PRÁCTICA 1.2: EJERCICIO "VELETA Y GENERADOR"

Instituto Politécnico Nacional
"La Técnica al Servicio de la Patria"

Ingeniería Telemática

Lógica Difusa

Abarca Romero José Ángel

Profesora

Gonzalez Navarro Yesenia Eleon

2TM9

Índice

1. Objetivo

1. Implemente en Python un sistema de inferencia de tipo Mamdani para el problema del archivo proporcionado.

   a) Un código que genere las gráficas de los universos de entrada y salida caracterizados, los conjuntos de salida recortados, la curva resultante de unir los conjuntos de salida recortados y los valores desdifusificados de las variables de salida Z*.

   b) Un código que genere las superficies de control para las variables de salida.

2. Marco Teórico

2.1. Sistemas de Inferencia de Tipo Mamdani

Estos sistemas de inferencia utilizan universos difusos tanto en su entrada como en su salida. Existen diversos métodos de desdifusificación entre los cuales se encuentran: izquierda del máximo, media del máximo, derecha del máximo, bicéctriz, promedio de pesos y centroide del área. El método del centroide del área resulta ser el de menor complejidad de todos y aún así se obtienen resultados lo suficientemente aproximados a otros métodos para considerarlo correcto. Su ecuación se muestra a continuación

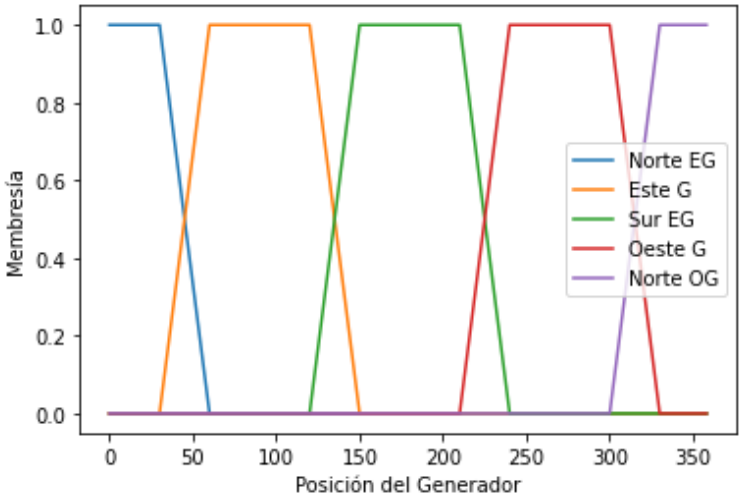
$$z^* = \frac{\sum \mu_{CT}(z) \cdot z}{\sum \mu_{CT}(z)} \tag{1}$$

donde z^* es el valor certero desdifusificado, que resulta de evaluar cada elemento z del universo de salida ponderado por su pertenencia en el conjunto de salida CT.

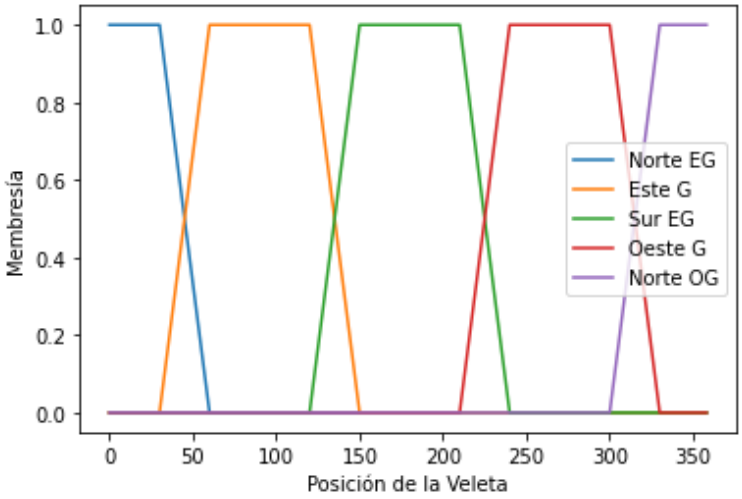
3. Desarrollo

3.1. Caracterización de los universos de entrada

En el archivo proporcionado por la profesora, se plantean los nombres para cada uno de los conjuntos pertenecientes a los universos difusos de entrada. Sin embargo, estos carecen de rangos, por lo tanto, me di a la tarea primero de asignarles rangos a cada uno de los conjuntos. Es necesario destacar que si bien ambas variables de entrada utilizan los mismos conjuntos, estos no comparten dependencia entre sí.



(a) Universo de Entrada 1: Posición del generador en grados (0-359°).

