CONTROL BORROSO

Sistemas de control inteligente

Universidad de Alcalá de Henares

Luis Alejandro Cabanillas Prudencio
Daniel López Moreno

PARTE 1

1.1: LLEGAR A UN PUNTO

En este apartado se plantea el control de posición de un robot móvil dentro de un entorno sin obstáculos de dimensiones 10x10 metros y cuyo origen de coordenadas se encuentra en el centro geométrico del entorno.

El objetivo de esta parte de la práctica es **diseñar un controlador borroso** de manera que el robot sea capaz de alcanzar una posición determinada por las referencias de posición refx y refy.

El esquema de control es el mismo que hemos utilizado en anteriores prácticas, con un bloque Position_errors, un bloque de Control y un bloque Robot. Sin embargo, a diferencia de otras prácticas, el bloque de control lo generaremos utilizando un controlador borroso tal y como hemos aprendido en clase.

Para generar este controlador utilizaremos el comando de Matlab **"fuzzy"**, el cual nos proporciona una interfaz desde la cual podemos configurar nuestro controlador borroso.

Para este controlador mantendremos las operaciones AND/OR y el desborrosificador por defecto. A continuación, añadiremos a nuestro controlador **2 entradas** (E_d y E_theta) **y 2 salidas** (V y W), cada una de ellas con distintos rangos:

$$Ed \in [0, 10], E_\theta \in [-\pi, \pi], V \in [0, 2], W \in [-1, 1]$$

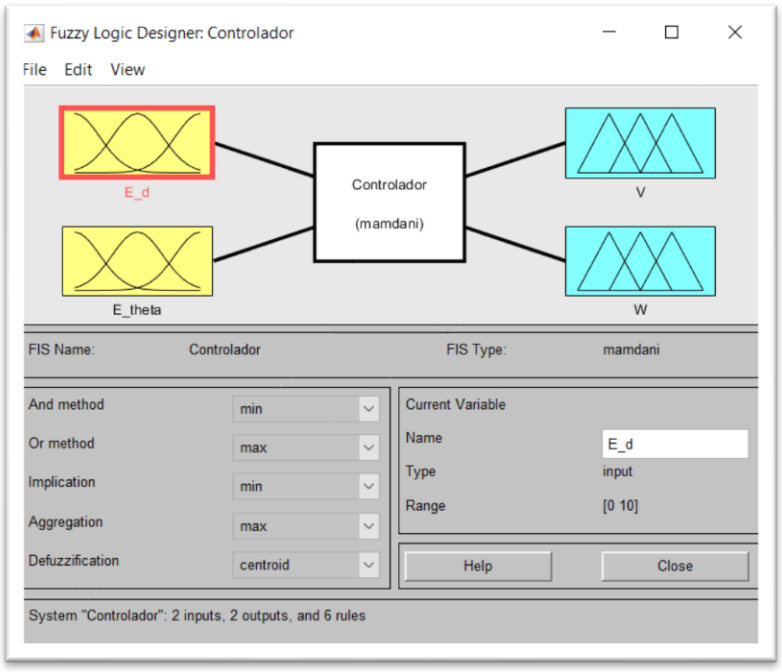

Figura 1: Diseño de nuestro controlador borroso

Ahora modificaremos las **funciones de pertenencia** de cada una de las variables como se puede ver en la Figura 2:

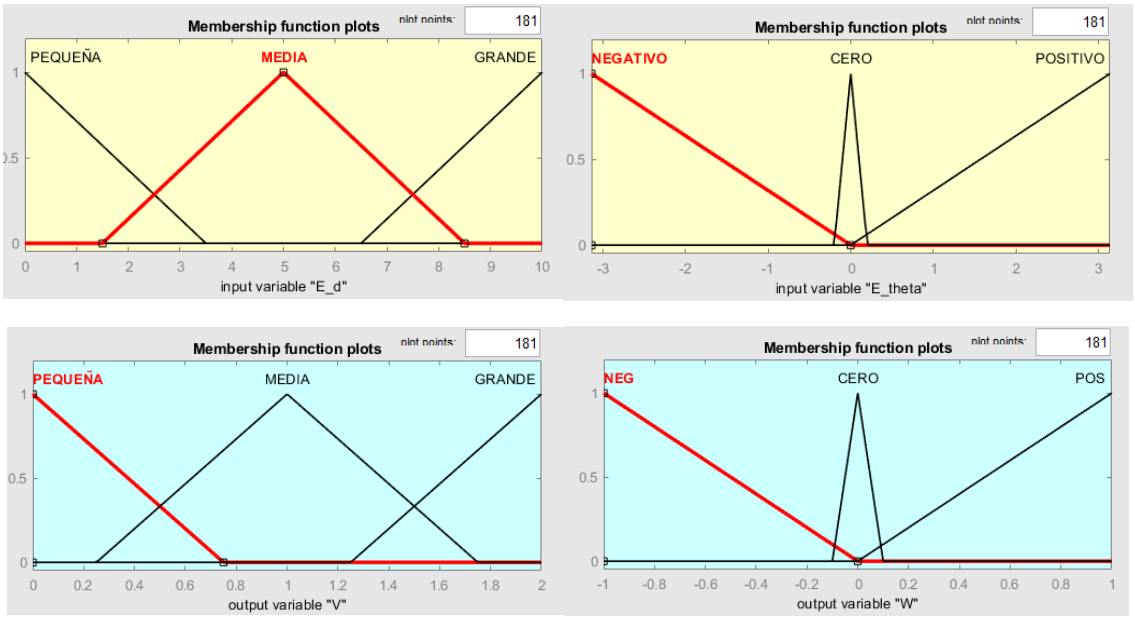

Figura 2: Funciones de pertenencia

Para determinar estas funciones de pertenencia hemos realizado diferentes pruebas hasta lograr un recorrido perfecto por parte del robot hasta el objetivo. Inicialmente las pruebas se realizaron controlando solo el giro y manteniendo una **velocidad constante** con valor **V=0.3**. Una vez logrado alcanzar el objetivo correctamente, añadimos las funciones de pertenencia para las variables de velocidad lineal (E_d y V).

