Importaciones de liberias

In [ ]:
```
import pygame
import sys
import math

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

Inicializar PyGame, para crear la pantalla y excenario

In [ ]:
```
# # Inicializar Pygame
# pygame.init()

# # Dimensiones de la pantalla
# screen_width = 800
# screen_height = 600

# # Colores
# white = (255, 255, 255)
# red = (255, 0, 0)
# green = (0, 180, 0)

# # Inicializar la pantalla
# screen = pygame.display.set_mode((screen_width, screen_height))
# pygame.display.set_caption("Simulación de Robot en el Mundial de Soccer")
```

Inicializar posicion de los tres elementos principales (pelota, robot y porteria)

In [ ]:
```
# Posición inicial del robot
robot_x = 700
robot_y = 300
# Posición inicial de la pelota
pelota_x = 500
pelota_y = 200
# Posición inicial de la porteria objetivo
porteria_x = 50
porteria_y = 300
```

Logica Difusa para encontrar la pelota

Variables de entrada / Crisp

In [ ]:
```
#Variables CRISP
distancia = ctrl.Antecedent(np.arange(0, 701, 1), 'distancia')
# Definir las funciones de membresía para distancia
distancia['cerca'] = fuzz.trimf(distancia.universe, [0, 125, 250])
distancia['media'] = fuzz.trimf(distancia.universe, [240, 370, 500])
distancia['lejos'] = fuzz.trimf(distancia.universe, [480, 590, 700])
```

