In [2]:
```
%pip install scikit-fuzzy
```

```
Requirement already satisfied: scikit-fuzzy in /home/hugo/Documents/Tec/IA/U1-NLP/.env/lib/py
thon3.10/site-packages (0.5.0)
Note: you may need to restart the kernel to use updated packages.
```

# Introducción al Control Difuso para un Seguidor de Línea

La lógica difusa es una técnica de inteligencia artificial que permite manejar la incertidumbre y la imprecisión en los datos, lo que la hace ideal para aplicaciones en robótica y control automático.

## Objetivos

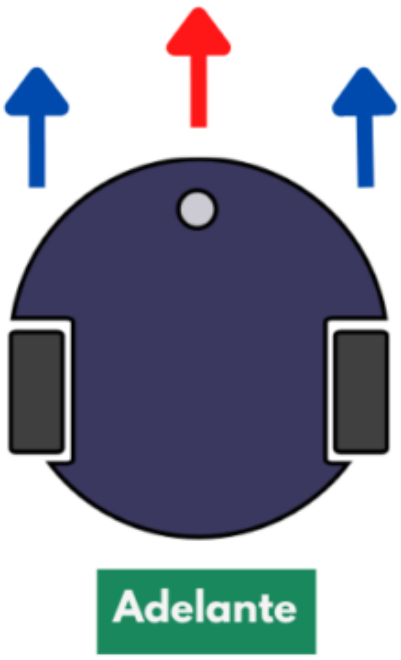
- Definir las variables de entrada y salida del sistema de control.
- Crear funciones de pertenencia para las variables utilizando diferentes tipos de funciones (triangulares, trapezoidales, gaussianas, etc.).
- Establecer reglas difusas que describan el comportamiento del sistema.
- Simular el sistema de control difuso y visualizar los resultados.

## Descripción del Problema

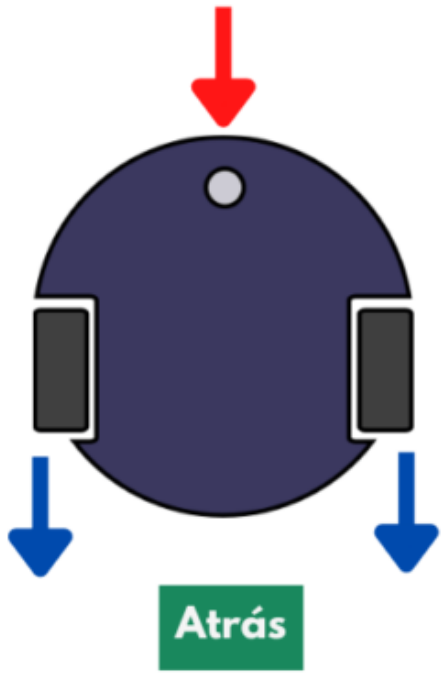
Un seguidor de línea es un robot que sigue una línea marcada en el suelo. Para lograr esto, el robot utiliza sensores que detectan la posición relativa de la línea. En este caso, utilizaremos tres sensores (izquierdo, central y derecho) que proporcionan información sobre la posición de la línea. Basándonos en las lecturas de estos sensores, el sistema de control difuso determinará el ángulo de giro necesario para mantener el robot en la trayectoria deseada.