La función de pertenencia **CERO** tanto en E_theta como en W sirven para evitar errores de ejecución ya que el **valor 0 no tiene grado de pertenencia** tanto en la función NEG como en POS.

Variable	Rango	Nombre de función	Forma	Parámetros
E_d	[0, 10]	**PEQUEÑA**	Trimf	[0 0 3.5]
		MEDIA	Trimf	[1.5 5 8.5]
		GRANDE	Trimf	[6.5 10 10]
E_theta	[-pi, pi]	**NEGATIVO**	Trimf	[-3.142 -3.142 0]
		CERO	Trimf	[-0.2 0 0.2]
		POSITIVO	Trimf	[0 3.142 3.142]
V	[0, 2]	**PEQUEÑA**	Trimf	[0 0 0.75]
		MEDIA	Trimf	[0.25 1 1.75]
		GRANDE	Trimf	[1.25 2 2]
W	[-1, 1]	**NEG**	Trimf	[-1 -1 0]
		CERO	Trimf	[-0.1 0 0.1]
		POS	Trimf	[0 1 1]

Tabla 1: Definición de las funciones de pertenencia

Una vez hemos definido correctamente las funciones de pertenencia para cada uno de los conjuntos es momento de **generar las reglas** que seguirá nuestro controlador. Utilizaremos estas reglas para **determinar el valor de nuestras variables de salida** en función de los valores de entrada:

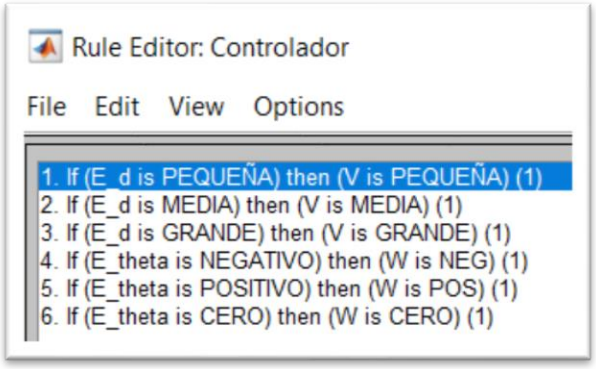

Figura 3: Reglas de nuestro controlador borroso

Por último, generaremos un bloque de *Simulink* con el nombre de **"Controlador Borroso"** que introduciremos en nuestro esquema de control mencionado anteriormente sustituyéndolo por el bloque Control tal y como se muestra en la Figura 4:

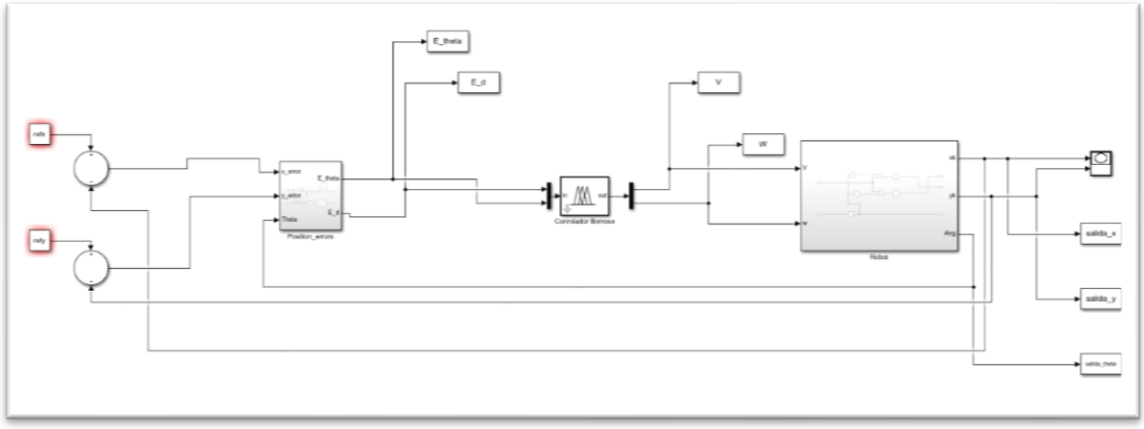

Figura 4: Esquema de control con el bloque Controlador Borroso

Ejecutaremos el código de prueba de la simulación y comprobaremos si nuestro Controlador Borroso es capaz de llevar el robot hasta el punto de destino indicado en las variables refx y refy:

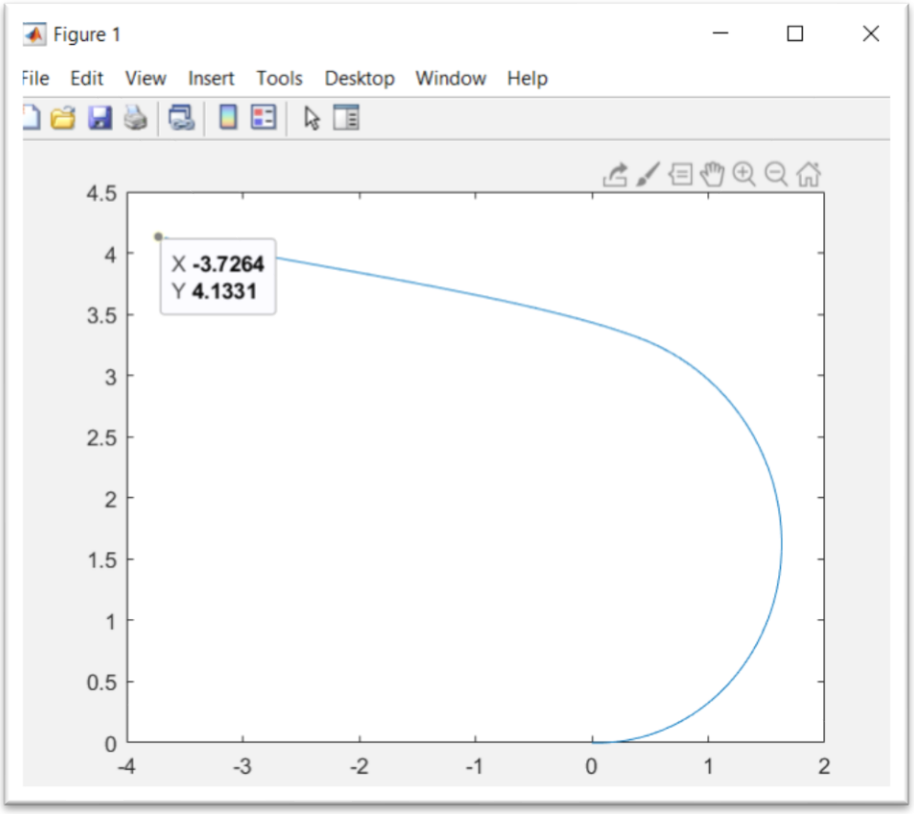

Figura 5: Trayectoria resultado de la simulación

Como podemos observar en la Figura 5, los valores objetivo a alcanzar por el robot son **refx=-3.7301** y **refy= 4.1338,** y nuestro Controlador Borroso ha sido capaz de alcanzarlo a la perfección realizando un giro desde la posición inicial.

1.2: COPIAR LA TRAYECTORIA DEL TRAJECTORY_GENERATION

En este apartado vamos a sustituir las referencias de posición de nuestro esquema de control (refx y refy) por el **bloque "Trajectory_control"** que ya hemos utilizado en anteriores prácticas y genera la siguiente trayectoria:

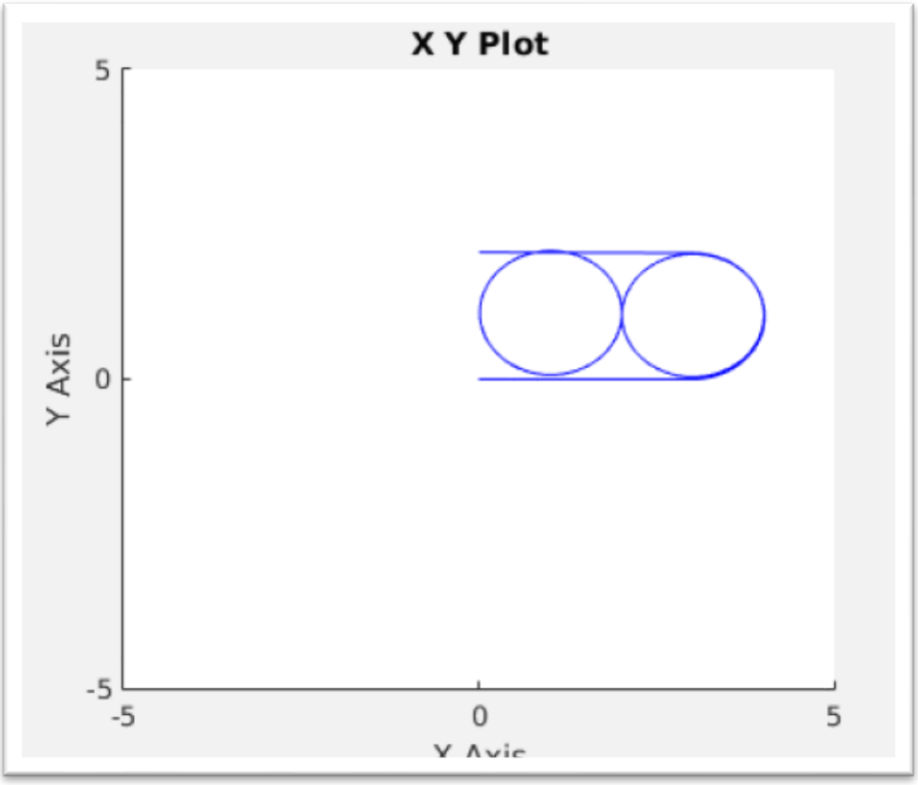

Figura 6: Trayectoria generada por el bloque "Trajectory_generation"

Tras introducir el bloque en nuestro esquema nos encontraremos con el esquema de *Simulink* mostrado en la Figura 7:

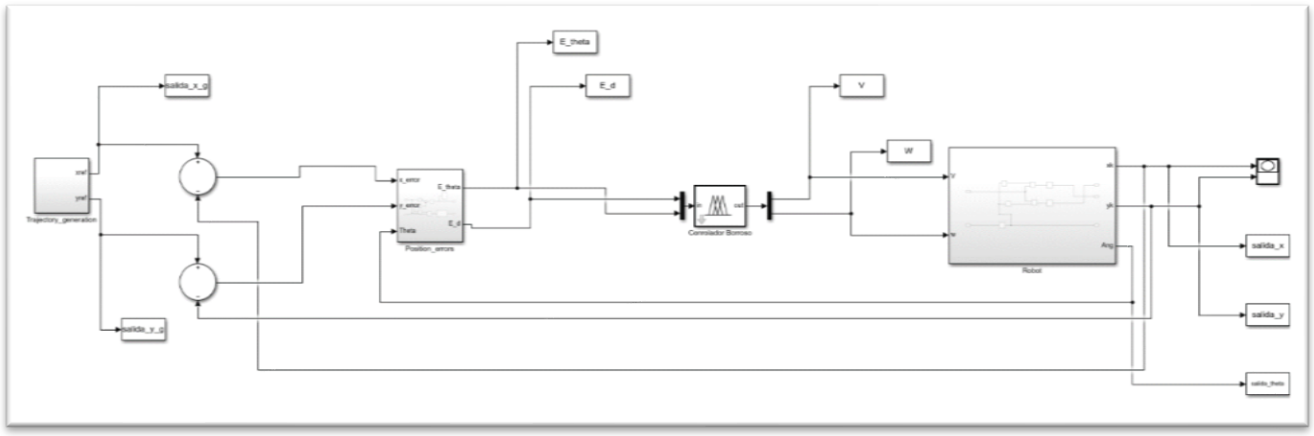

Figura 7: Esquema con el bloque Trajectory_generation

Al ejecutar una simulación con este esquema podemos observar como nuestro Controlador Borroso es capaz de imitar perfectamente la trayectoria creada por el bloque recién introducido:

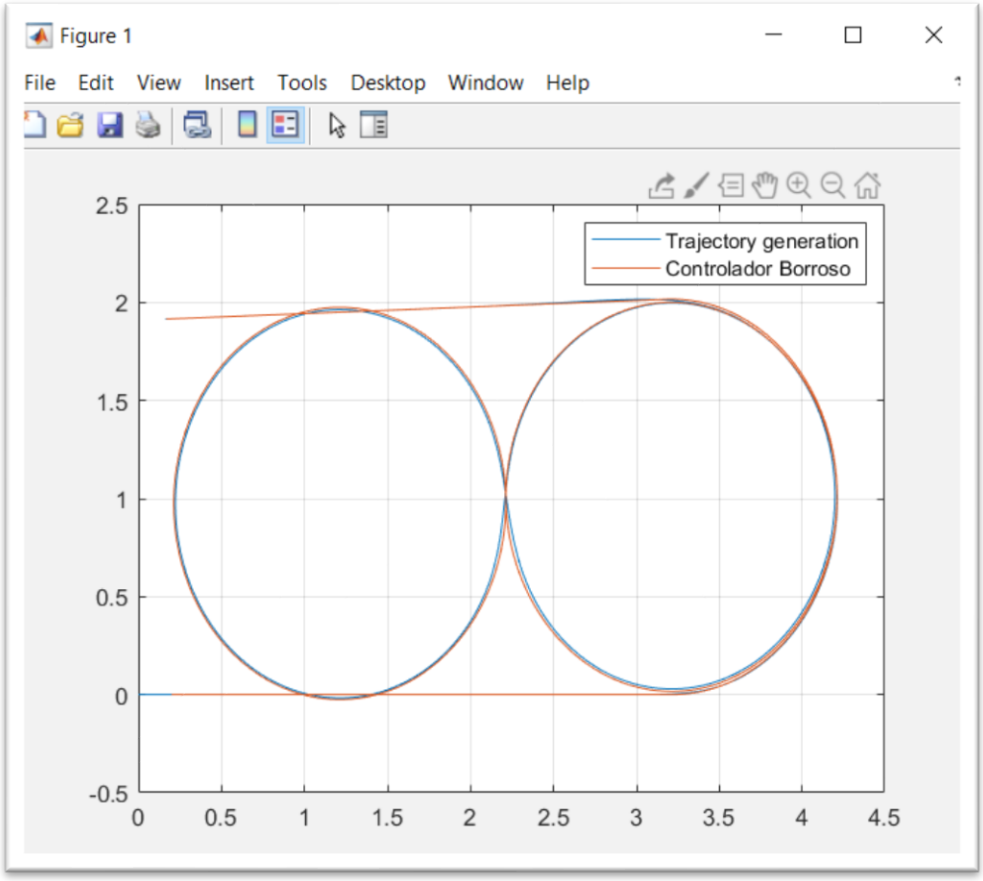

Figura 8: Trayectorias resultado de la simulación

Como se puede comprobar en la Figura 8, nuestro script de simulación compara en una misma figura ambas trayectorias en dos colores distintos: **Azul** (Trajectory_generation) y **Naranja** (Controlador Borroso). Finalmente, nuestro bloque de Controlador Borroso ha sido completamente capaz de **imitar el mismo recorrido** que el bloque generador haciendo uso de **las reglas** y **las funciones de pertenencia** generadas anteriormente.

PARTE 2

2.1 ALCANZAR UN PUNTO DESTINO

El objetivo de esta parte de la práctica es diseñar un controlador borroso de manera que el robot sea capaz de alcanzar una posición determinada por las referencias de posición refx y refy al mismo tiempo que se evita colisionar con un obstáculo que se encuentra en la posición obsx y obsy.
Para lograr este objetivo generaremos el esquema de *Simulink* mostrado en la Figura 9. El nuevo bloque Position_errors_obs es exactamente igual que el bloque Position_error, a excepción de que hemos retirado la condición de parada para evitar colisiones y que nuestro robot no llegue al objetivo.