```
resistencia = ctrl.Antecedent(np.arange(0, 101, 1), 'resistencia')
# Definir las funciones de membresía para distancia
resistencia['baja'] = fuzz.trimf(resistencia.universe, [0, 17, 34])
resistencia['normal'] = fuzz.trimf(resistencia.universe, [30, 50, 70])
resistencia['alta'] = fuzz.trimf(resistencia.universe,[66, 83, 100])
```

Varaiable de Salida

In [ ]:
```
# Definir la variable de salida
movimiento = ctrl.Consequent(np.arange(0, 101, 1), 'movimiento')
# Definir las etiquetas lingüísticas y funciones de membresía
movimiento['caminar'] = fuzz.trimf(movimiento.universe, [0, 17, 34])
movimiento['trotar'] = fuzz.trimf(movimiento.universe, [30, 50, 70])
movimiento['correr'] = fuzz.trimf(movimiento.universe, [66, 83, 100])
```

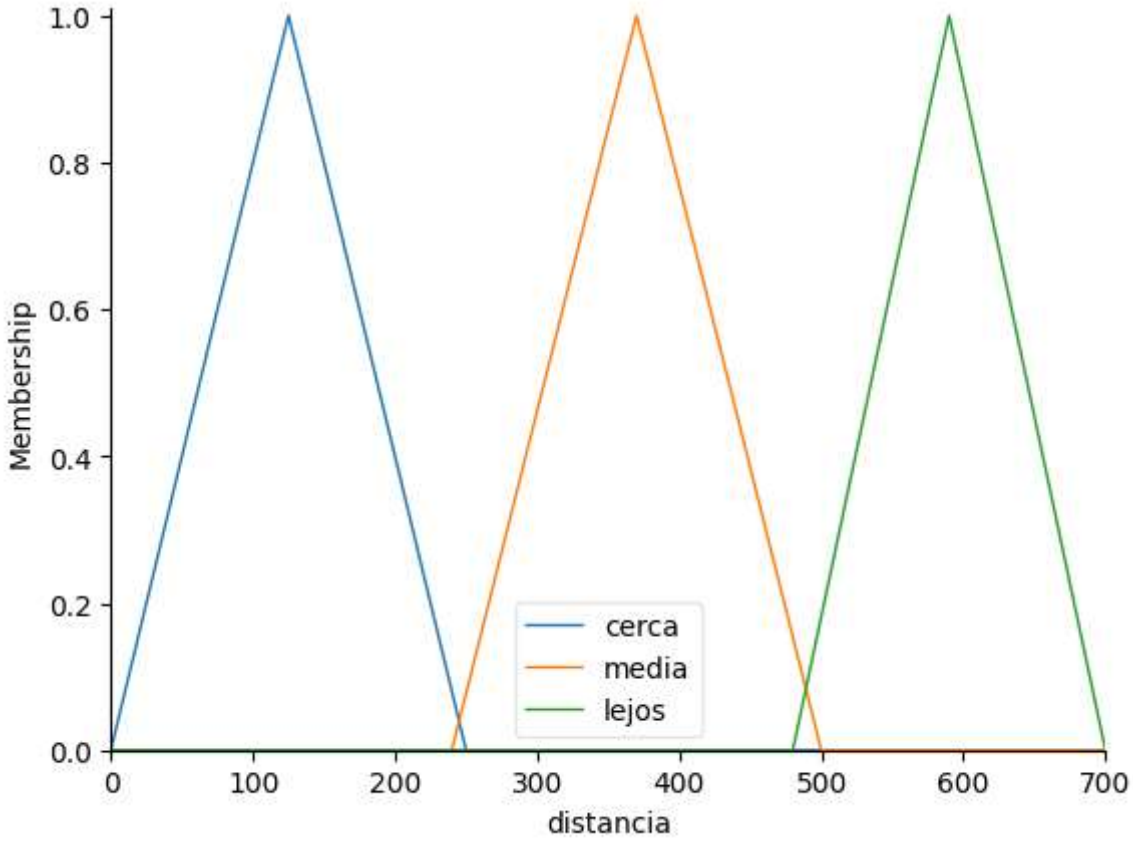