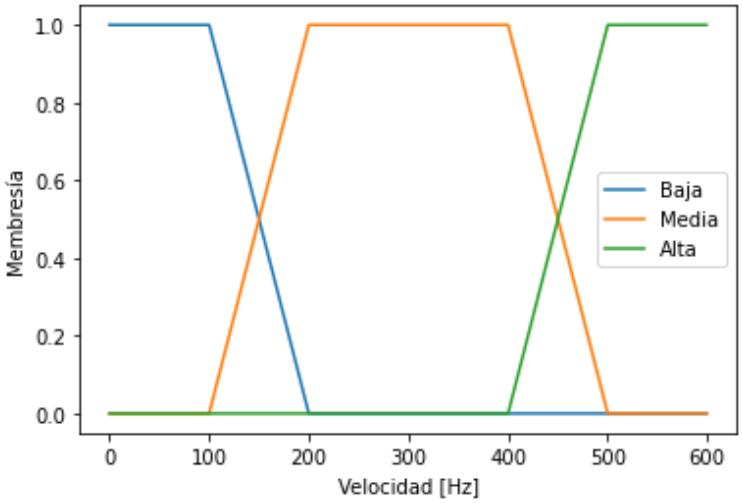


(b) Universo de Entrada 2: Posición de la veleta en grados (0-359°).

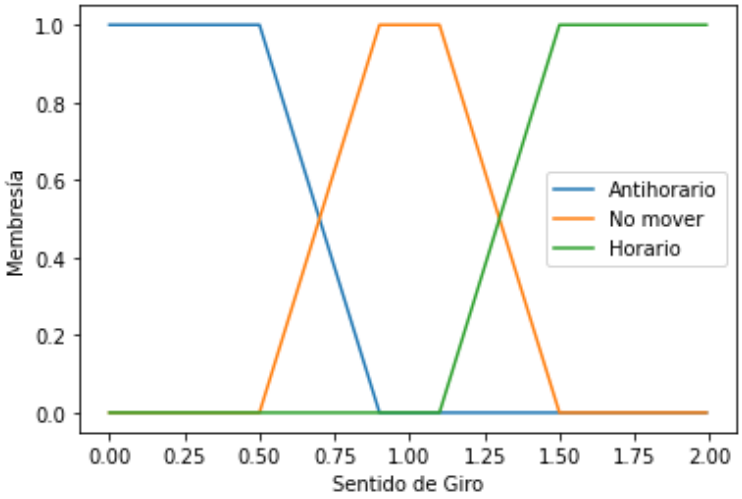
Figura 1: Variables de entrada al sistema.

3.2. Caracterización de los universos de salida

Los universos de salida están definidos en rangos totalmente distintos a los universos de entrada. Comparten entre ellos que cada uno posee 3 conjuntos definidos dentro de sí. De igual forma que para los universos de entrada, los rangos de cada uno de estos conjuntos los definí yo.



(a) Universo de Salida 1: Velocidad de giro del generador (Hz).



(b) Universo de Salida 2: Sentido de giro del generador.

Figura 2: Variables de salida del sistema

3.3. Tablas de inferencia hacia los conjuntos de salida

Las tablas de inferencia son las que determinan las reglas "si - entonces". A continuación presento las tablas de inferencia proporcionadas dentro del archivo de instrucciones de la práctica.

Posición Veleta / Posición Generador	NorteE_G	Este_G	Sur_G	Oeste_G	NorteO_G
NorteE_V	Baja	Media	Alta	Media	Baja
Este_V	Media	Baja	Media	Alta	Media
Sur_V	Alta	Media	Baja	Media	Alta
Oeste_V	Media	Alta	Media	Baja	Media
NorteO_V	Baja	Media	Alta	Media	Baja

Cuadro 1: Tabla de inferencia hacia Velocidad de giro.

Posición Veleta / Posición Generador	NorteE_G	Este_G	Sur_G	Oeste_G	NorteO_G
NorteE_V	No mover	Anti horario	Anti horario	Horario	No mover
Este_V	Horario	No mover	Anti horario	Horario	Horario
Sur_V	Horario	Horario	No mover	Anti horario	Anti horario
Oeste_V	Anti horario	Anti horario	Horario	No mover	Anti horario
NorteO_V	No mover	Anti horario	Horario	Horario	No mover

Cuadro 2: Tabla de inferencia hacia Sentido de giro.

En base a las tablas de inferencia anteriormente mostradas, se tienen 50 reglas "si - entonces" en total (25 por cada tabla).