*Giro en configuración diferencial*

*Giro en configuración ackerman*

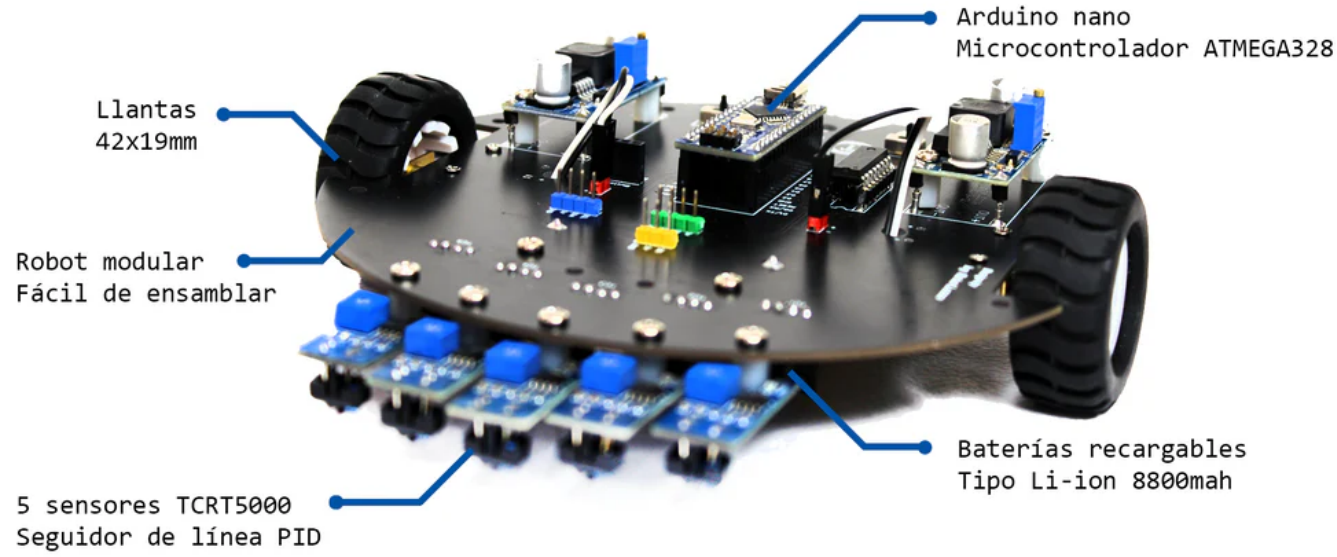


Adelante

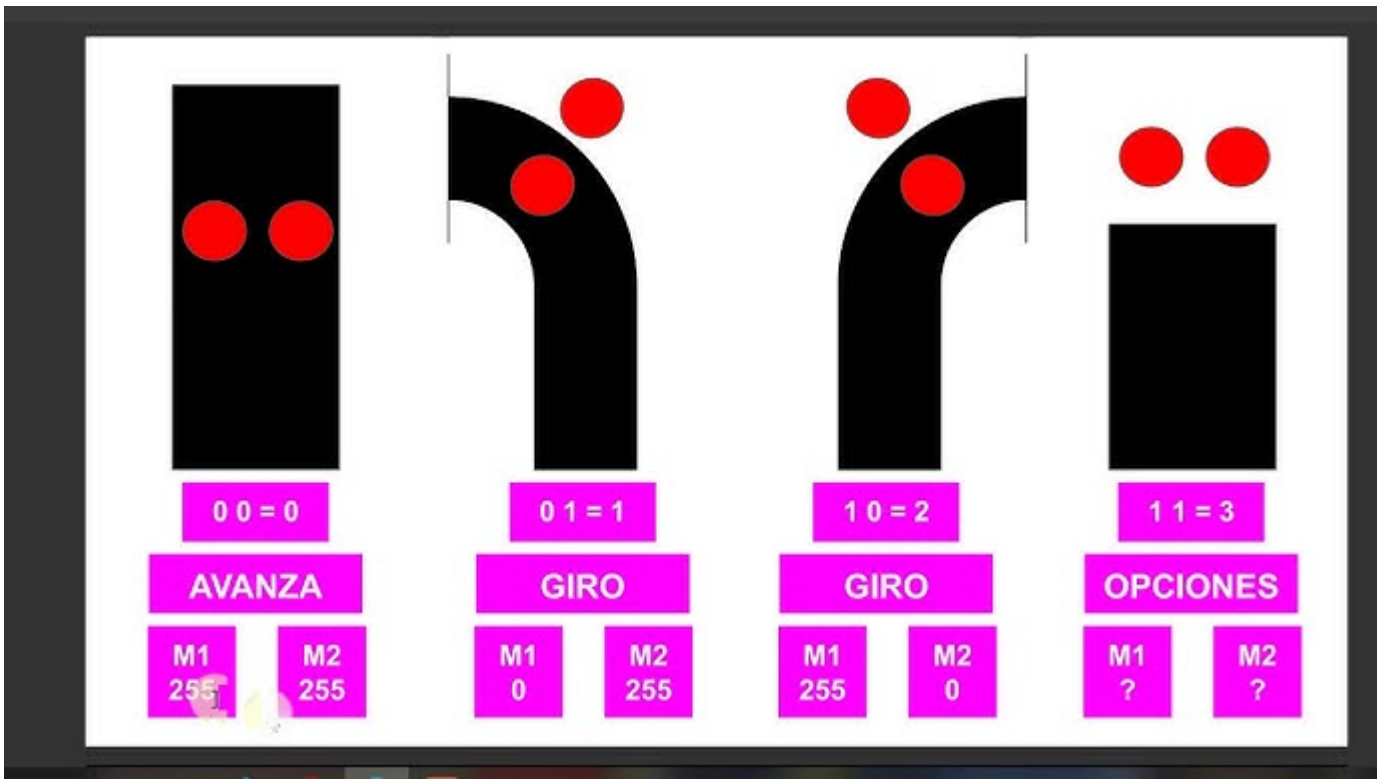


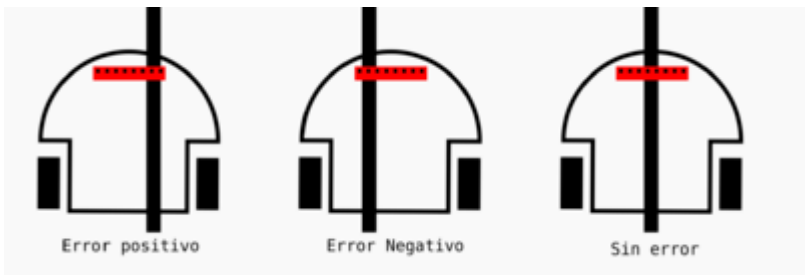
Atrás

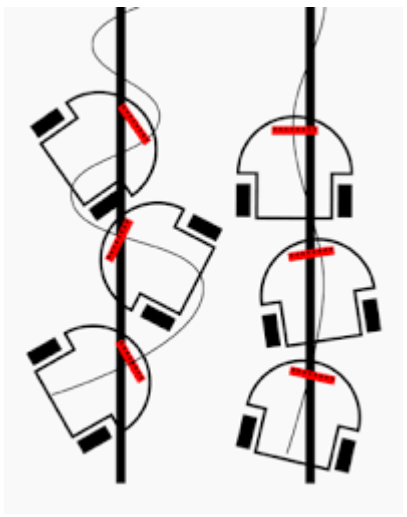
Talos
Electronics




www.taloselectronics.com



## Contenido del Notebook

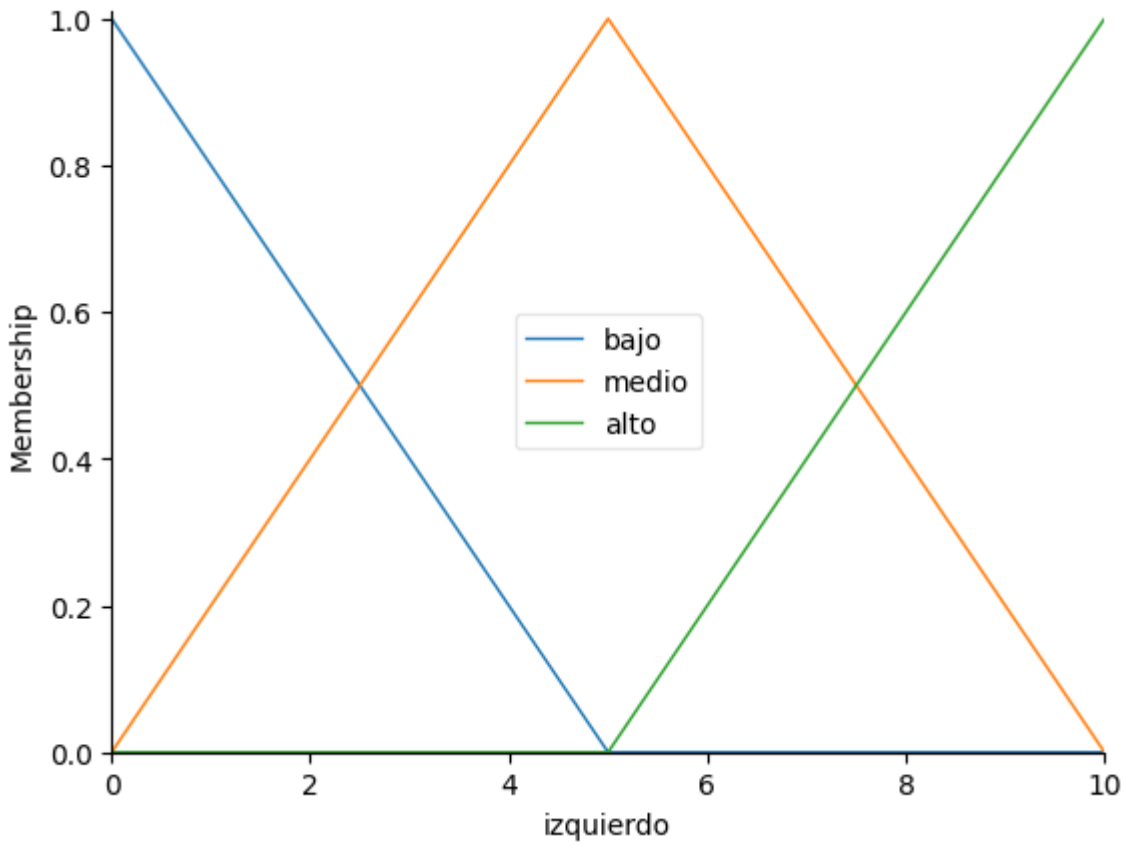
1. **Instalación de Dependencias**: Instalación de la biblioteca `scikit-fuzzy` necesaria para implementar la lógica difusa.
2. **Definición de Variables**: Definición de las variables de entrada (sensores) y salida (ángulo de giro) del sistema.
3. **Funciones de Pertenencia**: Creación de funciones de pertenencia para las variables utilizando diferentes tipos de funciones.
4. **Reglas Difusas**: Establecimiento de reglas difusas que describen el comportamiento del sistema.
5. **Simulación y Visualización**: Simulación del sistema de control difuso y visualización de los resultados.

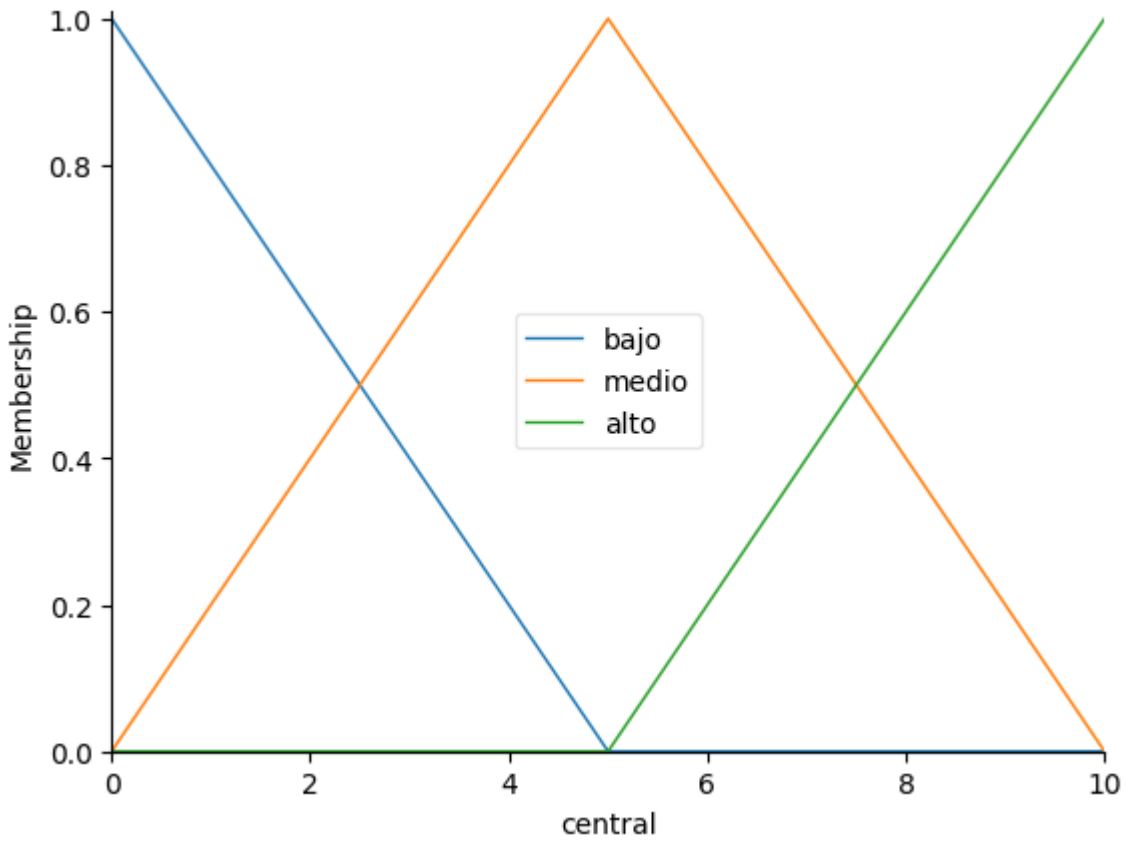
In [11]:
```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
import matplotlib.pyplot as plt

# Definir los rangos de las variables
sensor_range = np.arange(0, 11, 1)  # Rango de 0 a 10 para sensores
giro_range = np.arange(-100, 101, 1)  # Ángulo de giro de -100 (izquierda) a 100 (derecha)
```

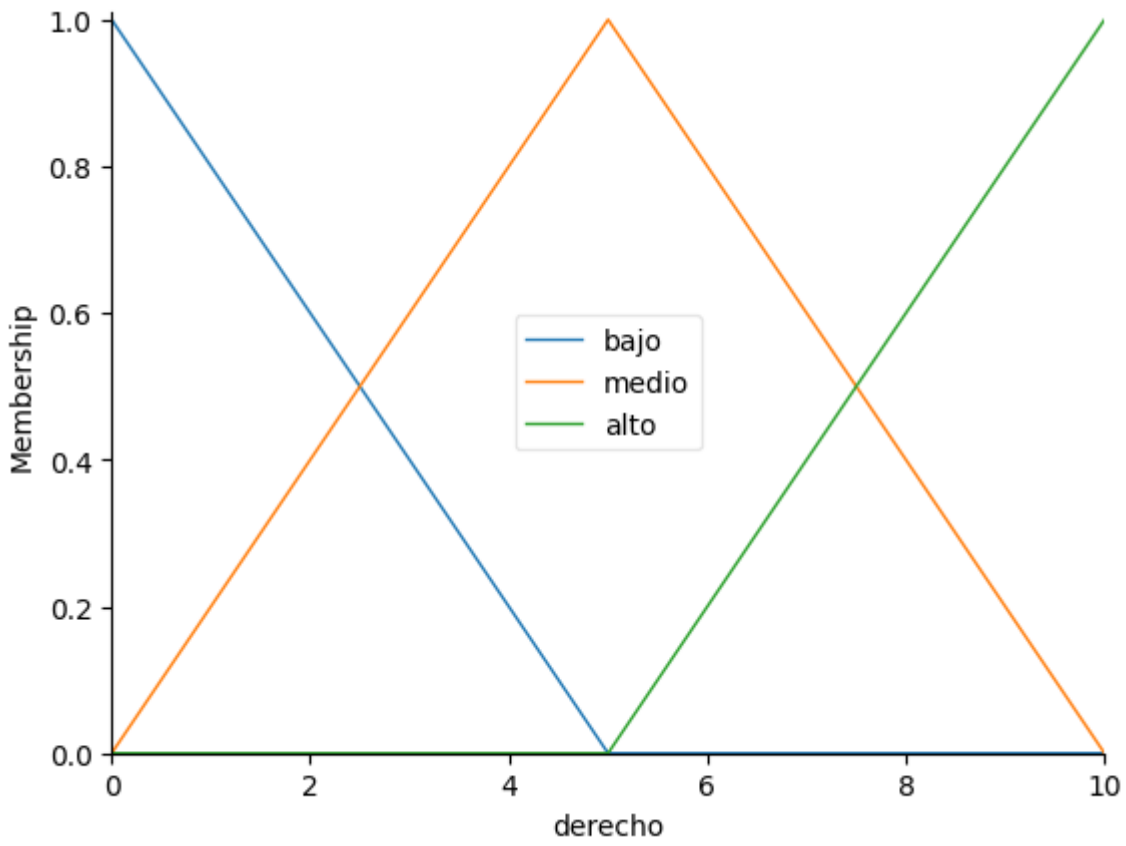
```
# Variables de entrada
izquierdo = ctrl.Antecedent(sensor_range, 'izquierdo')
central = ctrl.Antecedent(sensor_range, 'central')
derecho = ctrl.Antecedent(sensor_range, 'derecho')

# Variable de salida
giro = ctrl.Consequent(giro_range, 'giro')

# Funciones de pertenencia para los sensores
izquierdo['bajo'] = fuzz.trimf(sensor_range, [0, 0, 5])
izquierdo['medio'] = fuzz.trimf(sensor_range, [0, 5, 10])
izquierdo['alto'] = fuzz.trimf(sensor_range, [5, 10, 10])

central['bajo'] = fuzz.trimf(sensor_range, [0, 0, 5])
central['medio'] = fuzz.trimf(sensor_range, [0, 5, 10])
central['alto'] = fuzz.trimf(sensor_range, [5, 10, 10])

derecho['bajo'] = fuzz.trimf(sensor_range, [0, 0, 5])
derecho['medio'] = fuzz.trimf(sensor_range, [0, 5, 10])
derecho['alto'] = fuzz.trimf(sensor_range, [5, 10, 10])

# Funciones de pertenencia para el giro
giro['izquierda'] = fuzz.trimf(giro_range, [-100, -50, 0])
giro['recto'] = fuzz.trimf(giro_range, [-10, 0, 10])
giro['derecha'] = fuzz.trimf(giro_range, [0, 50, 100])
```