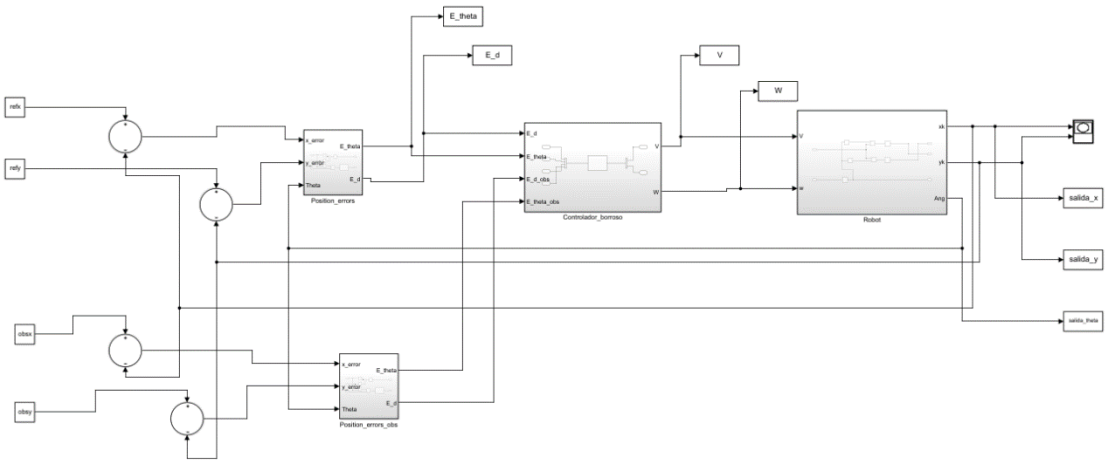

Figura 9: Esquema general de un control de posición con evitación de obstáculos

Esta vez crearemos un nuevo Controlador Borroso con la función "fuzzy" en Matlab. A diferencia del controlador borroso utilizado en el anterior apartado de la práctica, el nuevo controlador contará con 4 entradas, 2 para los errores de referencia (E_d y E_θ) y otras 2 para los errores respecto al obstáculo (E_{d_obs} y E_{θ_obs}). El número de salidas se mantendrá igual, 2 salidas para velocidad lineal V y velocidad angular W.

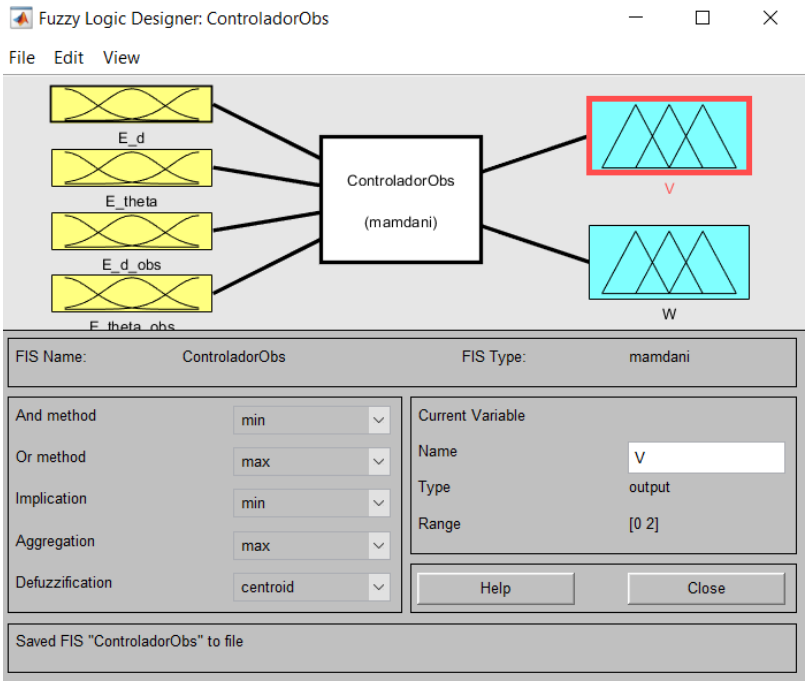

Figura 10: Diseño del nuevo controlador borroso.

A continuación, es necesario definir las funciones de pertenencia de las nuevas entradas (E_{d_obs} y E_{θ_obs}). Como hemos introducido 2 nuevas variables de entrada, es posible que haya que realizar ligeros cambios en las funciones de pertenencia de las variables ya existentes. En la Figura 11 se pueden apreciar todas las funciones de pertenencia:

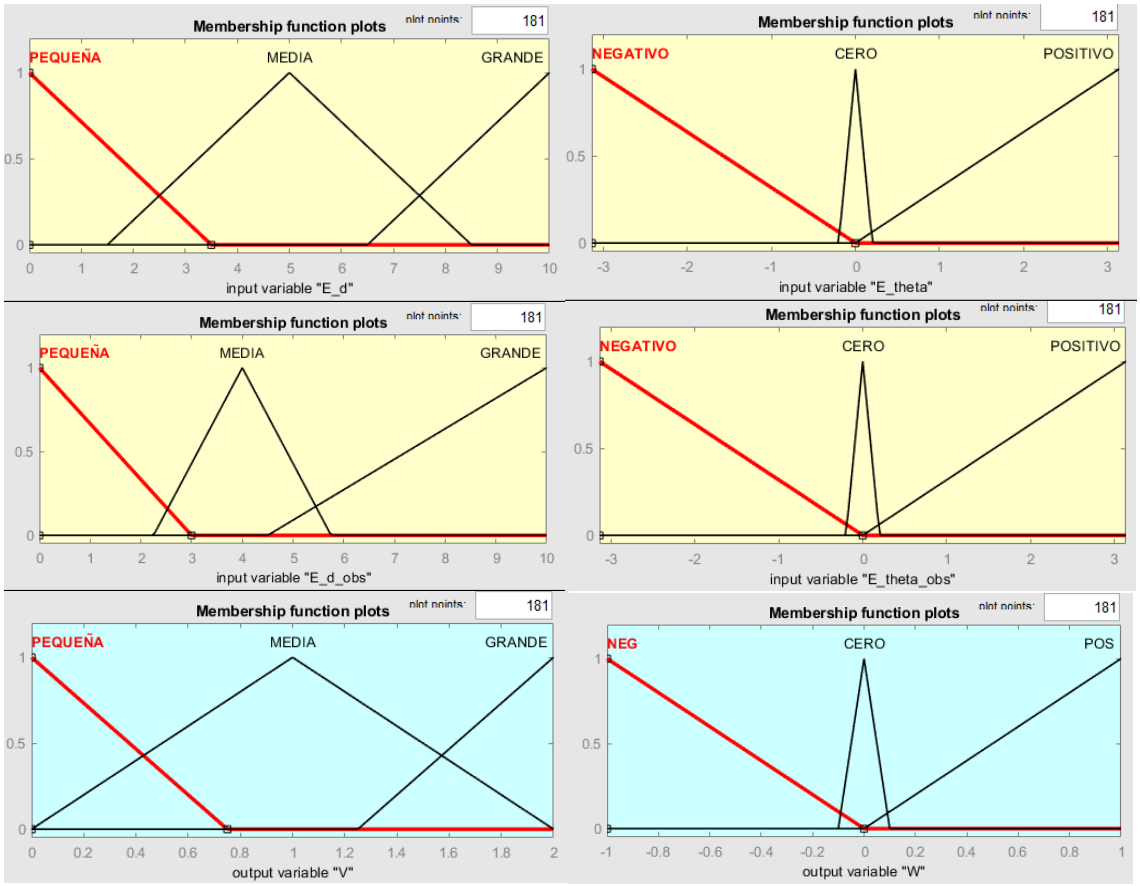

Figura 11: Nuevas funciones de pertenencia para las seis variables.

Variable	Rango	Nombre de función	Forma	Parámetros
E_d	[0, 10]	**PEQUEÑA**	Trimf	[0 0 3.5]
		MEDIA	Trimf	[1.5 5 8.5]
		GRANDE	Trimf	[6.5 10 10]
E_theta	[-pi, pi]	**NEGATIVO**	Trimf	[-3.142 -3.142 0]
		CERO	Trimf	[-0.2 0 0.2]
		POSITIVO	Trimf	[0 3.142 3.142]
E_d_obs	[0, 10]	**PEQUEÑA**	Trimf	[0 0 3]
		MEDIA	Trimf	[2.25 4 5.75]
		GRANDE	Trimf	[4.5 10 10]
E_theta_obs	[-pi, pi]	**NEGATIVO**	Trimf	[-3.142 -3.142 0]
		CERO	Trimf	[-0.2 0 0.2]
		POSITIVO	Trimf	[0 3.142 3.142]
V	[0, 2]	**PEQUEÑA**	Trimf	[0 0 0.75]
		MEDIA	Trimf	[0 1 2]
		GRANDE	Trimf	[1.25 2 2]
W	[-1, 1]	**NEG**	Trimf	[-1 -1 0]
		CERO	Trimf	[-0.1 0 0.1]
		POS	Trimf	[0 1 1]

Tabla 2: Definición de las funciones de pertenencia

Como hemos agregado 2 nuevas entradas, los valores de salida ya no dependerán únicamente del valor de E_d y E_θ, por lo que deberemos modificar las reglas del controlador para incorporar el comportamiento con respecto a las nuevas variables correspondientes al obstáculo.

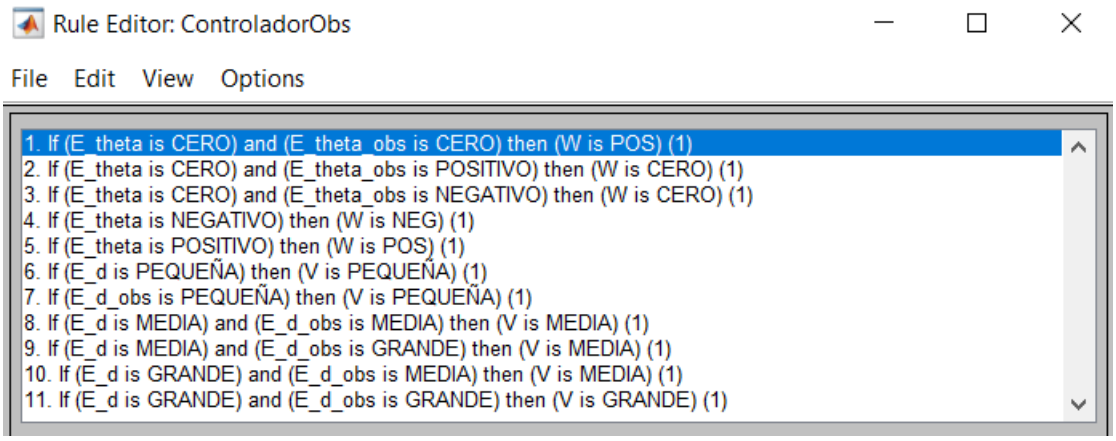

Figura 12: Reglas del nuevo controlador borroso.

V			
E_d_obs\E_d	PEQUEÑA	MEDIA	GRANDE
PEQUEÑA	PEQUEÑA	PEQUEÑA	PEQUEÑA
MEDIA	PEQUEÑA	MEDIA	MEDIA
GRANDE	PEQUEÑA	MEDIA	GRANDE

Tabla 3: Matriz de reglas para la velocidad lineal V.

W			
E_theta_obs\E_theta	NEGATIVO	CERO	POSITIVO
NEGATIVO	NEG	CERO	POS
CERO	NEG	POS	POS
POSITIVO	NEG	CERO	POS

Tabla 4: Matriz de reglas para la velocidad angular W*.

*En el caso de E_theta_obs= CERO y E_theta=CERO, hemos elegido que la salida sea POS, pero valdría también NEG, son intercambiables.