3.4. Programación del sistema de inferencia en Python

Con base a las reglas "si - entonces" generadas para las tablas de inferencia 1 y 2, diseñé un programa que me permite difusificar dos números certeros y desdifusificar dos variables de salida para obtener valores numéricos certeros a partir del uso de un sistema de inferencia de tipo Mamdani. A continuación muestro las gráficas correspondientes al proceso de desdifusificación de un valor difuso para cada uno de los universos de salida a partir de dos valores de entrada $x = 135$ y $y = 315$.

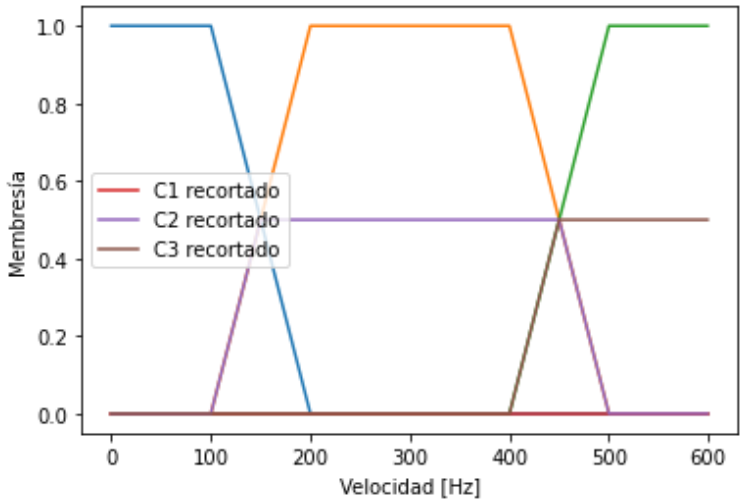


Figura 3: Cortes en los conjuntos del universo de salida "Velocidad de giro".

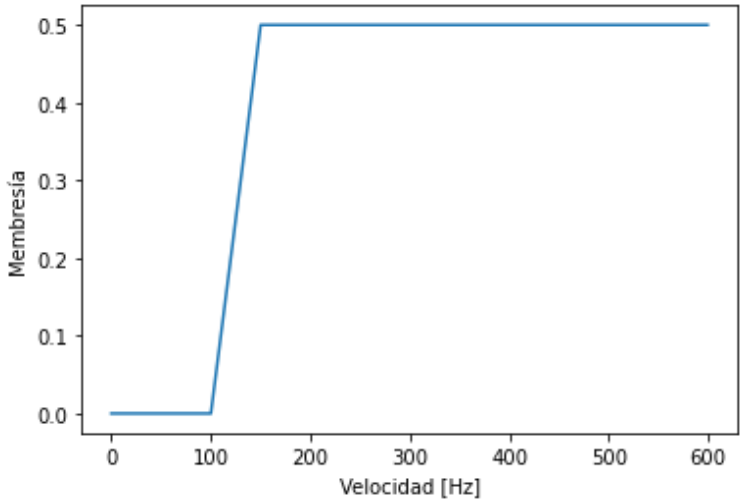


Figura 4: Unión de los conjuntos recortados

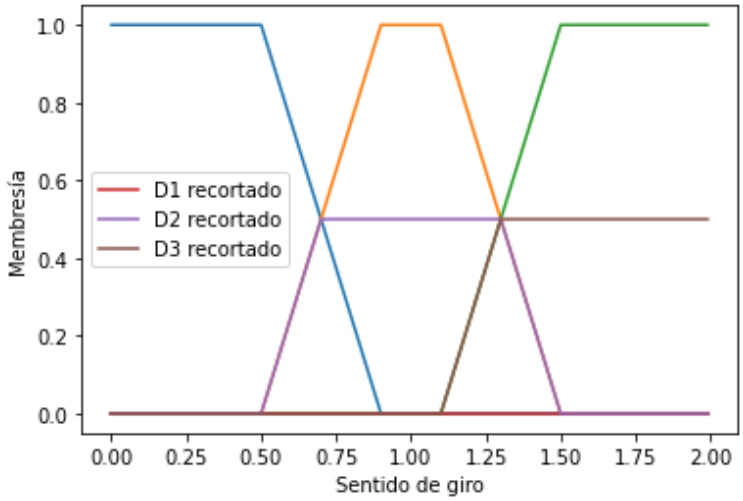


Figura 5: Cortes a los conjuntos del universo de salida "Sentido de giro".