Clausulas de Horn

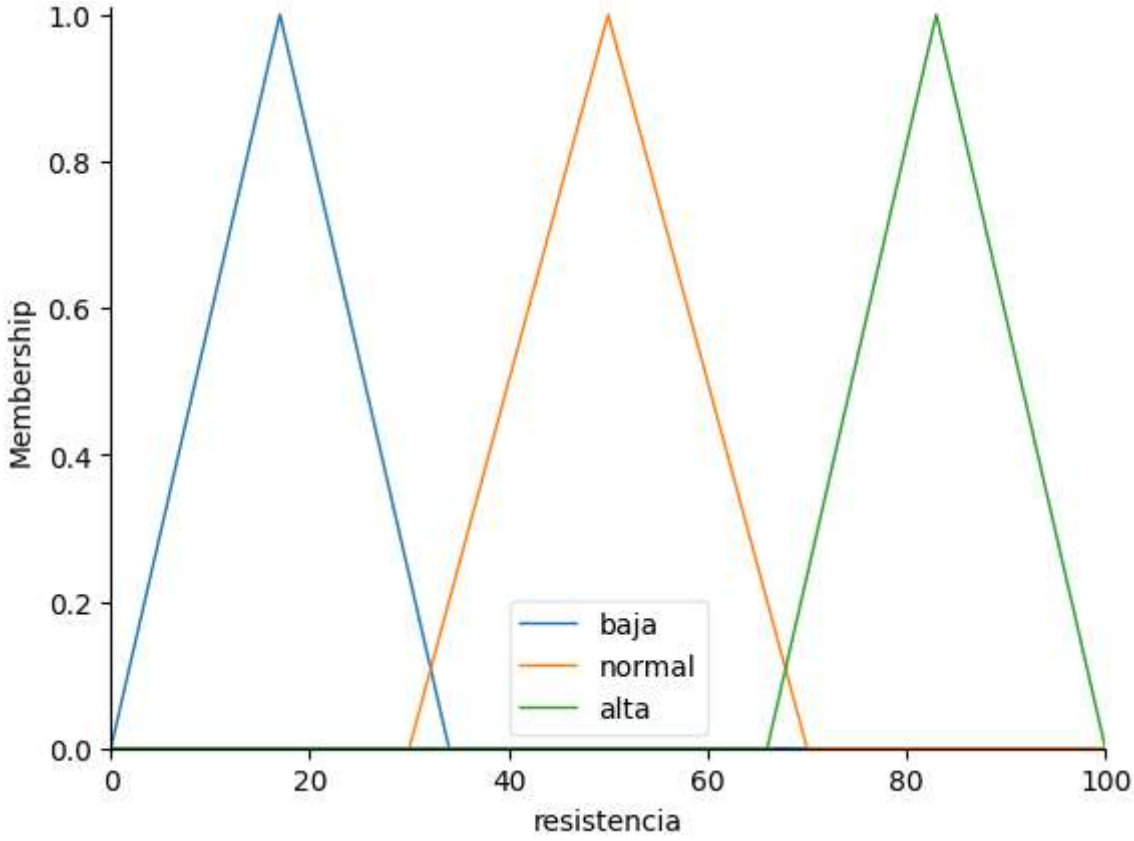
In [ ]:
```
#Clausulas de Horn / Definir las reglas difusas
regla1 = ctrl.Rule(distancia['cerca'] & resistencia['baja'], movimiento['caminar'])
regla2 = ctrl.Rule(distancia['cerca'] & resistencia['normal'], movimiento['trotar']
regla3 = ctrl.Rule(distancia['cerca'] & resistencia['alta'], movimiento['correr'])
regla4 = ctrl.Rule(distancia['media'] & resistencia['baja'], movimiento['caminar'])
regla5 = ctrl.Rule(distancia['media'] & resistencia['normal'], movimiento['trotar']
regla6 = ctrl.Rule(distancia['media'] & resistencia['alta'], movimiento['correr'])
regla7 = ctrl.Rule(distancia['lejos'] & resistencia['baja'], movimiento['caminar'])
regla8 = ctrl.Rule(distancia['lejos'] & resistencia['normal'], movimiento['trotar']