Una vez hemos configurado nuestro controlador, lo añadiremos a nuestro esquema de *Simulink* y ejecutaremos un código de simulación con la referencia en el punto R(5, 5) y el obstáculo en el punto O(2.5, 2.5). En la Figura 13 podemos apreciar la trayectoria que ha seguido nuestro robot hasta el punto de referencia evitando el obstáculo:

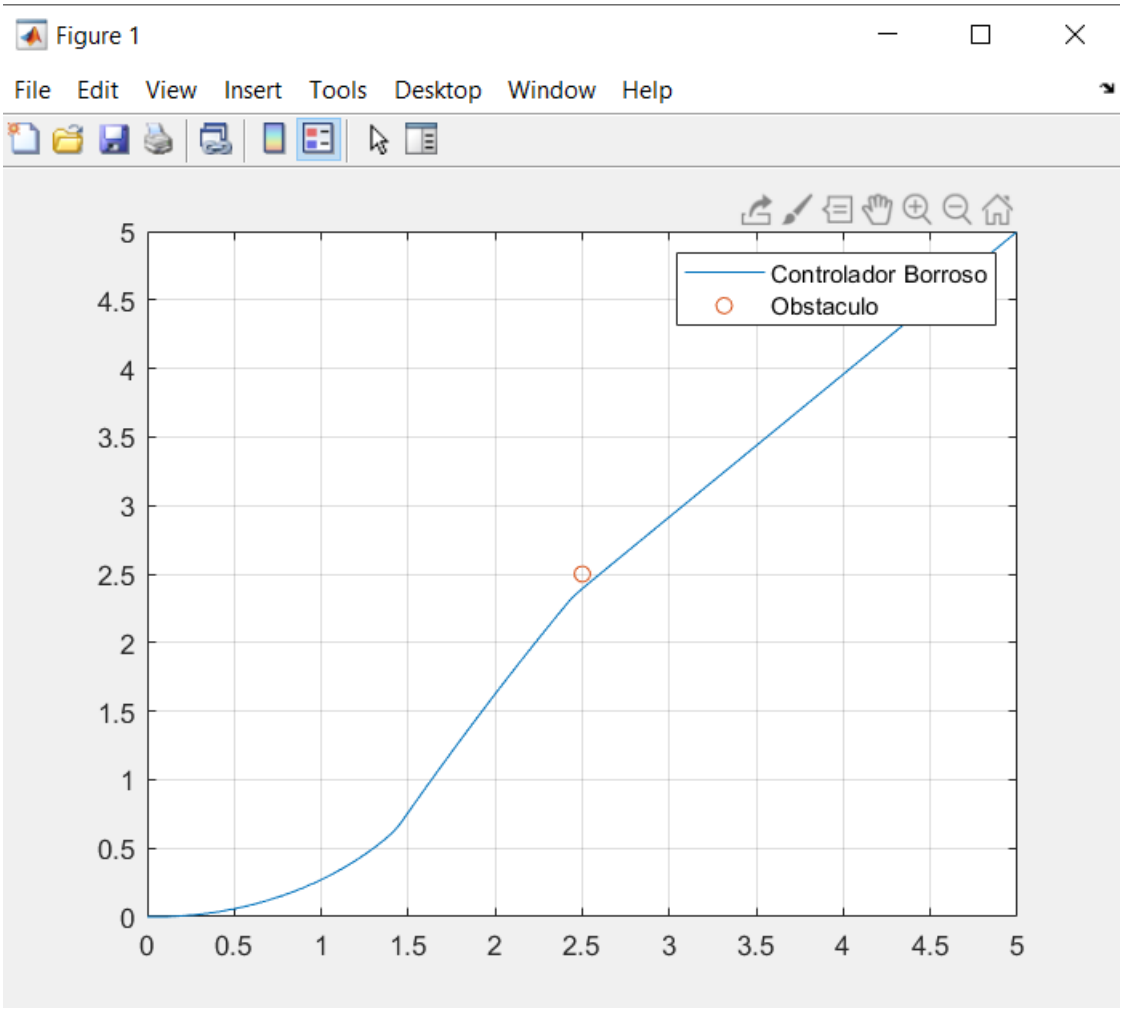

Figura 13: Trayectoria seguida por el robot evitando el obstáculo.

Como se puede apreciar, el robot realiza la trayectoria completa desde el punto inicial I(0, 0) hasta la referencia R(5, 5) **esquivando el obstáculo** situado en el punto O(2.5, 2.5) sin problemas.

2.2 SEGUIMIENTO DE TRAJECTORY_GENERATION CON OBSTÁCULO

Al igual que en el apartado 1.2, intercambiaremos los parámetros refx y refy por el bloque "Trajectory_generation" e introduciremos un obstáculo en la trayectoria generada por el bloque para comprobar si nuestro Controlador Borroso es capaz de evitarlo y reincorporarse a la trayectoria original.

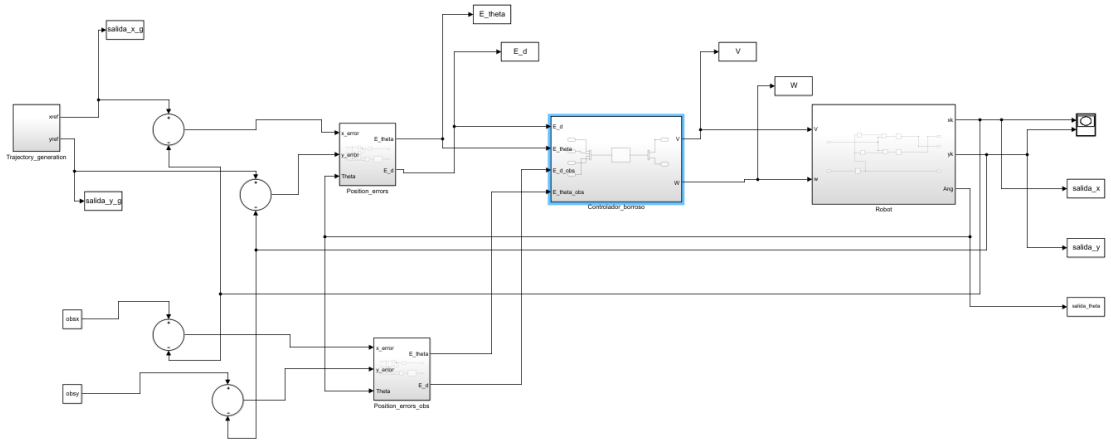

Figura 14: Esquema con Trajectory_generation y evitación de obstáculos

Realizaremos distintas pruebas con obstáculos en distintas posiciones. Para empezar, colocaremos un obstáculo en el punto $O_1(2, 0)$. Ejecutaremos el siguiente código:

```
clear all;
N=1;
x_0=0.2;
y_0=0;
th_0=0;
for i=1:N
    %Tiempo de muestreo
    Ts=100e-3
    % Referencia x-y de posicion
    refx=5;
    refy=5;
    obsx=1;
    obsy=0;
    % Ejecutar Simulacion
    sim('EvitarObstaculo.slx')
    % Mostrar
    x=salida_x.signals.values;
    y=salida_y.signals.values;
    figure;
    plot(x,y);
    hold on;
    x1=salida_x_g.signals.values;
    y1=salida_y_g.signals.values;
    plot(x1,y1);
    plot(obsx,obsy,'o');
    legend("Controlador Borroso","Trajectory_generation","Obstaculo");
    hold off;
    grid on;
end
```

