todo lo demás no se evalúa ni considera. Por tanto, tomando esto en cuenta, obtenemos los siguientes resultados:




```
y = 11.28491427874488
y = 11.28
Pertenencia a 'low': 0.001
Pertenencia a 'medium': 0.921
Pertenencia a 'high': 0.000
```

La pertenencia a high es de 0, pero probablemente sea por mero redondeo, puesto que el valor de pertenencia en low apenas es de 0.001, estando en 11, mucho más cerca de low que de high.

En teoría, al ser funciones gaussianas, aunque tengan una pertenencia, por ejemplo, más arraigada a la clase medium, también arrastra más de low y un poco de high en comparación con la clase triangular. Ahora pasó a representar una clase diferente, que seguramente se deba al cambio de la forma del área combinada tras aplicar las reglas.

Ejercicio 4

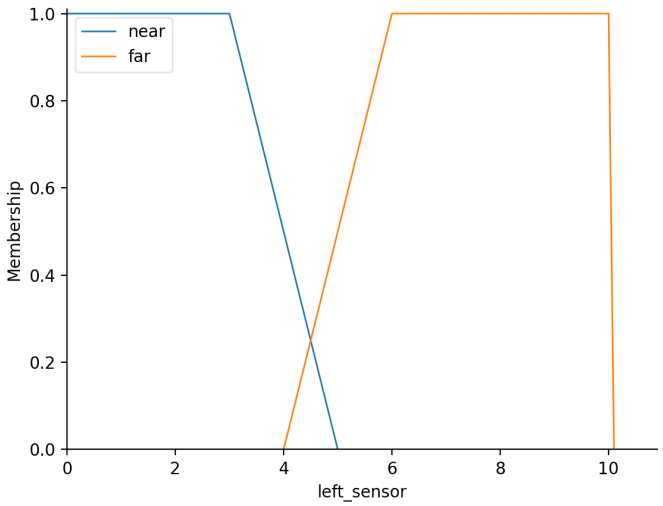
Ahora consideremos el problema de controlar un pequeño robot móvil.

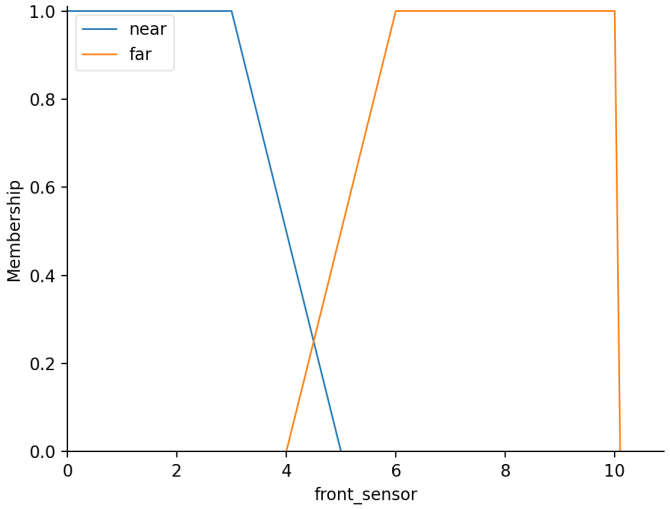
```python
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
import matplotlib.pyplot as plt

# === Universos ===
sensor_range = np.arange(0, 11, 0.1)  # 0 = cerca, 10 = lejos
motor_range = np.arange(0, 10, 0.1)   # 0 = reversa, 9 = adelante
```