regla9 = ctrl.Rule(distancia['lejos'] & resistencia['alta'], movimiento['correr'])
```

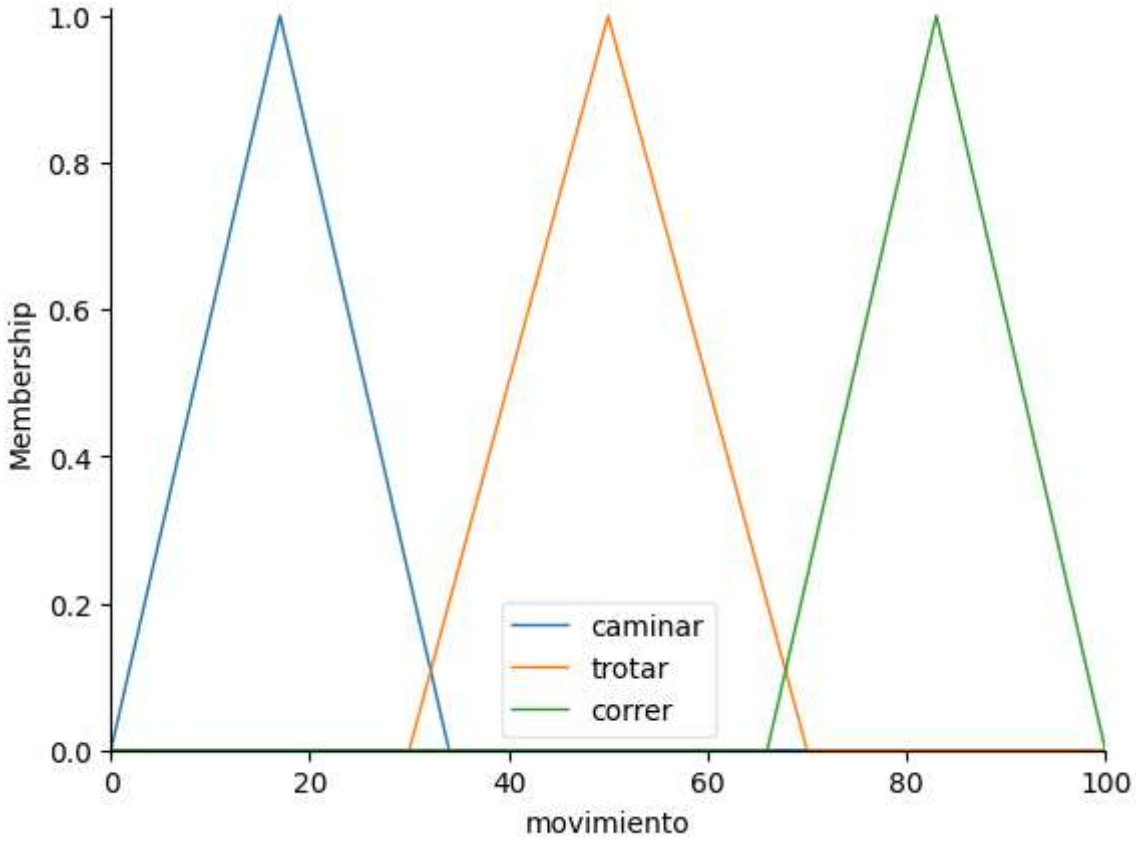
In [ ]:
```
# Crear el sistema de control
sistema_de_control = ctrl.ControlSystem([regla1, regla2, regla3, regla4, regla5, re
sistema = ctrl.ControlSystemSimulation(sistema_de_control)
```

Graficas (distancia, resistencia, movimiento)

In [ ]:
```
distancia.view()
resistencia.view()
movimiento.view()
```


1.0
0.8
0.6
0.4
0.2
0.0
Membership
cerca
media
lejos
0
100
200
300
400
500
600
700
distancia


1.0
0.8
0.6
0.4
0.2
0.0
Membership
baja
normal
alta
0
20
40
60
80
100
resistencia



Calculos

In [ ]:
```
distancia_robot_pelota = ((robot_x - pelota_x) ** 2 + (robot_y - pelota_y) ** 2) **