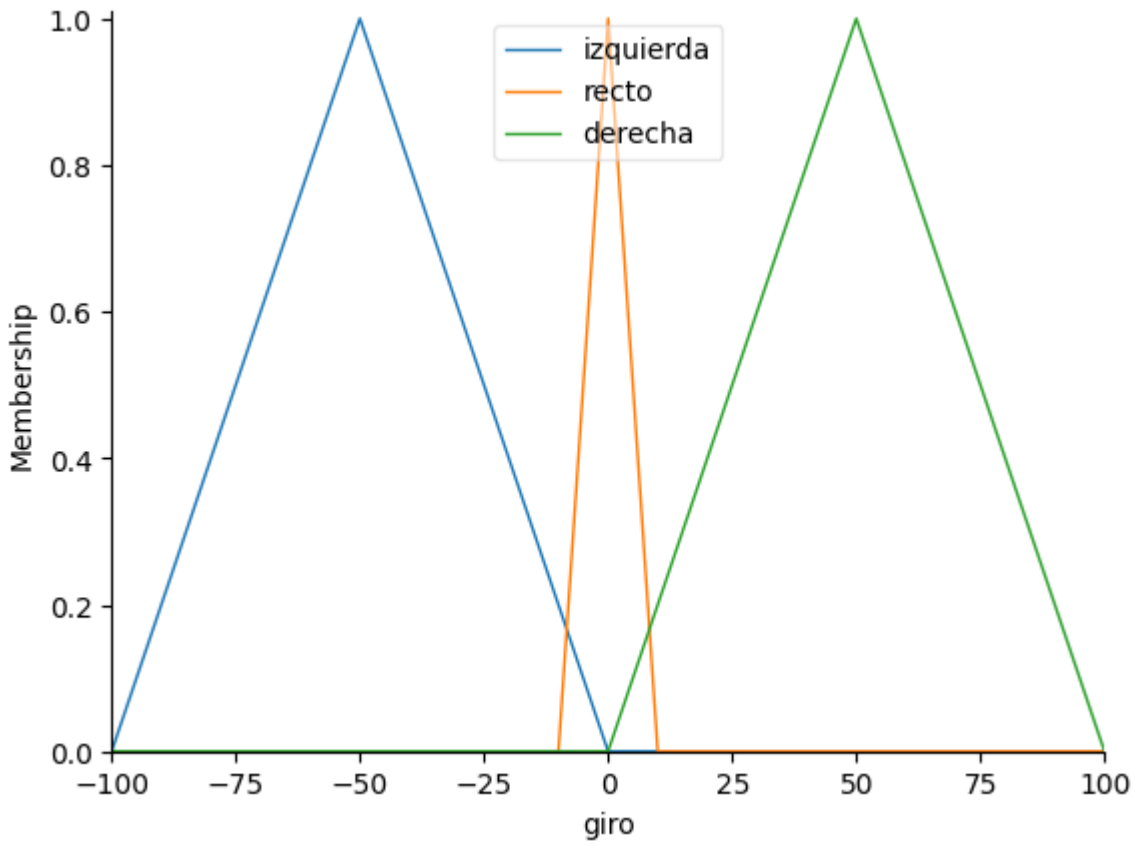
In [12]:
```
rule1 = ctrl.Rule(izquierdo['alto'] & central['bajo'] & derecho['bajo'], giro['izquierda'])
rule2 = ctrl.Rule(izquierdo['bajo'] & central['alto'] & derecho['bajo'], giro['recto'])
rule3 = ctrl.Rule(izquierdo['bajo'] & central['bajo'] & derecho['alto'], giro['derecha'])
```

In [13]:
```
izquierdo.view()
central.view()
derecho.view()
giro.view()
```




1.0
0.8
0.6
0.4
0.2
0.0
Membership
bajo
medio
alto
0
2
4
6
8
10
central


1.0
0.8
0.6
0.4
0.2
0.0
bajo
medio
alto
0
2
4
6
8
10
derecho

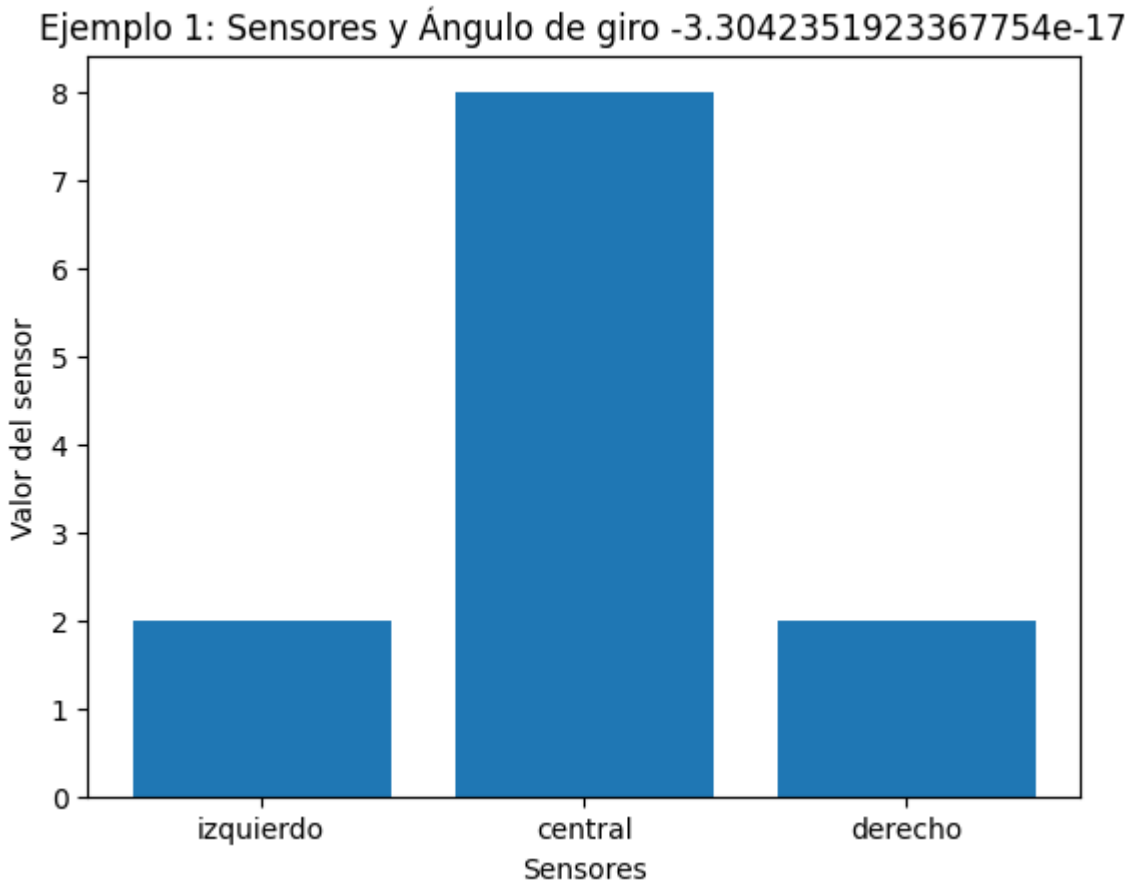



In [14]:
```python
giro_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])
giro_simulador = ctrl.ControlSystemSimulation(giro_ctrl)

# Lista de ejemplos de entrada
ejemplos_sensores = [
    {'izquierdo': 2, 'central': 8, 'derecho': 2},  # Ejemplo 1
    {'izquierdo': 8, 'central': 3, 'derecho': 1},  # Ejemplo 2
    {'izquierdo': 1, 'central': 3, 'derecho': 8},  # Ejemplo 3
    {'izquierdo': 7, 'central': 1, 'derecho': 1},  # Ejemplo 4
    {'izquierdo': 1, 'central': 1, 'derecho': 9},  # Ejemplo 5
    {'izquierdo': 6, 'central': 3, 'derecho': 1},  # Ejemplo 6
    {'izquierdo': 9, 'central': 9, 'derecho': 9},  # Ejemplo 7
]
```

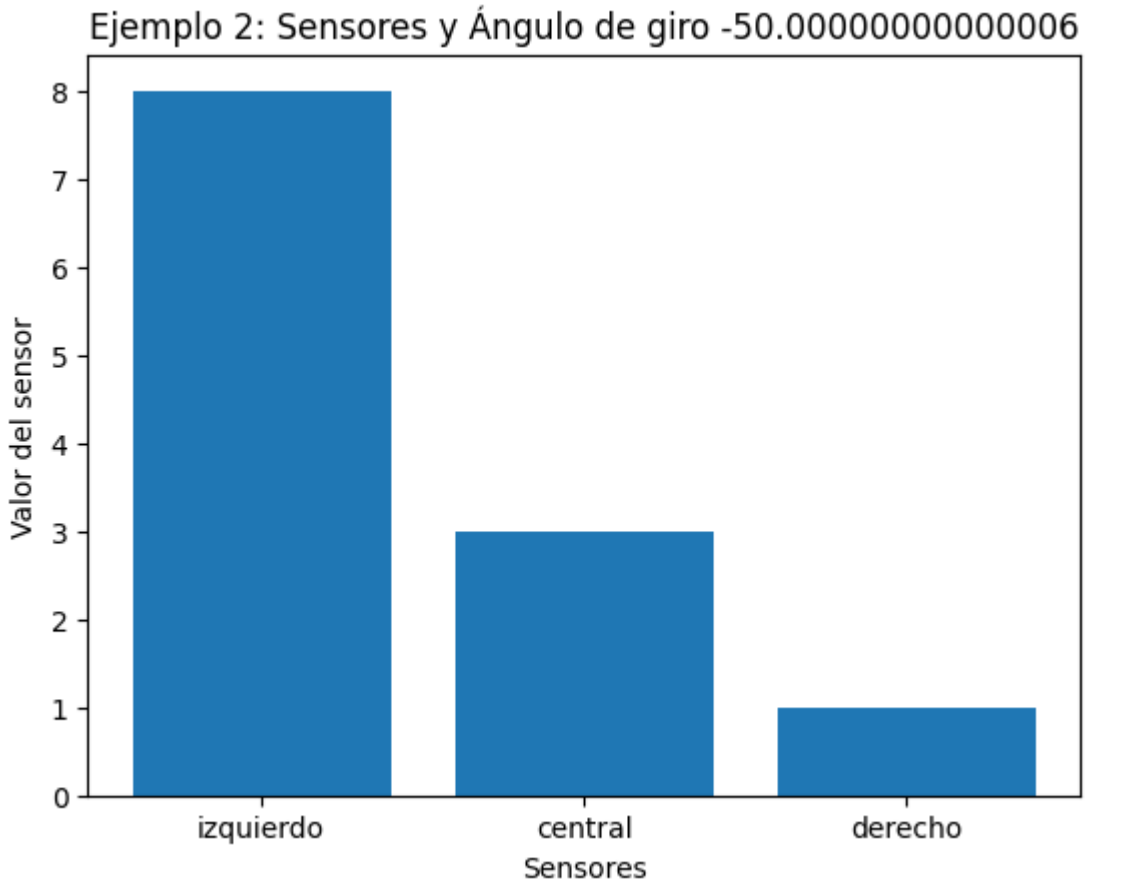
In [15]:
```python
import time

# Función para simular y graficar el resultado
def graficar_resultados(ejemplos):
    for i, ejemplo in enumerate(ejemplos):
        giro_simulador.input['izquierdo'] = ejemplo['izquierdo']
        giro_simulador.input['central'] = ejemplo['central']
        giro_simulador.input['derecho'] = ejemplo['derecho']

        # Calcular el resultado
        giro_simulador.compute()

        # Obtener el resultado del giro
        print(giro_simulador.output)
        time.sleep(1)
        try:
            angulo_giro = giro_simulador.output['giro']

        except:
            angulo_giro = 0
        # Graficar los valores de los sensores como una gráfica de barras
        _, ax = plt.subplots()
        sensores = ['izquierdo', 'central', 'derecho']
        valores = [ejemplo['izquierdo'], ejemplo['central'], ejemplo['derecho']]
        ax.bar(sensores, valores)
        ax.set_title(f"Ejemplo {i+1}: Sensores y Ángulo de giro {angulo_giro}")
        ax.set_ylabel("Valor del sensor")
        ax.set_xlabel("Sensores")
        plt.show()
```

```
# Ejecutar la simulación y graficar los puntos
graficar_resultados(ejemplos_sensores)

plt.tight_layout()
plt.show()
```