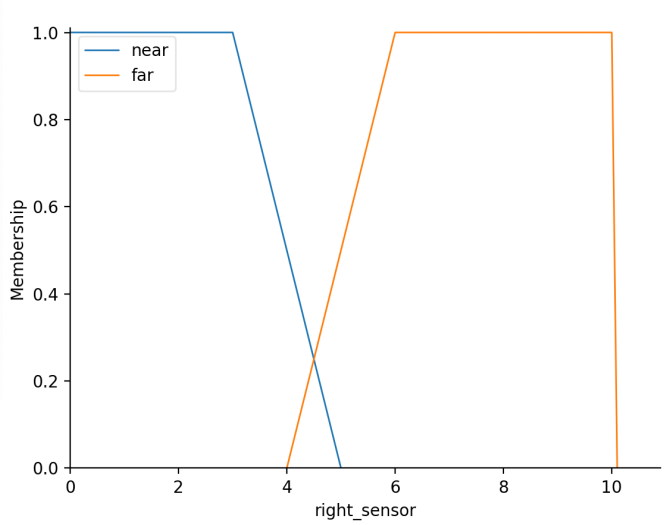
```python
# === Variables de entrada ===
left_sensor = ctrl.Antecedent(sensor_range, 'left_sensor')
front_sensor = ctrl.Antecedent(sensor_range, 'front_sensor')
right_sensor = ctrl.Antecedent(sensor_range, 'right_sensor')

# === Variables de salida ===
left_motor = ctrl.Consequent(motor_range, 'left_motor')
right_motor = ctrl.Consequent(motor_range, 'right_motor')

# === Funciones de membresía para sensores ===
left_sensor['near'] = fuzz.trapmf(sensor_range, [0, 0, 3, 5])
left_sensor['far']  = fuzz.trapmf(sensor_range, [4, 6, 10, 10])
front_sensor['near'] = fuzz.trapmf(sensor_range, [0, 0, 3, 5])
front_sensor['far']  = fuzz.trapmf(sensor_range, [4, 6, 10, 10])
right_sensor['near'] = fuzz.trapmf(sensor_range, [0, 0, 3, 5])
right_sensor['far']  = fuzz.trapmf(sensor_range, [4, 6, 10, 10])

# === Funciones de membresía para motores ===
left_motor['reverse'] = fuzz.trapmf(motor_range, [6, 7, 9, 9])
left_motor['forward'] = fuzz.trapmf(motor_range, [0, 0, 3, 4])
right_motor['reverse'] = fuzz.trapmf(motor_range, [6, 7, 9, 9])
right_motor['forward'] = fuzz.trapmf(motor_range, [0, 0, 3, 4])

# === Reglas de comportamiento reactivo (según tabla del artículo de Rusu) ===
rules = [
    ctrl.Rule(left_sensor['far'] & front_sensor['far'] & right_sensor['far'],
              (left_motor['forward'], right_motor['forward'])),

    ctrl.Rule(left_sensor['far'] & front_sensor['far'] & right_sensor['near'],
              (left_motor['reverse'], right_motor['forward'])),

    ctrl.Rule(left_sensor['far'] & front_sensor['near'] & right_sensor['near'],
              (left_motor['reverse'], right_motor['forward'])),

    ctrl.Rule(left_sensor['far'] & front_sensor['near'] & right_sensor['far'],
              (left_motor['forward'], right_motor['reverse'])),

    ctrl.Rule(left_sensor['near'] & front_sensor['near'] & right_sensor['far'],
              (left_motor['forward'], right_motor['reverse'])),

    ctrl.Rule(left_sensor['near'] & front_sensor['near'] & right_sensor['near'],
              (left_motor['reverse'], right_motor['reverse'])),

    ctrl.Rule(left_sensor['near'] & front_sensor['far'] & right_sensor['near'],
              (left_motor['forward'], right_motor['reverse'])),

    ctrl.Rule(left_sensor['near'] & front_sensor['far'] & right_sensor['far'],
              (left_motor['forward'], right_motor['reverse'])),
```

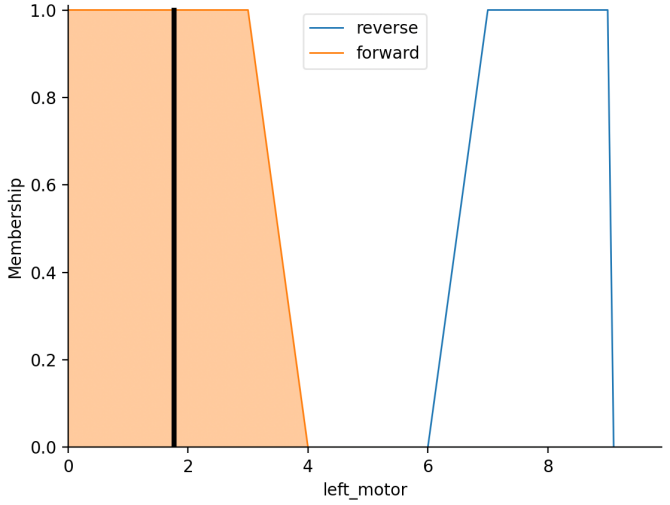
```
]

# === Sistema de control ===
robot_ctrl = ctrl.ControlSystem(rules)
robot_sim = ctrl.ControlSystemSimulation(robot_ctrl)

# Simulación con sensores activados
robot_sim.input['left_sensor'] = 3.0   # cerca del obstáculo a la izquierda
robot_sim.input['front_sensor'] = 2.0  # muy cerca enfrente
robot_sim.input['right_sensor'] = 6.0  # lejos a la derecha

# Ejecutar lógica difusa
robot_sim.compute()

# Mostrar resultados
print(f"Velocidad del motor izquierdo: {robot_sim.output['left_motor']:.2f}")
print(f"Velocidad del motor derecho: {robot_sim.output['right_motor']:.2f}")

# Ver funciones de entrada
left_sensor.view()
front_sensor.view()
right_sensor.view()

# Ver funciones de salida
left_motor.view(sim=robot_sim)
right_motor.view(sim=robot_sim)

plt.show()
```