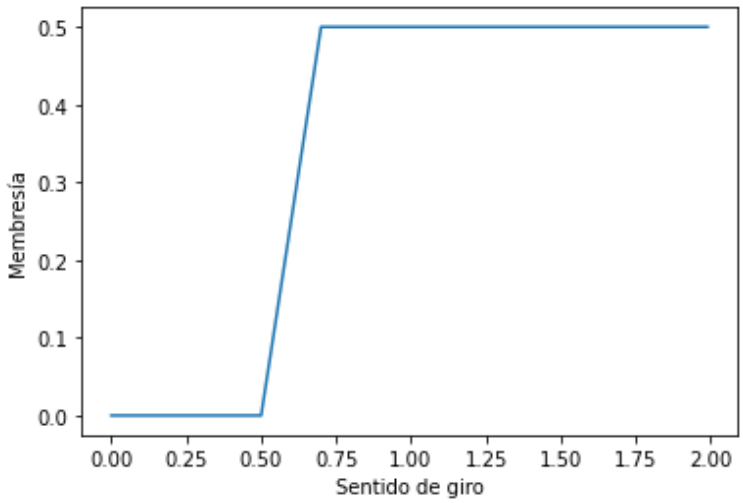


Figura 6: Unión de los conjuntos recortados

```
Ingrese el valor de la posición del generador(0-359): 135
Ingrese el valor de la veleta (0-359): 315
Tabla de inferencia:  [[0.0, 0.0, 0.0, 0.0, 0.0], [0.0,
0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.5,
0.5, 0.0, 0.0], [0.0, 0.5, 0.5, 0.0, 0.0]]
z* =  362.0305584826133  [Hz]
z* =  1.2963082437275986
```

Figura 7: Tabla de inferencia y valores desfusificados para $x = 135$ y $y = 315$.

En la siguiente sección muestro las superficies de control obtenidas.

4. Resultados

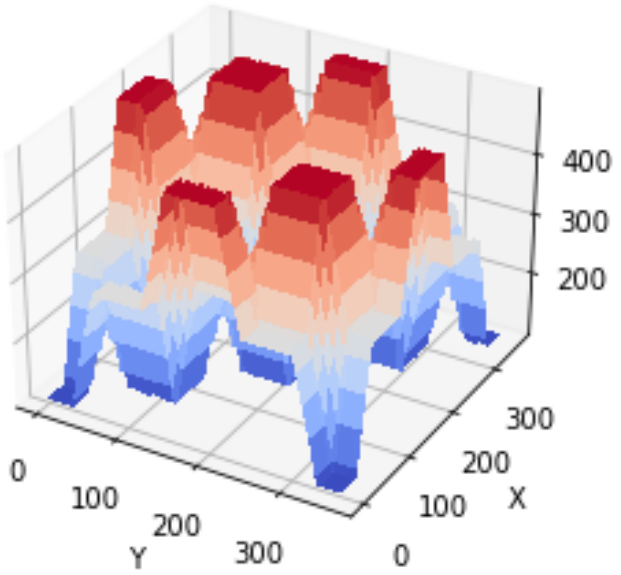


Figura 8: Superficie de control para "Velocidad de giro".

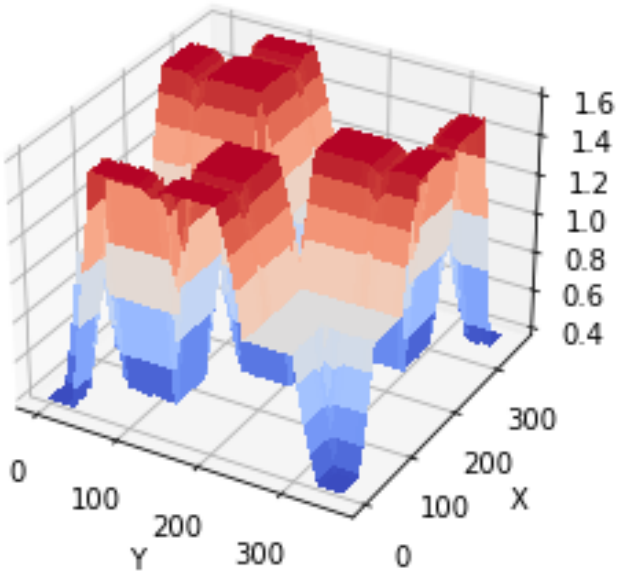


Figura 9: Superficie de control para "Sentido de giro".

5. Conclusiones

Al observar las figuras 8 y 9 se puede ver cierta similitud entre ambas superficies de control. Esto ocurre debido a que en ambas variables de salida tenemos 3 conjuntos. Sin embargo, las reglas "si - entonces" aunadas al rango en el que están definidas las variables afectan la forma de las superficies. He de resaltar que estas superficies las obtuve después de detectar un error en el uso que le di a la función *np.maximum()*, pues quise realizar la comparación de 3 conjuntos siendo que la función solo permite la comparación de 2 conjuntos a la vez como máximo. Una vez corregido este error, obtuve los resultados de desdifusificación correctos así como las superficies de control esperadas.