# Definir valores de entrada
sistema.input['distancia'] = distancia_robot_pelota
sistema.input['resistencia'] = 10

# Calcular la salida difusa
sistema.compute()

result_1 = ''

if sistema.output['movimiento'] > 66:
    result_1= "Correr"
elif sistema.output['movimiento'] > 33:
    result_1= "Trotar"
else:
    result_1= "Caminar"

print(f"Accion: {result_1} - ({sistema.output['movimiento']})")
```

```
Accion: Caminar - (16.999999999999996)
```

Logica Difusa para calcular fuerza para patear la pelota

Variables de entrada / Crisp

In [ ]:
```
distancia_porteria = ctrl.Antecedent(np.arange(0, 701, 1), 'distancia_porteria')
# Definir las funciones de membresía para distancia
distancia_porteria['cerca'] = fuzz.trimf(distancia_porteria.universe, [0, 120, 250]
distancia_porteria['media'] = fuzz.trimf(distancia_porteria.universe, [240, 360, 50
distancia_porteria['lejos'] = fuzz.trimf(distancia_porteria.universe, [480, 600, 70


angulo = ctrl.Antecedent(np.arange(-180, 181, 1), 'angulo')
# Definir las funciones de membresía para la variable angulo
angulo['izquierda'] = fuzz.trimf(angulo.universe, [-180, -120, -60])
angulo['centro'] = fuzz.trimf(angulo.universe, [-60, 0, 60])
angulo['derecha'] = fuzz.trimf(angulo.universe, [60, 120, 180])
```

Varaiable de Salida

In [ ]:
```