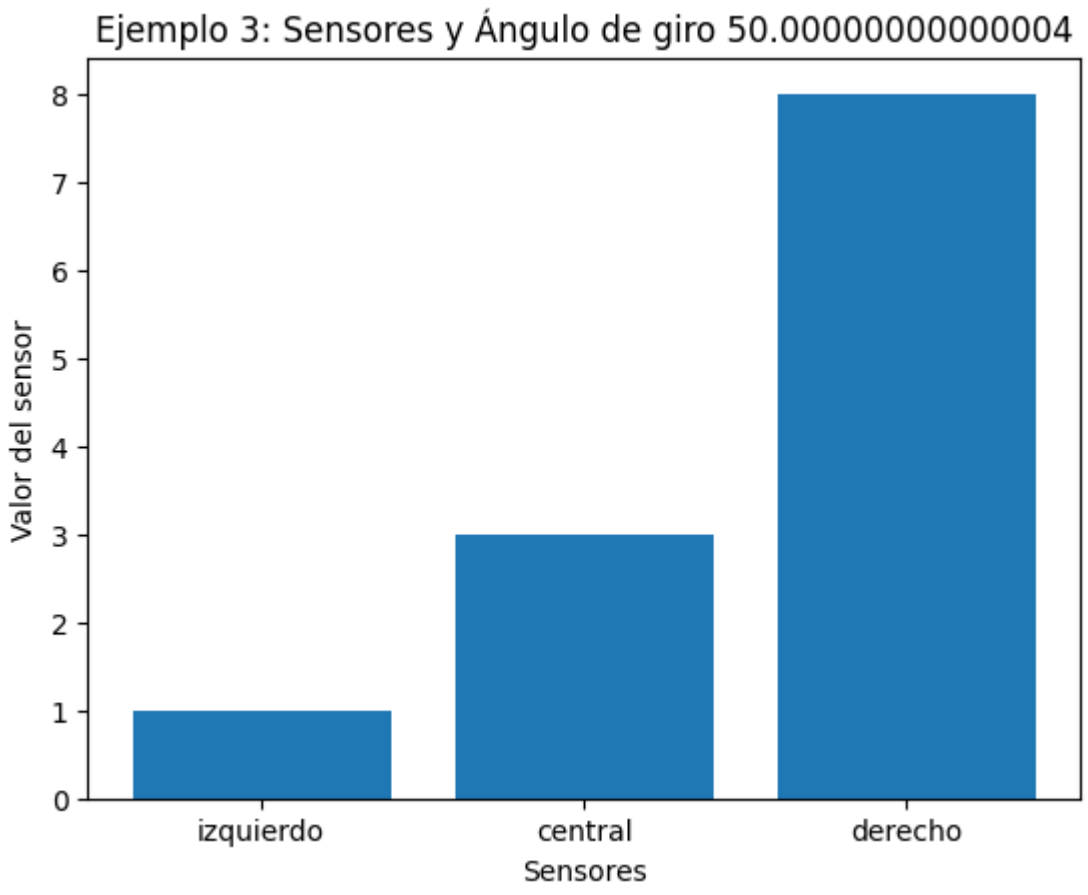
```
{'giro': -3.3042351923367754e-17}
```