6. Apéndices

Código para la generación de gráficas de los conjuntos de entrada y salida caracterizados {graficas.py}

```
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(0, 359, 1)
y = np.arange(0, 359, 1)
z1 = np.arange(0, 600, 1)
z2 = np.arange(0, 2, 0.01)
#Posición del generador
A1 = np.zeros(len(x)) #Norte EG
A2 = np.zeros(len(x)) #Este G
A3 = np.zeros(len(x)) #Sur G
A4 = np.zeros(len(x)) #Oeste G
A5 = np.zeros(len(x)) #Norte OG
#Posición de la veleta
B1 = np.zeros(len(y)) #Norte EG
B2 = np.zeros(len(y)) #Este G
B3 = np.zeros(len(y)) #Sur G
B4 = np.zeros(len(y)) #Oeste G
B5 = np.zeros(len(y)) #Norte OG
#Velocidad de giro
C1 = np.zeros(len(z1)) #Baja
C2 = np.zeros(len(z1)) #Media
C3 = np.zeros(len(z1)) #Alta
#Sentido de giro
D1 = np.zeros(len(z2)) #0
D2 = np.zeros(len(z2)) #1
D3 = np.zeros(len(z2)) #2
def funcionTrapezoidal(a,b,c,d,C,r):
for i in range(len(r)):
if i <= a:
C[i] = 0
elif a <i and i <= b:
C[i] = (i - a)/(b - a)
elif b <i and i <= c:
C[i] = 1
elif c <i and i <= d:
C[i] = (d - i)/(d - c)
elif d <i:
C[i] = 0
return C
#Generación de los conjuntos difusos
#Posición del generador
A1 = funcionTrapezoidal(-1,-1,30,60,A1,x)
A2 = funcionTrapezoidal(30,60,120,150,A2,x)
A3 = funcionTrapezoidal(120,150,210,240,A3,x)
A4 = funcionTrapezoidal(210,240,300,330,A4,x)
A5 = funcionTrapezoidal(300,330,359,359,A5,x)
plt.figure(1)
plt.plot(x,A1,label = "Norte EG")
plt.plot(x,A2,label = "Este G")
```

```
plt.plot(x,A3,label = "Sur EG")
plt.plot(x,A4,label = "Oeste G")
plt.plot(x,A5,label = "Norte OG")
plt.legend()
plt.xlabel("Posición del Generador")
plt.ylabel("Membresía")
#Posición de la veleta
B1 = funcionTrapezoidal(-1,-1,30,60,B1,y)
B2 = funcionTrapezoidal(30,60,120,150,B2,y)
B3 = funcionTrapezoidal(120,150,210,240,B3,y)
B4 = funcionTrapezoidal(210,240,300,330,B4,y)
B5 = funcionTrapezoidal(300,330,359,359,B5,y)
plt.figure(2)
plt.plot(y,B1,label = "Norte EG")
plt.plot(y,B2,label = "Este G")
plt.plot(y,B3,label = "Sur EG")
plt.plot(y,B4,label = "Oeste G")
plt.plot(y,B5,label = "Norte OG")
plt.legend()
plt.xlabel("Posición de la Veleta")
plt.ylabel("Membresía")
#Velocidad de giro
C1 = funcionTrapezoidal(-1,-1,100,200,C1,z1)
C2 = funcionTrapezoidal(100,200,400,500,C2,z1)
C3 = funcionTrapezoidal(400,500,600,600,C3,z1)
plt.figure(3)
plt.plot(z1,C1,label = "Baja")
plt.plot(z1,C2,label = "Media")
plt.plot(z1,C3,label = "Alta")
plt.legend()
plt.xlabel("Velocidad [Hz]")
plt.ylabel("Membresía")
#Sentido de giro
D1 = funcionTrapezoidal(-1*100,-1*100,0.5*100,0.9*100,D1,z2)
D2 = funcionTrapezoidal(0.5*100,0.9*100,1.1*100,1.5*100,D2,z2)
D3 = funcionTrapezoidal(1.1*100,1.5*100,2*100,2*100,D3,z2)
plt.figure(4)
plt.plot(z2,D1,label = "Antihorario")
plt.plot(z2,D2,label = "No mover")
plt.plot(z2,D3,label = "Horario")
plt.legend()
plt.xlabel("Sentido de Giro")
plt.ylabel("Membresía")
#Obtención de los grados de pertenencia a los conjuntos de los universos de entrada
valx = int(input("Ingrese el valor de la posición del generador(0-359): "))
memA = [A1[valx],A2[valx],A3[valx],A4[valx],A5[valx]]
valy = int(input("Ingrese el valor de la veleta (0-359): "))
memB = [B1[valy],B2[valy],B3[valy],B4[valy],B5[valy]]
#Composición max - min (Evaluación de las reglas "si-entonces")
#Tablas de inferencia (intersección - min)