# Definir la variable de salida
fuerza = ctrl.Consequent(np.arange(0, 101, 1), 'fuerza')
# Definir las etiquetas lingüísticas y funciones de membresía
fuerza['suave'] = fuzz.trimf(fuerza.universe, [0, 17, 34])
fuerza['medio'] = fuzz.trimf(fuerza.universe, [30, 50, 70])
fuerza['fuerte'] = fuzz.trimf(fuerza.universe, [66, 83, 100])
```

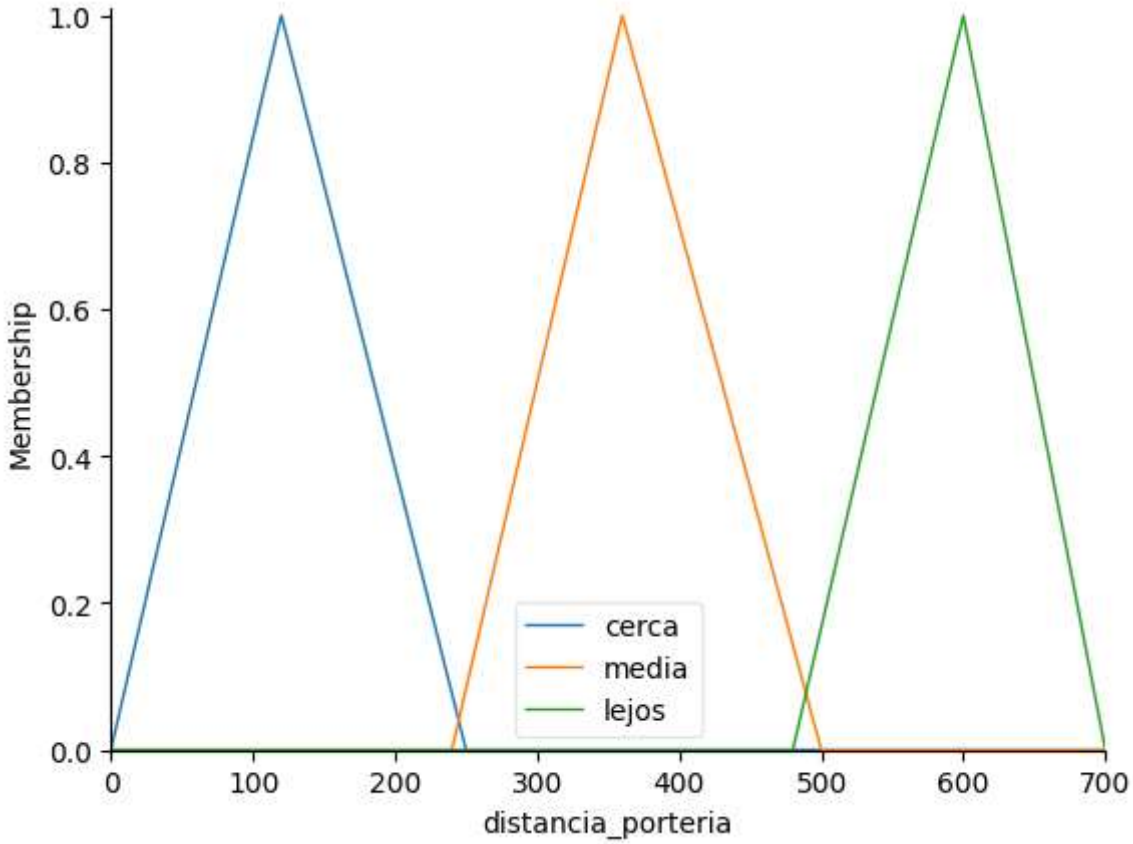
Clausulas de Horn

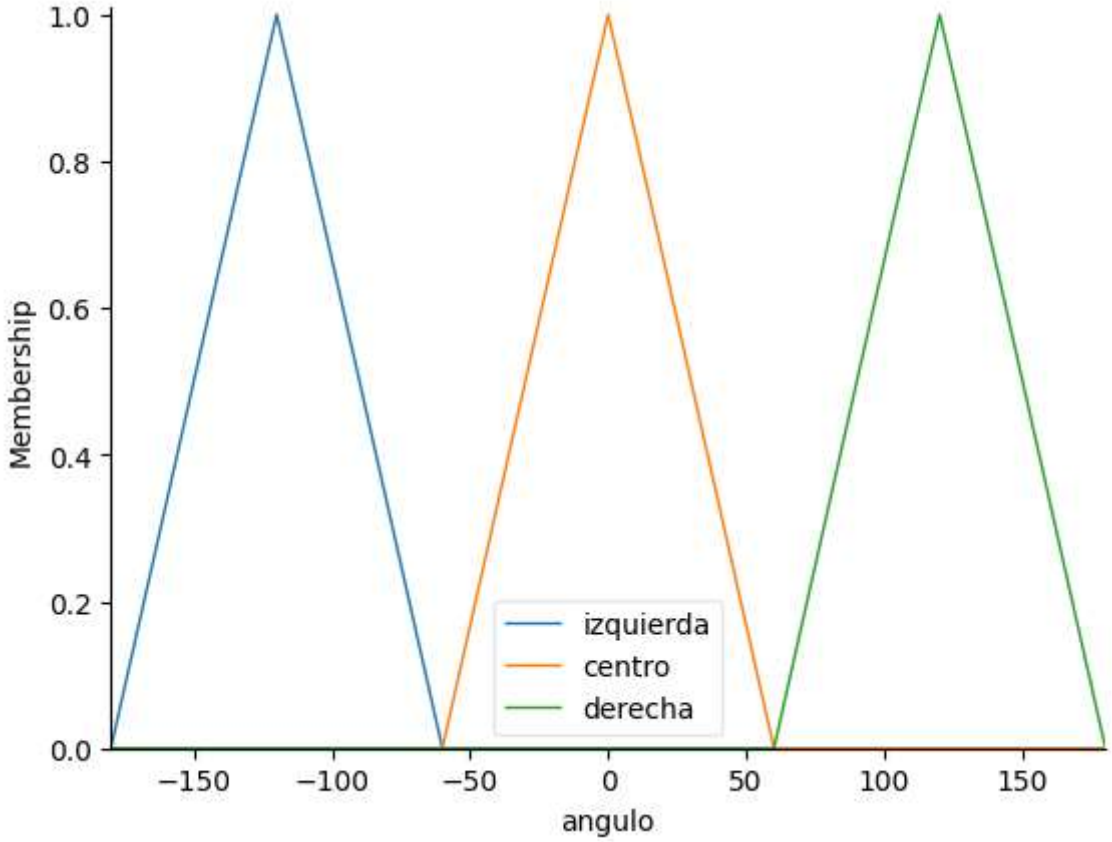
In [ ]:
```
#Clausulas de Horn / Definir las reglas difusas
regla1 = ctrl.Rule(distancia_porteria['cerca'] & angulo['izquierda'], fuerza['medio
regla2 = ctrl.Rule(distancia_porteria['cerca'] & angulo['centro'], fuerza['suave'])
regla3 = ctrl.Rule(distancia_porteria['cerca'] & angulo['derecha'], fuerza['medio']
regla4 = ctrl.Rule(distancia_porteria['media'] & angulo['izquierda'], fuerza['fuert
regla5 = ctrl.Rule(distancia_porteria['media'] & angulo['centro'], fuerza['medio'])
regla6 = ctrl.Rule(distancia_porteria['media'] & angulo['derecha'], fuerza['fuerte'
regla7 = ctrl.Rule(distancia_porteria['lejos'] & angulo['izquierda'], fuerza['fuert
regla8 = ctrl.Rule(distancia_porteria['lejos'] & angulo['centro'], fuerza['fuerte']
regla9 = ctrl.Rule(distancia_porteria['lejos'] & angulo['derecha'], fuerza['fuerte'
```

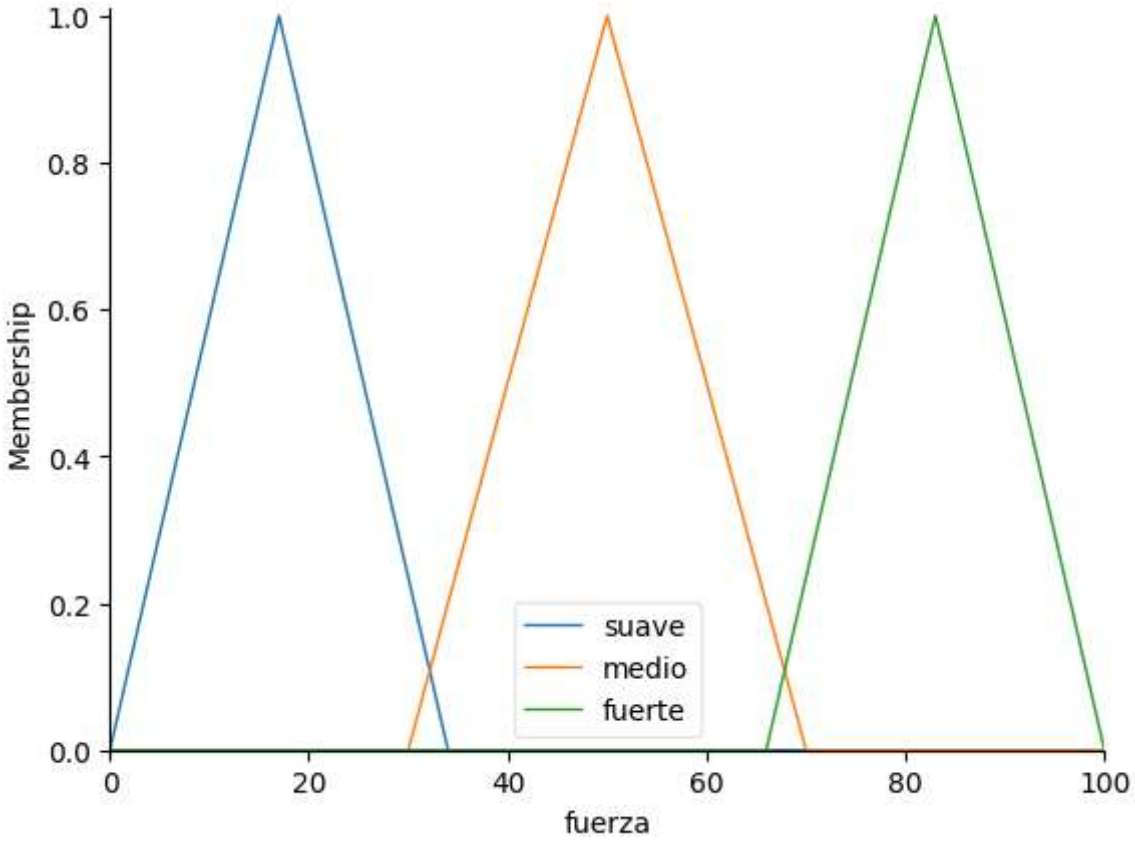
In [ ]:
```
# Crear el sistema de control
sistema_de_control2 = ctrl.ControlSystem([regla1, regla2, regla3, regla4, regla5, r