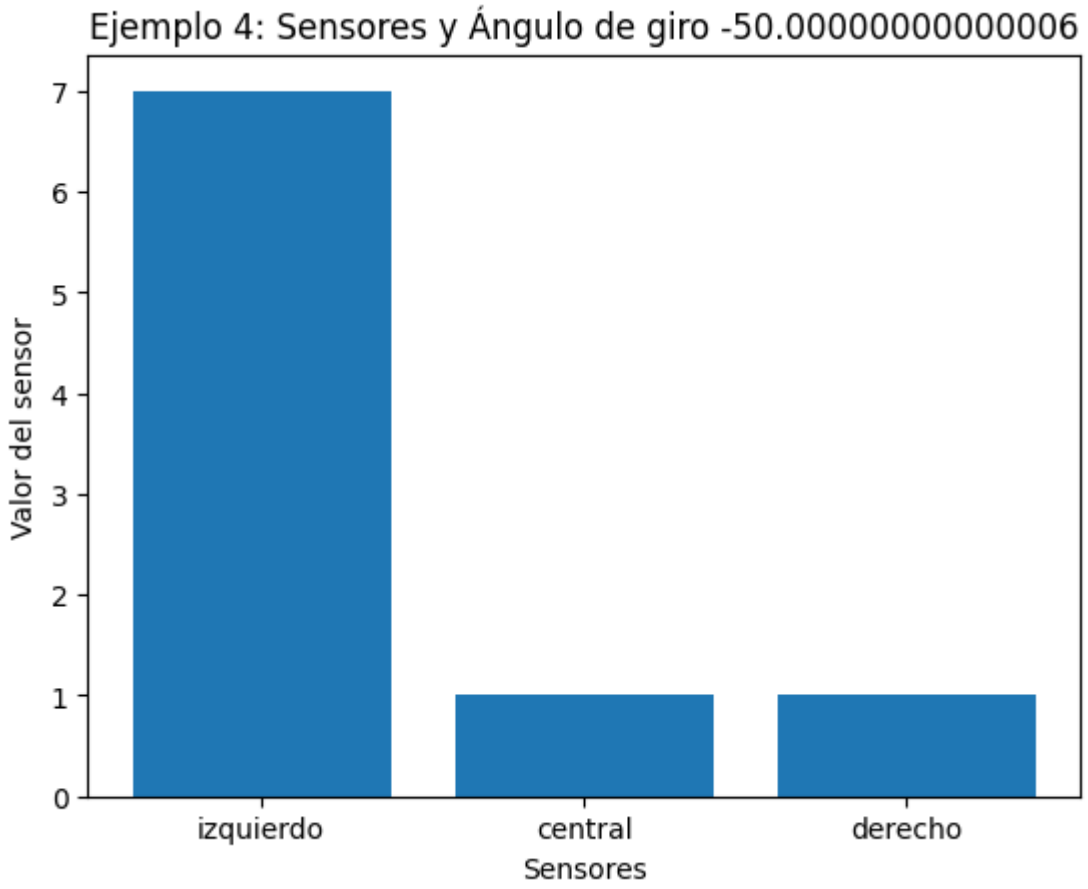

```
{'giro': -50.00000000000006}
```



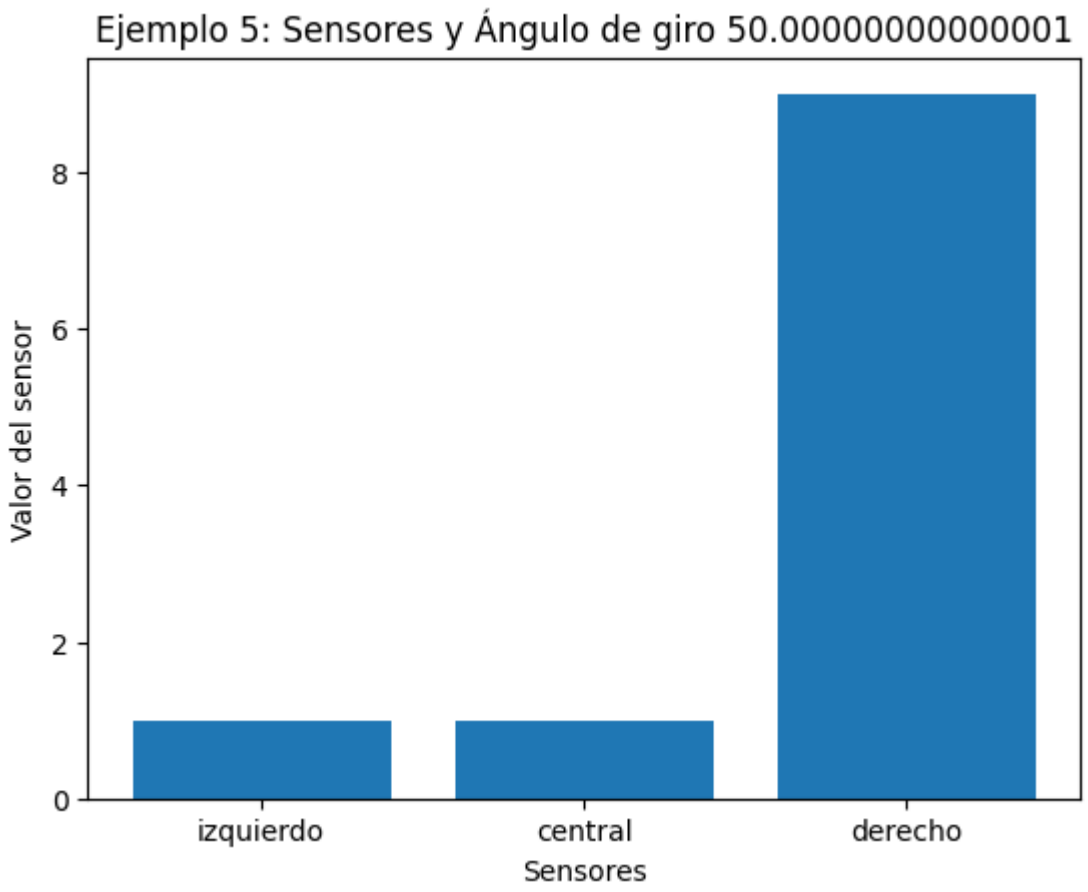

```
{'giro': 50.00000000000004}
```



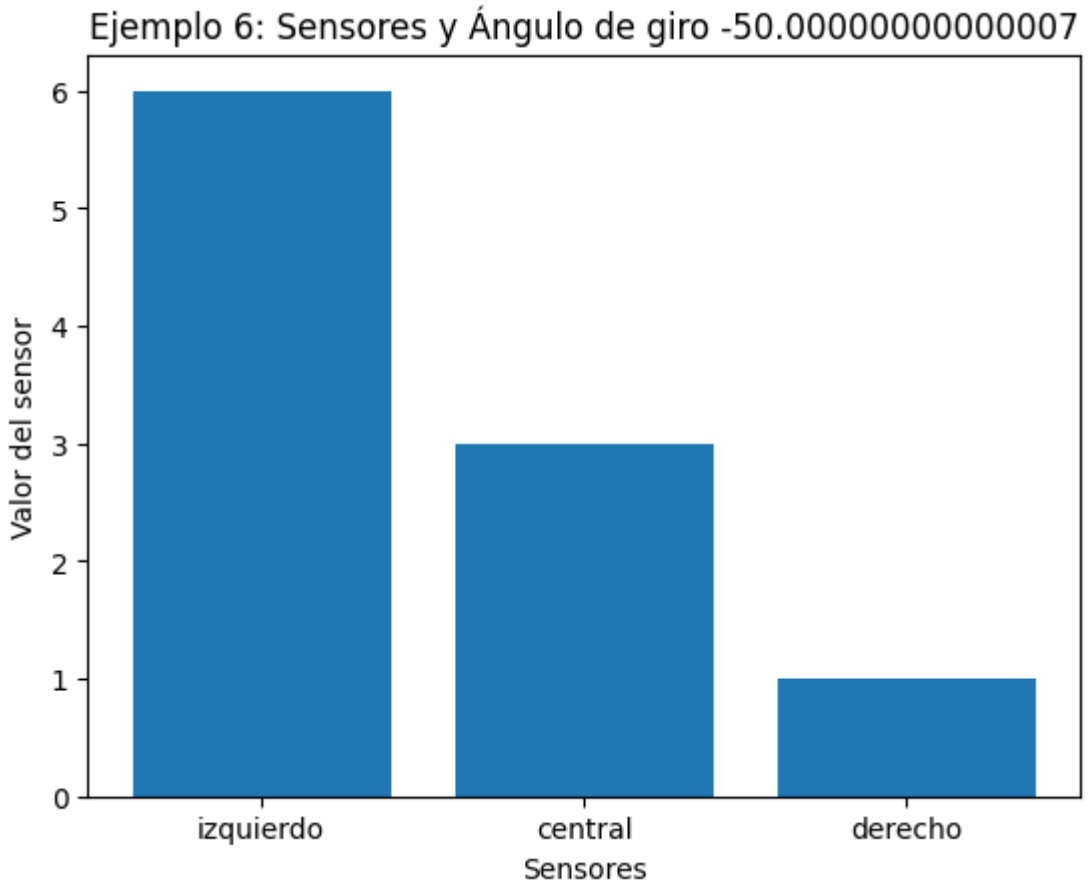

```
{'giro': -50.00000000000006}
```



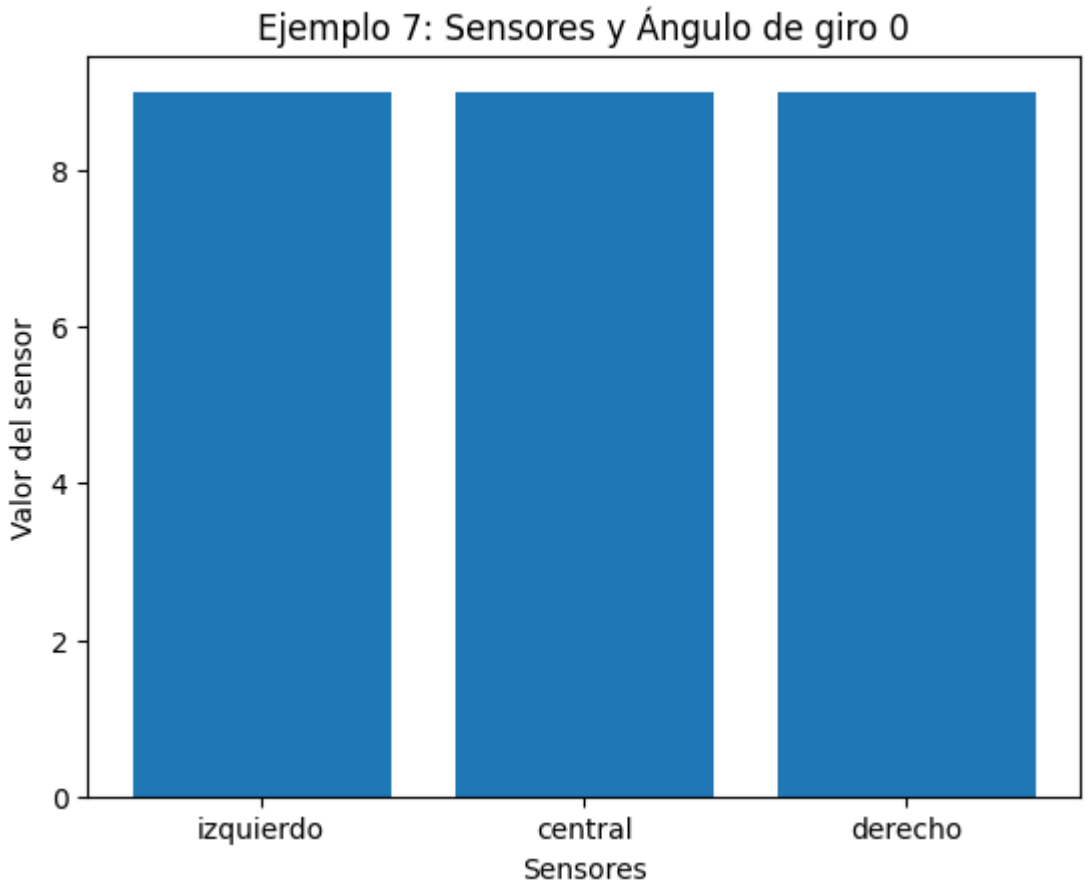

```
{'giro': 50.00000000000001}
```




```
{'giro': -50.00000000000007}
```




```
{}
```