tabla = [[np.minimum(memA[0],memB[0]),
np.minimum(memA[1],memB[0]),
np.minimum(memA[2],memB[0]),
np.minimum(memA[3],memB[0]),
```

```
np.minimum(memA[4],memB[0])],
[np.minimum(memA[0],memB[1]),
np.minimum(memA[1],memB[1]),
np.minimum(memA[2],memB[1]),
np.minimum(memA[3],memB[1]),
np.minimum(memA[4],memB[1])],
[np.minimum(memA[0],memB[2]),
np.minimum(memA[1],memB[2]),
np.minimum(memA[2],memB[2]),
np.minimum(memA[3],memB[2]),
np.minimum(memA[4],memB[2])],
[np.minimum(memA[0],memB[3]),
np.minimum(memA[1],memB[3]),
np.minimum(memA[2],memB[3]),
np.minimum(memA[3],memB[3]),
np.minimum(memA[4],memB[3])],
[np.minimum(memA[0],memB[4]),
np.minimum(memA[1],memB[4]),
np.minimum(memA[2],memB[4]),
np.minimum(memA[3],memB[4])
,np.minimum(memA[4],memB[4])]]
print("Tabla de inferencia: ",str(tabla))
#Agregación - max hacia Velocidad de giro
agregacionC1 = max(tabla[0][0],max(tabla[1][1],
max(tabla[2][2],
max(tabla[3][3],max(tabla[4][4],
max(tabla[4][0],tabla[0][4])))))) #Baja
agregacionC2 = max(tabla[0][1],max(tabla[0][3],
max(tabla[1][0],
max(tabla[1][2],max(tabla[1][4],
max(tabla[2][1],
max(tabla[2][3],max(tabla[3][0],
max(tabla[3][2],
max(tabla[3][4],max(tabla[4][1],
tabla[4][3])))))))))) #Media
agregacionC3 = max(tabla[0][2],max(tabla[1][3],
max(tabla[2][0],
max(tabla[2][4],max(tabla[3][1],
tabla[4][2])))))) #Alta
#Agregación - max hacia Sentido de giro
agregacionD1 = max(tabla[0][0],max(tabla[1][1],
max(tabla[2][2],max(tabla[3][3],max(tabla[4][4],
max(tabla[4][0],tabla[0][4])))))) #Baja
agregacionD2 = max(tabla[0][1],max(tabla[0][2],
max(tabla[1][2],max(tabla[2][3],max(tabla[2][4],
max(tabla[3][0],max(tabla[3][1],
max(tabla[3][4],tabla[4][1])))))))) #Media
agregacionD3 = max(tabla[0][3],max(tabla[1][0],
max(tabla[1][3],max(tabla[1][4],max(tabla[2][0],
max(tabla[2][1],max(tabla[3][2],
max(tabla[4][2],tabla[4][3])))))))) #Alta
aux1 = agregacionC1*np.ones(len(C1))
aux2 = agregacionC2*np.ones(len(C2))
aux3 = agregacionC3*np.ones(len(C3))
```

```
aux4 = agregacionD1*np.ones(len(D1))
aux5 = agregacionD2*np.ones(len(D2))
aux6 = agregacionD3*np.ones(len(D3))
C1_r = np.minimum(C1,aux1)
C2_r = np.minimum(C2,aux2)
C3_r = np.minimum(C3,aux3)
D1_r = np.minimum(D1,aux4)
D2_r = np.minimum(D2,aux5)
D3_r = np.minimum(D3,aux6)
plt.figure(5)
plt.plot(z1,C1,z1,C2,z1,C3)
plt.plot(z1,C1_r,label = "C1 recortado")
plt.plot(z1,C2_r,label = "C2 recortado")
plt.plot(z1,C3_r,label = "C3 recortado")
plt.xlabel("Velocidad [Hz]")
plt.ylabel("Membresía")
plt.legend()
plt.figure(6)
plt.plot(z2,D1,z2,D2,z2,D3)
plt.plot(z2,D1_r,label = "D1 recortado")
plt.plot(z2,D2_r,label = "D2 recortado")
plt.plot(z2,D3_r,label = "D3 recortado")
plt.xlabel("Sentido de giro")
plt.ylabel("Membresía")
plt.legend()
#Dedifusificación
#Velocidad de Giro
z_d = 0
nume = 0
deno = 0
CT1 = np.maximum(C1_r,np.maximum(C2_r,C3_r))
plt.figure(7)
plt.plot(z1,CT1)
plt.xlabel("Velocidad [Hz]")
plt.ylabel("Membresía")
for i in range(len(z1)):
nume += CT1[i]*i
deno += CT1[i]
z_d = nume/deno
print("z* = ", str(z_d), " [Hz]")
#Sentido de giro
z_d = 0
nume = 0
deno = 0
CT2 = np.maximum(D1_r,np.maximum(D2_r,D3_r))
plt.figure(8)
plt.plot(z2,CT2)
plt.xlabel("Sentido de giro")
plt.ylabel("Membresía")
for i in range(len(z2)):
nume += CT2[i]*i/100
deno += CT2[i]
z_d = nume/deno
print("z* = ", str(z_d))
```