sistema2 = ctrl.ControlSystemSimulation(sistema_de_control2)
```

Graficas (distancia, angulo, fuerza)

In [ ]:
```
distancia_porteria.view()
angulo.view()
fuerza.view()
```


1.0
0.8
0.6
0.4
0.2
0.0
Membership
cerca
media
lejos
0
100
200
300
400
500
600
700
distancia_porteria


1.0
0.8
0.6
0.4
0.2
0.0
Membership
izquierda
centro
derecha
−150
−100
−50
0
50
100
150
angulo



Calculos

Como conocemos la posicion de la pelota y la porteria, podemos calcular la fuerza desde antes

In [ ]:
```
# Como conocemos las coordenadas de la porteria y pelota, podemos definir la fuerza
distancia_porteria_pelota_inicial = ((pelota_x - porteria_x) ** 2 + (pelota_y - por
# Calcular el ángulo entre la porteria y la pelota
angulo_porteria = math.atan2(porteria_y - pelota_y, porteria_x - pelota_x)

# Definir valores de entrada
sistema2.input['distancia_porteria'] = distancia_porteria_pelota_inicial
sistema2.input['angulo'] = angulo_porteria

# Calcular la salida difusa
sistema2.compute()

result_2 = ''

if sistema2.output['fuerza'] > 66:
    result_1= "Patear fuerte"
elif sistema2.output['fuerza'] > 33:
    result_1= "Patear normal"
else:
    result_1= "Patear suave"

print(f"Fuerza: {result_1} - ({sistema2.output['fuerza']:.0f}%)")
```

```
Fuerza: Patear normal - (50%)
```