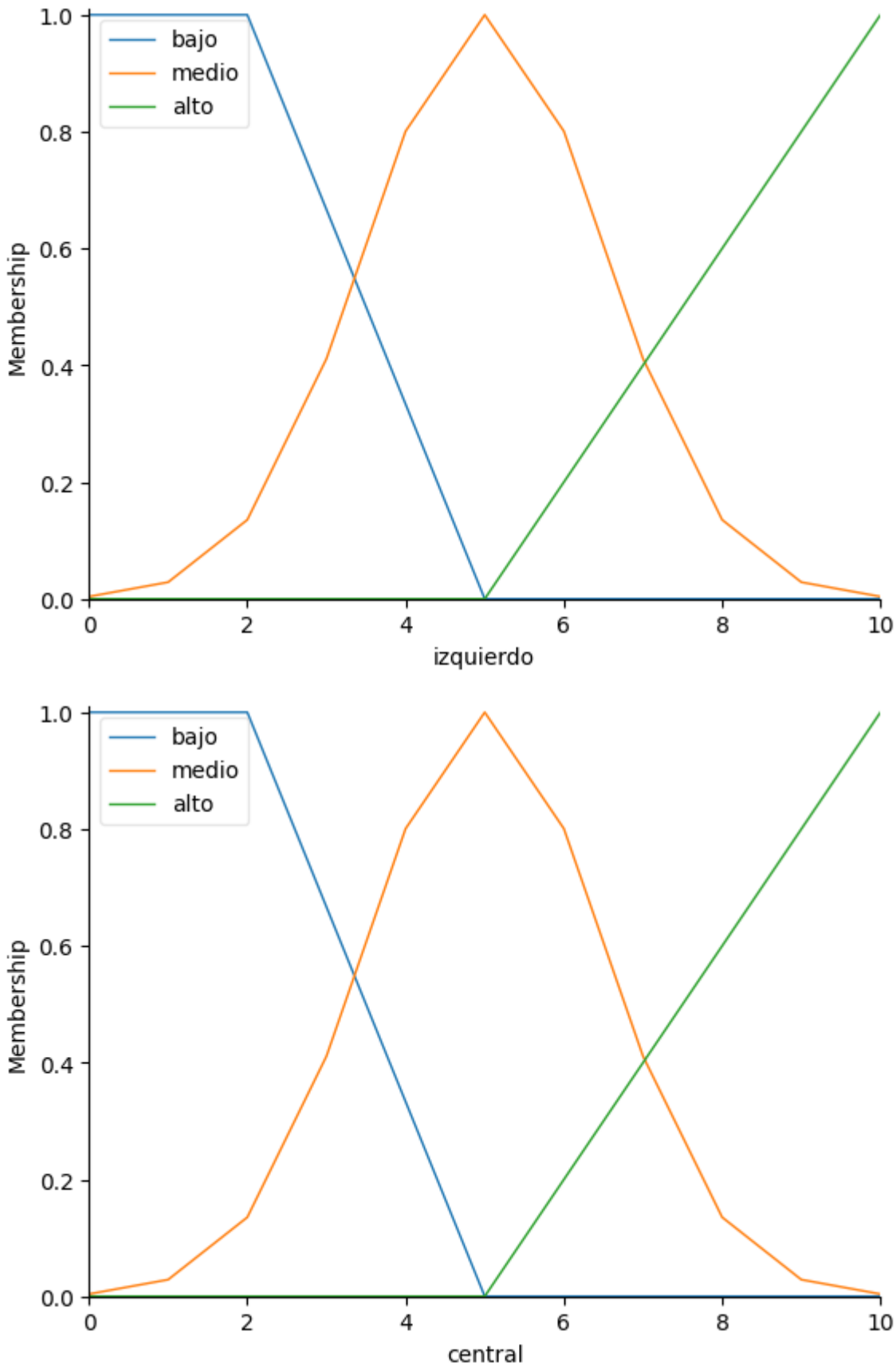


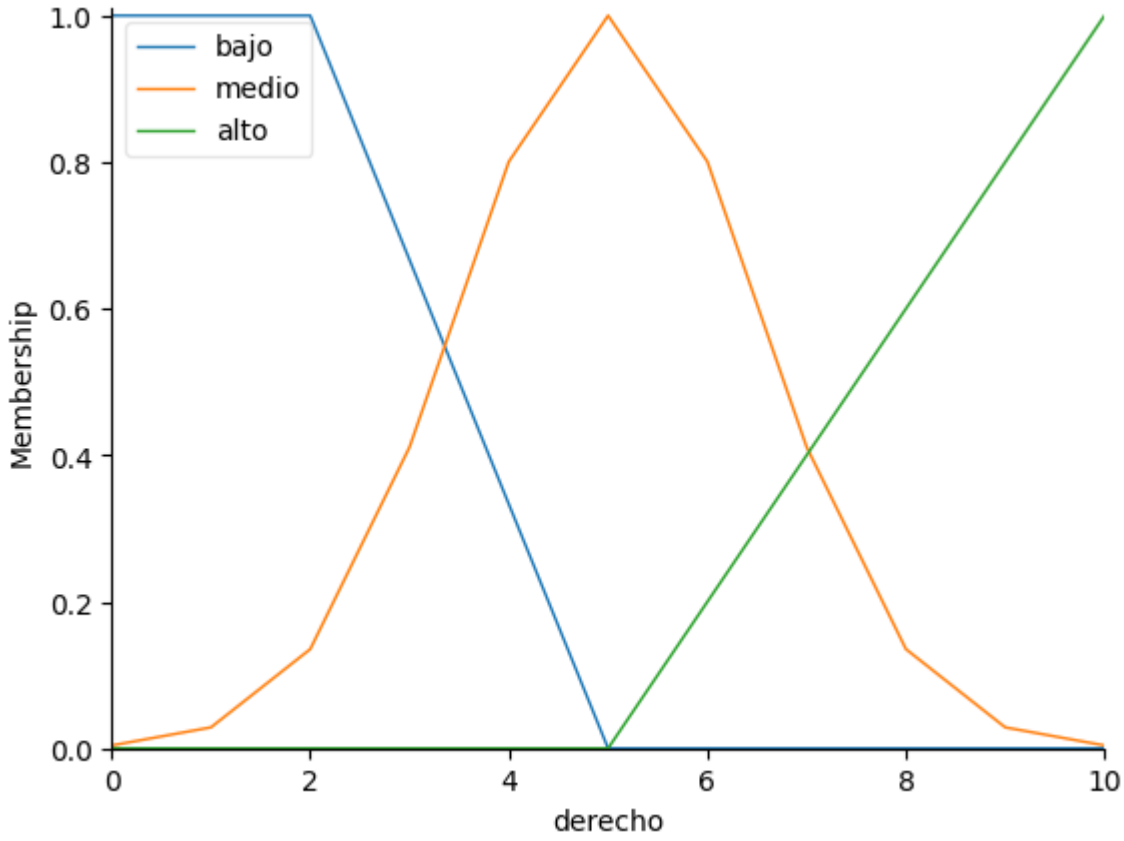

```
<Figure size 640x480 with 0 Axes>
```

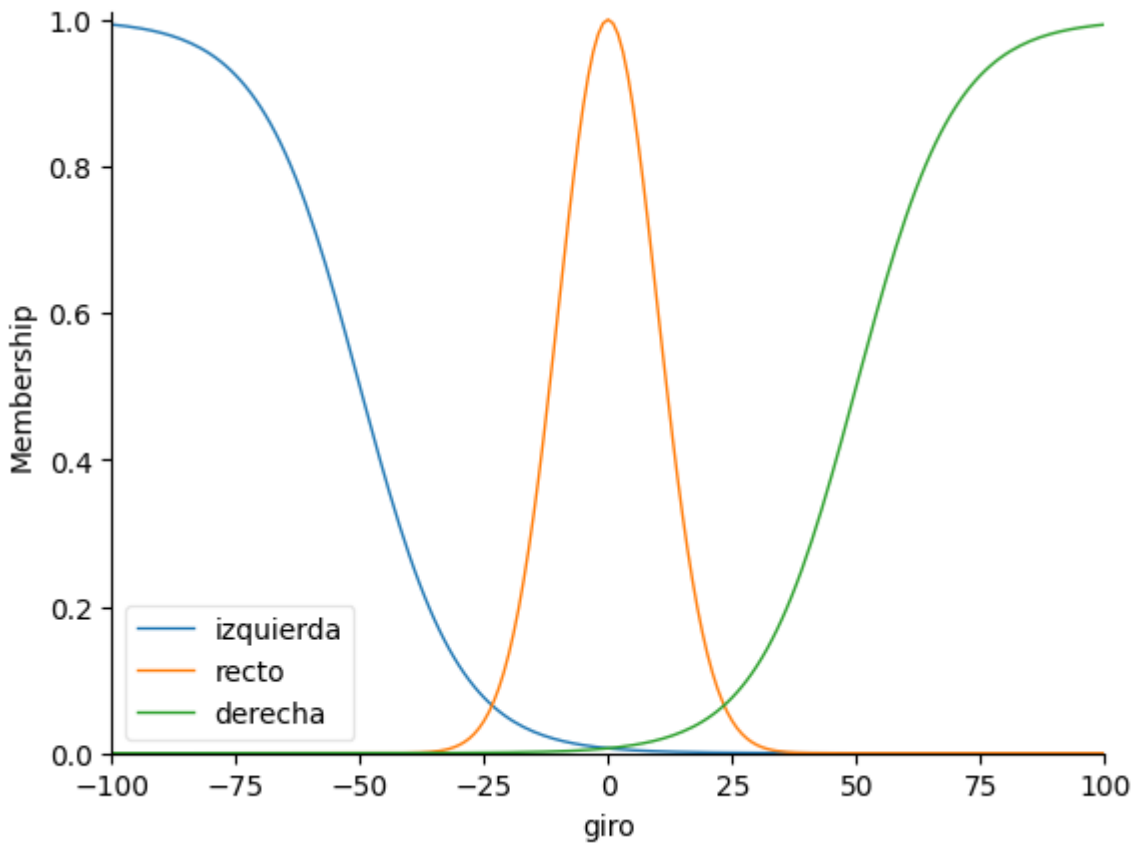
In [16]:
```python
# Definir los rangos de las variables
sensor_range = np.arange(0, 11, 1)  # Rango de 0 a 10 para sensores
giro_range = np.arange(-100, 101, 1)  # Ángulo de giro de -100 (izquierda) a 100 (derecha)

# Variables de entrada (sensores)
izquierdo = ctrl.Antecedent(sensor_range, 'izquierdo')
central = ctrl.Antecedent(sensor_range, 'central')
derecho = ctrl.Antecedent(sensor_range, 'derecho')

# Variable de salida (giro)
giro = ctrl.Consequent(giro_range, 'giro')

# Funciones de pertenencia para los sensores usando diferentes tipos de funciones
izquierdo['bajo'] = fuzz.trapmf(sensor_range, [0, 0, 2, 5])  # Trapezoidal
izquierdo['medio'] = fuzz.gaussmf(sensor_range, 5, 1.5)      # Gaussiana
izquierdo['alto'] = fuzz.trimf(sensor_range, [5, 10, 10])    # Triangular

central['bajo'] = fuzz.trapmf(sensor_range, [0, 0, 2, 5])     # Trapezoidal
central['medio'] = fuzz.gaussmf(sensor_range, 5, 1.5)         # Gaussiana
central['alto'] = fuzz.trimf(sensor_range, [5, 10, 10])       # Triangular

derecho['bajo'] = fuzz.trapmf(sensor_range, [0, 0, 2, 5])     # Trapezoidal
derecho['medio'] = fuzz.gaussmf(sensor_range, 5, 1.5)         # Gaussiana
derecho['alto'] = fuzz.trimf(sensor_range, [5, 10, 10])       # Triangular

# Funciones de membresía para la salida (giro)
giro['izquierda'] = fuzz.sigmf(giro_range, -50, -0.1)        # Sigmoidal para giro a la izqu
giro['recto'] = fuzz.gaussmf(giro_range, 0, 10)              # Gaussiana para ir recto
giro['derecha'] = fuzz.sigmf(giro_range, 50, 0.1)            # Sigmoidal para giro a la dere

# Definir las reglas difusas
rule1 = ctrl.Rule(izquierdo['alto'] & central['bajo'] & derecho['bajo'], giro['izquierda'])
rule2 = ctrl.Rule(izquierdo['bajo'] & central['alto'] & derecho['bajo'], giro['recto'])
rule3 = ctrl.Rule(izquierdo['bajo'] & central['bajo'] & derecho['alto'], giro['derecha'])

# Crear el sistema de control difuso y simular
giro_control = ctrl.ControlSystem([rule1, rule2, rule3])
giro_simulacion = ctrl.ControlSystemSimulation(giro_control)

# Ejemplo de entrada (modifica estos valores para simular diferentes lecturas de sensor)
giro_simulacion.input['izquierdo'] = 2  # Valor del sensor izquierdo
giro_simulacion.input['central'] = 7    # Valor del sensor central
giro_simulacion.input['derecho'] = 2    # Valor del sensor derecho
```

```
# Realizar la simulación
giro_simulacion.compute()

# Mostrar el resultado del giro
print("Giro:", giro_simulacion.output['giro'])

# Visualización opcional de los resultados de las funciones de membresía
izquierdo.view()
central.view()
derecho.view()
giro.view()
```

```
Giro: -8.502022402451779e-16
```