Código para la generación de superficies de control {3d.py}

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
x = np.arange(0, 359, 1)
y = np.arange(0, 359, 1)
z1 = np.arange(0, 600, 1)
z2 = np.arange(0, 2, 0.01)
X, Y = np.meshgrid(x, y)
#Posición del generador
A1 = np.zeros(len(x)) #Norte EG
A2 = np.zeros(len(x)) #Este G
A3 = np.zeros(len(x)) #Sur G
A4 = np.zeros(len(x)) #Oeste G
A5 = np.zeros(len(x)) #Norte OG
#Posición de la veleta
B1 = np.zeros(len(y)) #Norte EG
B2 = np.zeros(len(y)) #Este G
B3 = np.zeros(len(y)) #Sur G
B4 = np.zeros(len(y)) #Oeste G
B5 = np.zeros(len(y)) #Norte OG
#Velocidad de giro
C1 = np.zeros(len(z1)) #Baja
C2 = np.zeros(len(z1)) #Media
C3 = np.zeros(len(z1)) #Alta
#Sentido de giro
D1 = np.zeros(len(z2)) #0
D2 = np.zeros(len(z2)) #1
D3 = np.zeros(len(z2)) #2
def funcionTrapezoidal(a,b,c,d,C,r):
for i in range(len(r)):
if i <= a:
C[i] = 0
elif a <i and i <= b:
C[i] = (i - a)/(b - a)
elif b <i and i <= c:
C[i] = 1
elif c <i and i <= d:
C[i] = (d - i)/(d - c)
elif d <i:
C[i] = 0
return C
#Generación de los conjuntos difusos
#Posición del generador
A1 = funcionTrapezoidal(-1,-1,30,60,A1,x)
A2 = funcionTrapezoidal(30,60,120,150,A2,x)
A3 = funcionTrapezoidal(120,150,210,240,A3,x)
A4 = funcionTrapezoidal(210,240,300,330,A4,x)
A5 = funcionTrapezoidal(300,330,359,359,A5,x)
#Posición de la veleta
B1 = funcionTrapezoidal(-1,-1,30,60,B1,y)
B2 = funcionTrapezoidal(30,60,120,150,B2,y)
B3 = funcionTrapezoidal(120,150,210,240,B3,y)
```