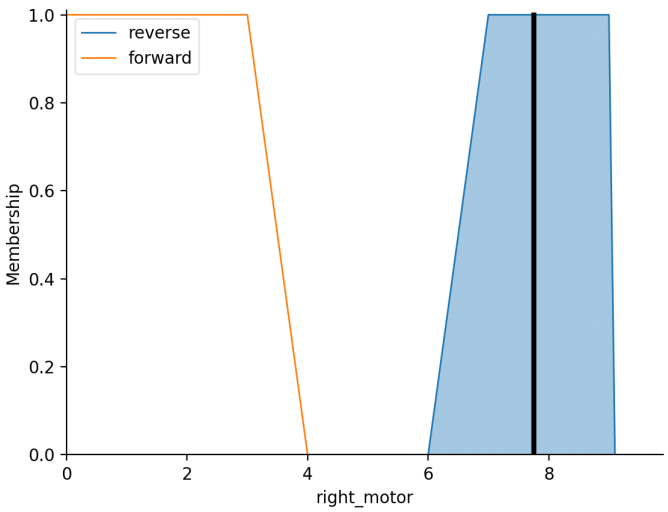
```
Velocidad del motor izquierdo: 1.76
Velocidad del motor derecho: 7.76
```




near
far
Membership
front_sensor


near
far
Membership
right_sensor


reverse
forward
Membership
left_motor



Con los parámetros de entrada que se usaron, significa que el robot tiene un obstáculo a la izquierda a 3 unidades, activando near; un obstáculo al frente a 2 unidades, también activando near; y un espacio libre a la derecha de 6 unidades, activando far.

Con esto, se activa esta regla:

```
ctrl.Rule(left_sensor['near'] & front_sensor['near'] & right_sensor['far'],
              (left_motor['forward'], right_motor['reverse'])),
```

Esta regla hace que se dé reversa con ambos motores.

El código regresa un valor de 7.76 para ambos motores que, por las funciones de membresía que se utilizan en el código, se traducen en reversa:

```
left_motor['reverse'] = fuzz.trapmf(motor_range, [6, 7, 9, 9])
left_motor['forward'] = fuzz.trapmf(motor_range, [0, 0, 3, 4])
right_motor['reverse'] = fuzz.trapmf(motor_range, [6, 7, 9, 9])
right_motor['forward'] = fuzz.trapmf(motor_range, [0, 0, 3, 4])
```

Por ejemplo, con valores de los sensores lejanos:

```
robot_sim.input['left_sensor'] = 8.0
robot_sim.input['front_sensor'] = 8.0
robot_sim.input['right_sensor'] = 8.0
```

Obtenemos lo siguiente:

```
Velocidad del motor izquierdo: 1.76
Velocidad del motor derecho: 1.76
```

Que se traduce en vleocidad hacia adelante para ambos motores.

Otro ejemplo donde ambos motores dan avance en sentido contrario:

```
robot_sim.input['left_sensor'] = 2.0
robot_sim.input['front_sensor'] = 8.0
robot_sim.input['right_sensor'] = 8.0
```

```
Velocidad del motor izquierdo: 1.76
Velocidad del motor derecho: 7.76
```

TABLE III
FUZZY RULES FOR THE ROBOT MOVEMENT

Left_Sensor	Frot_Sensor	Right_Sensor	Left_Motor	Right_Motor
Far	Far	Far	Forward	Forward
Far	Far	Near	Reverse	Forward
Far	Near	Near	Reverse	Forward
Far	Near	Far	Forward	Reverse
Near	Near	Far	Forward	Reverse
Near	Near	Near	Reverse	Reverse
Near	Far	Near	Forward	Reverse
Near	Far	Far	Forward	Reverse

Esta tabla de reglas difusas permite que el robot tome decisiones aun con incertidumbre, combina varias entradas sensoriales para determinar la mejor opción del momento y, puede que lo más importante, genera transiciones suaves en las velocidades de los motores, en lugar de comportamientos bruscos. Lo único un tanto extraño es la condición de Near Near Far, en la que prefiere usar la reversa para ambas ruedas