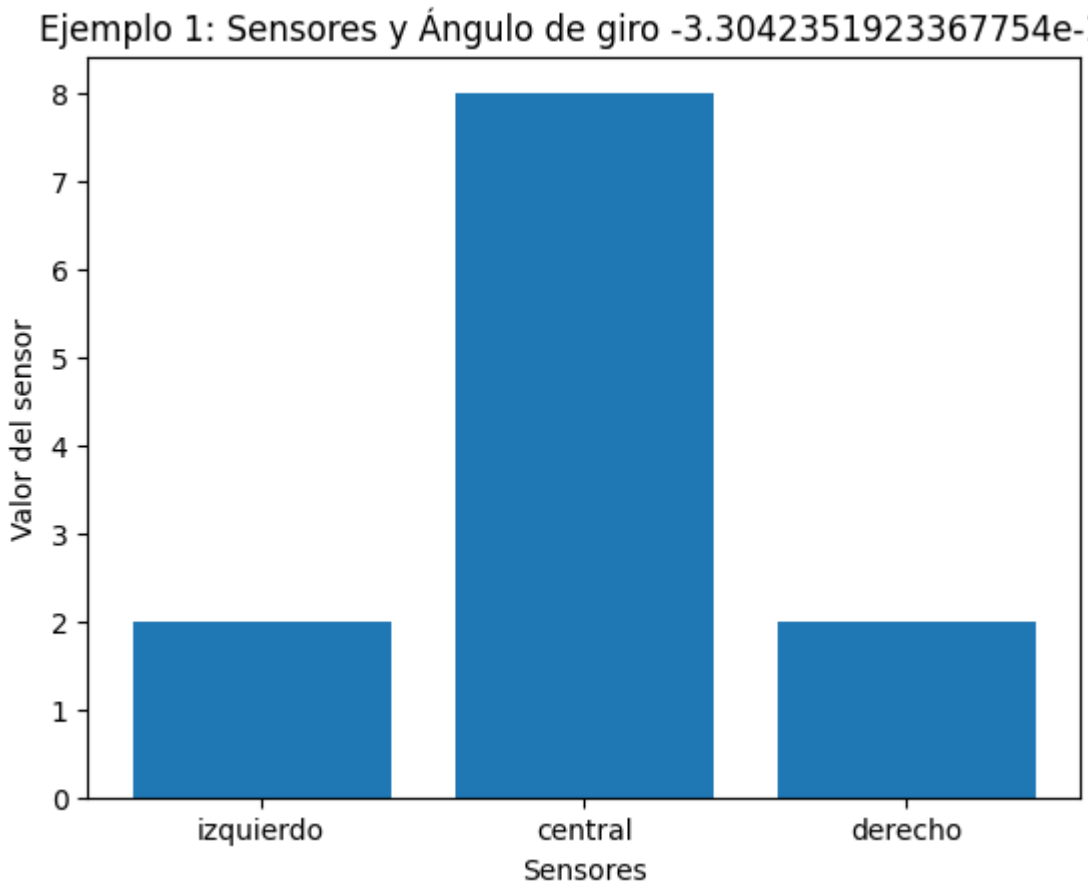


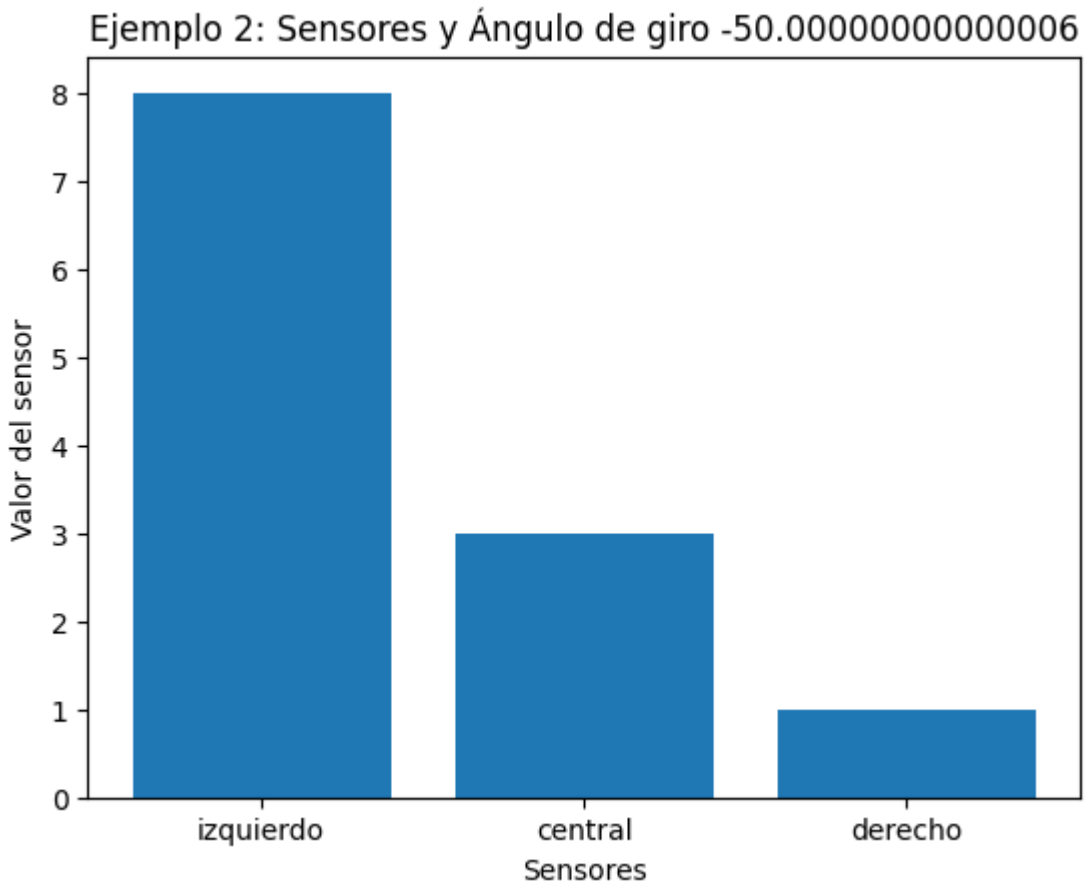

In [17]:
```python
import time

# Función para simular y graficar el resultado
def graficar_resultados(ejemplos):
    for i, ejemplo in enumerate(ejemplos):
        giro_simulador.input['izquierdo'] = ejemplo['izquierdo']
        giro_simulador.input['central'] = ejemplo['central']
        giro_simulador.input['derecho'] = ejemplo['derecho']

        # Calcular el resultado
        giro_simulador.compute()

        # Obtener el resultado del giro
        print(giro_simulador.output)
        time.sleep(1)
        try:
            angulo_giro = giro_simulador.output['giro']

        except:
            angulo_giro = 0
```

```
        # Graficar los valores de los sensores como una gráfica de barras
        _, ax = plt.subplots()
        sensores = ['izquierdo', 'central', 'derecho']
        valores = [ejemplo['izquierdo'], ejemplo['central'], ejemplo['derecho']]
        ax.bar(sensores, valores)
        ax.set_title(f"Ejemplo {i+1}: Sensores y Ángulo de giro {angulo_giro}")
        ax.set_ylabel("Valor del sensor")
        ax.set_xlabel("Sensores")
        plt.show()



# Ejecutar la simulación y graficar los puntos
graficar_resultados(ejemplos_sensores)

plt.tight_layout()
plt.show()
```

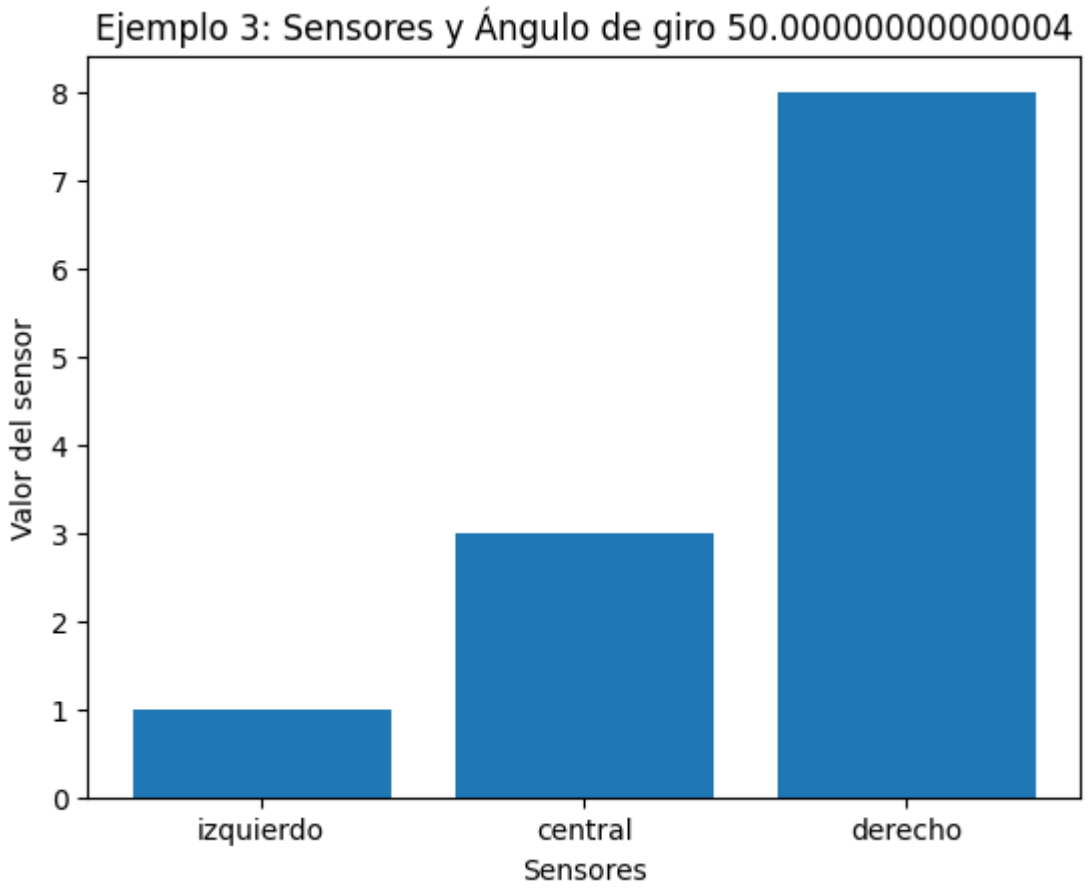
```
{'giro': -3.3042351923367754e-17}
```