```
B4 = funcionTrapezoidal(210,240,300,330,B4,y)
B5 = funcionTrapezoidal(300,330,359,359,B5,y)
#Velocidad de giro
C1 = funcionTrapezoidal(-1,-1,170,230,C1,z1)
C2 = funcionTrapezoidal(170,230,370,430,C2,z1)
C3 = funcionTrapezoidal(370,430,600,600,C3,z1)
#Sentido de giro
D1 = funcionTrapezoidal(-1*100,-1*100,0.5*100,1*100,D1,z2)
D2 = funcionTrapezoidal(0.7*100,0.9*100,1.1*100,1.3*100,D2,z2)
D3 = funcionTrapezoidal(1*100,1.5*100,2*100,2*100,D3,z2)
CZ1 = np.zeros((len(x),len(y)))
CZ2 = np.zeros((len(x),len(y)))
#Obtención de los grados de pertenencia a los conjuntos de los universos de entrada
def difusificacion(valx,valy):
memA = [A1[valx],A2[valx],A3[valx],A4[valx],A5[valx]]
memB = [B1[valy],B2[valy],B3[valy],B4[valy],B5[valy]]
#Composición max - min (Evaluación de las reglas "si-entonces")
#Tabla de inferencia (intersección - min)
tabla = [[np.minimum(memA[0],memB[0]),np.minimum(memA[1],memB[0]),
np.minimum(memA[2],memB[0]),np.minimum(memA[3],memB[0]),
np.minimum(memA[4],memB[0])],
[np.minimum(memA[0],memB[1]),np.minimum(memA[1],memB[1]),
np.minimum(memA[2],memB[1]),np.minimum(memA[3],memB[1]),
np.minimum(memA[4],memB[1])],
[np.minimum(memA[0],memB[2]),np.minimum(memA[1],memB[2]),
np.minimum(memA[2],memB[2]),np.minimum(memA[3],memB[2]),
np.minimum(memA[4],memB[2])],
[np.minimum(memA[0],memB[3]),np.minimum(memA[1],memB[3]),
np.minimum(memA[2],memB[3]),np.minimum(memA[3],memB[3]),
np.minimum(memA[4],memB[3])],
[np.minimum(memA[0],memB[4]),np.minimum(memA[1],memB[4]),
np.minimum(memA[2],memB[4]),np.minimum(memA[3],memB[4]),
np.minimum(memA[4],memB[4])]]
#Agregación - max hacia Velocidad de giro
agregacionC1 = max(tabla[0][0],max(tabla[1][1],
max(tabla[2][2],
max(tabla[3][3],max(tabla[4][4],
max(tabla[4][0],tabla[0][4])))))) #Baja
agregacionC2 = max(tabla[0][1],max(tabla[0][3],
max(tabla[1][0],
max(tabla[1][2],max(tabla[1][4],
max(tabla[2][1],
max(tabla[2][3],
max(tabla[3][0],max(tabla[3][2],
max(tabla[3][4],
max(tabla[4][1],
tabla[4][3])))))))))) #Media
agregacionC3 = max(tabla[0][2],max(tabla[1][3],
max(tabla[2][0],
max(tabla[2][4],max(tabla[3][1],tabla[4][2])))))) #Alta
#Agregación - max hacia Sentido de giro
agregacionD1 = max(tabla[0][0],max(tabla[1][1],
max(tabla[2][2],
max(tabla[3][3],max(tabla[4][4],
```

```
max(tabla[4][0],tabla[0][4]))))) #Baja
agregacionD2 = max(tabla[0][1],max(tabla[0][2],
max(tabla[1][2],
max(tabla[2][3],max(tabla[2][4],
max(tabla[3][4],max(tabla[3][0],
max(tabla[3][1],tabla[4][1])))))))) #Media
agregacionD3 = max(tabla[0][3],max(tabla[1][0],
max(tabla[1][3],
max(tabla[1][4],max(tabla[2][0],
max(tabla[2][1],max(tabla[3][2],
max(tabla[4][2],tabla[4][3])))))))) #Alta
aux1 = agregacionC1*np.ones(len(C1))
aux2 = agregacionC2*np.ones(len(C2))
aux3 = agregacionC3*np.ones(len(C3))
aux4 = agregacionD1*np.ones(len(D1))
aux5 = agregacionD2*np.ones(len(D2))
aux6 = agregacionD3*np.ones(len(D3))
C1_r = np.minimum(C1,aux1)
C2_r = np.minimum(C2,aux2)
C3_r = np.minimum(C3,aux3)
D1_r = np.minimum(D1,aux4)
D2_r = np.minimum(D2,aux5)
D3_r = np.minimum(D3,aux6)
z_d = np.zeros((1,2))
nume = 0
deno = 0
CT1 = np.maximum(C1_r,np.maximum(C2_r,C3_r))
CT2 = np.maximum(D1_r,np.maximum(D2_r,D3_r))
for i in range(len(z1)):
nume += CT1[i]*i
deno += CT1[i]
z_d[0,0] = nume/deno
nume = 0
deno = 0
for i in range(len(z2)):
nume += CT2[i]*i/100
deno += CT2[i]
z_d[0,1] = nume/deno
return z_d
for i in range(len(x)):
for j in range(len(y)):
CZ1[i,j] = difusificacion(i,j)[0][0]
CZ2[i,j] = difusificacion(i,j)[0][1]
fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
ax.set_xlabel('Y')
ax.set_ylabel('X')
ax.set_zlabel('Z')
surf = ax.plot_surface(X, Y, CZ1, cmap=cm.coolwarm,
linewidth=0, antialiased=False)
fig2, ax2 = plt.subplots(subplot_kw={"projection": "3d"})
ax2.set_xlabel('Y')
ax2.set_ylabel('X')
ax2.set_zlabel('Z')
surf = ax2.plot_surface(X, Y, CZ2, cmap=cm.coolwarm, linewidth=0, antialiased=False)
```

Referencias

[1] B. Granados Rojas P. N. Cortez Herrera Y. E. González Navarro. Cómo Diseñar un Sistema de Inferencia Difusa de Tipo Mamdani. 1st. La Laguna Ticomán, CDMX, MX: Boletín UPIITA, 2019.