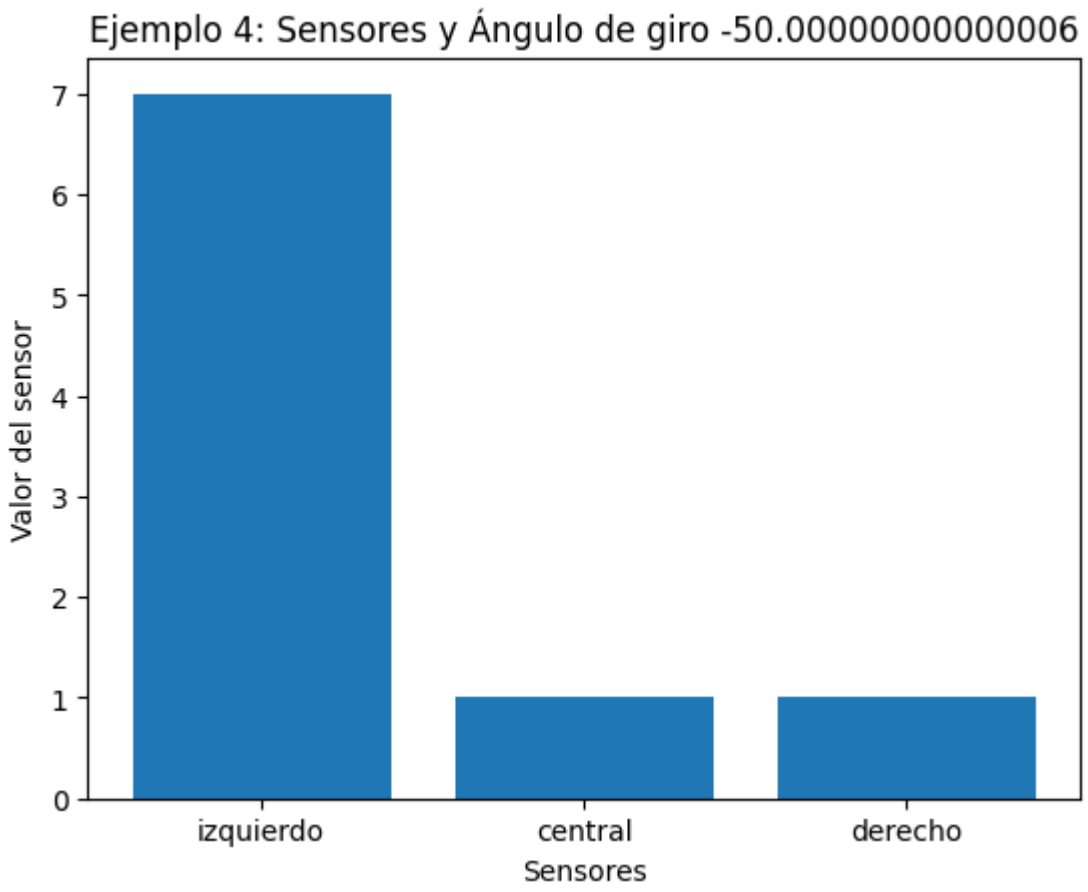

```
{'giro': -50.00000000000006}
```



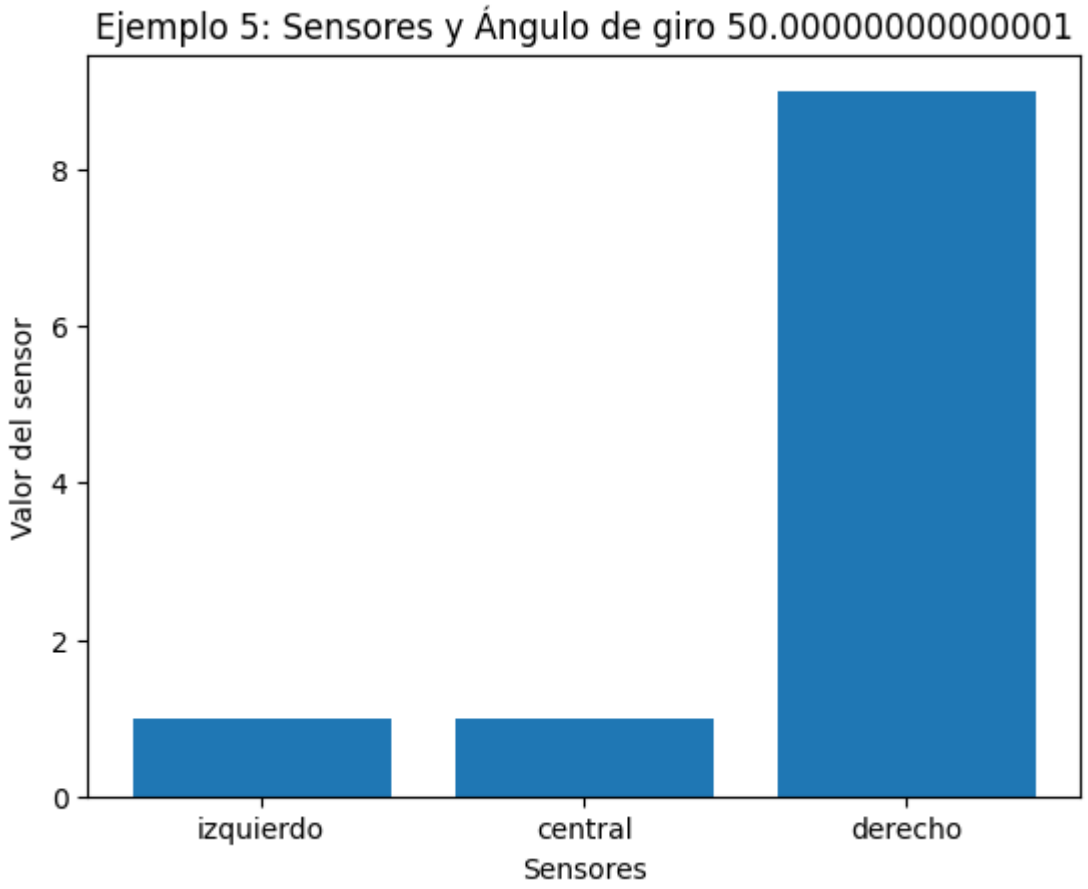

```
{'giro': 50.00000000000004}
```



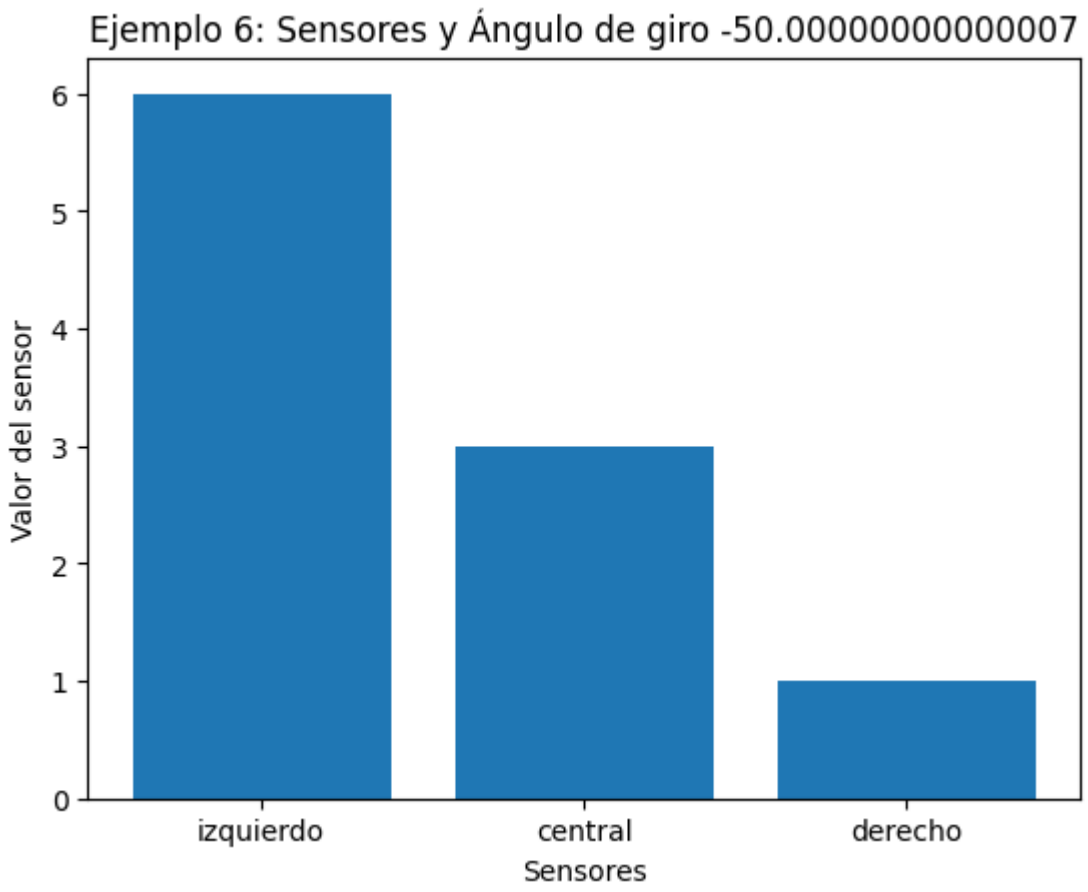

```
{'giro': -50.00000000000006}
```



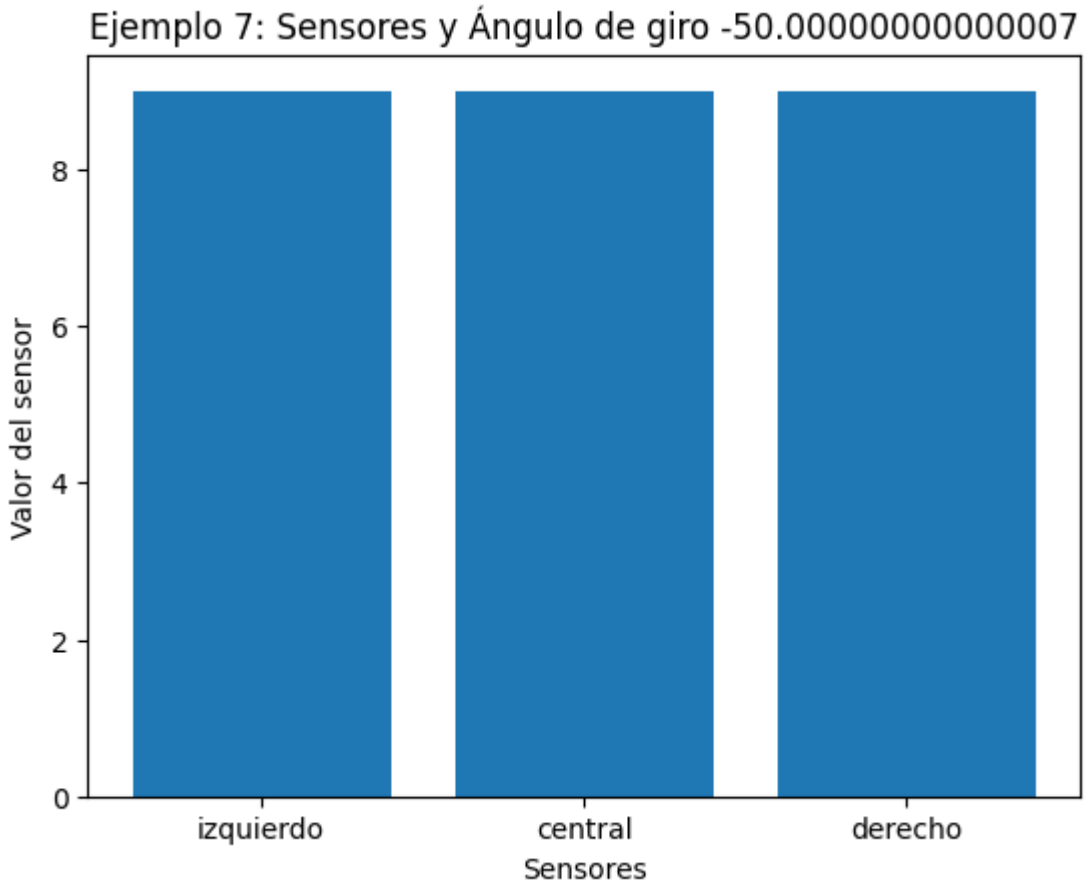

```
{'giro': 50.00000000000001}
```




```
{'giro': -50.00000000000007}
```




```
{'giro': -50.00000000000007}
```




```
<Figure size 640x480 with 0 Axes>
```