Figura 15: Código de la simulación

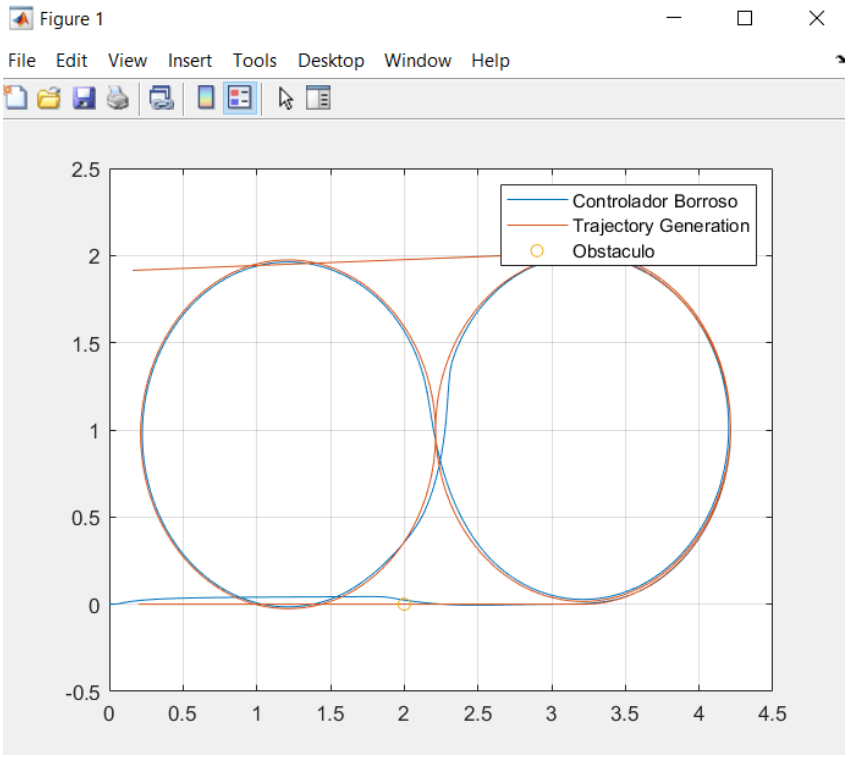

Figura 16: Resultado de esquivar el obstáculo $O_1(2, 0)$.

Nuestro controlador borroso ha seguido de cerca la trayectoria generada esquivando el obstáculo y una vez lo ha esquivado se ha adaptado perfectamente de nuevo a la trayectoria hasta el final.

Ahora realizaremos una segunda prueba colocando el obstáculo en el punto en el que se cruzan los 2 círculos generados en la trayectoria, el punto $O_2(2.2, 1)$.

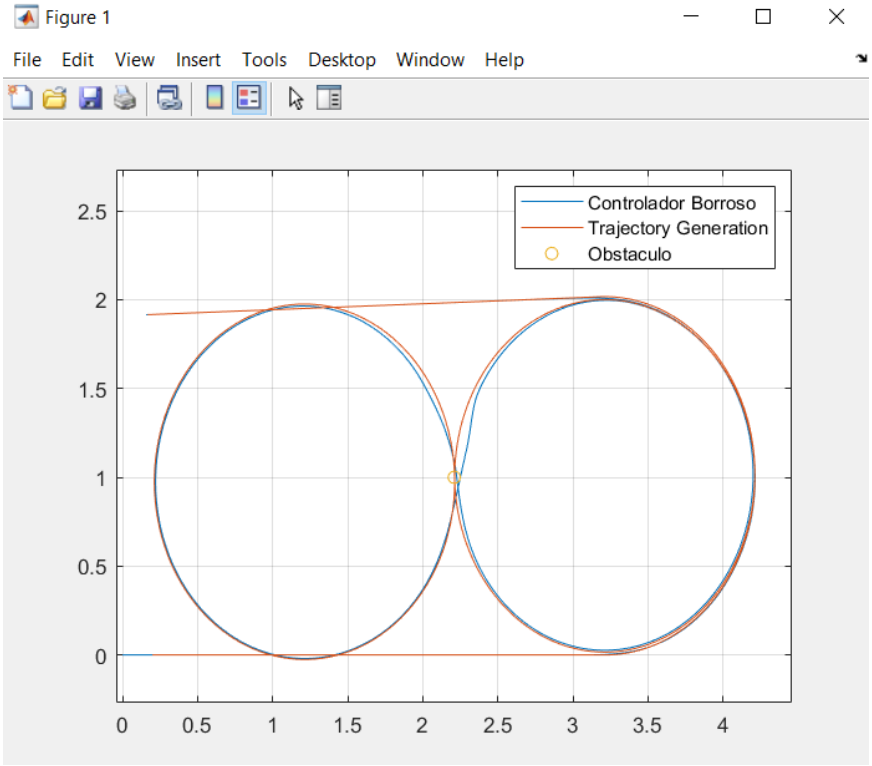

Figura 17: Resultado de esquivar el obstáculo $O_2(2.2, 1)$.

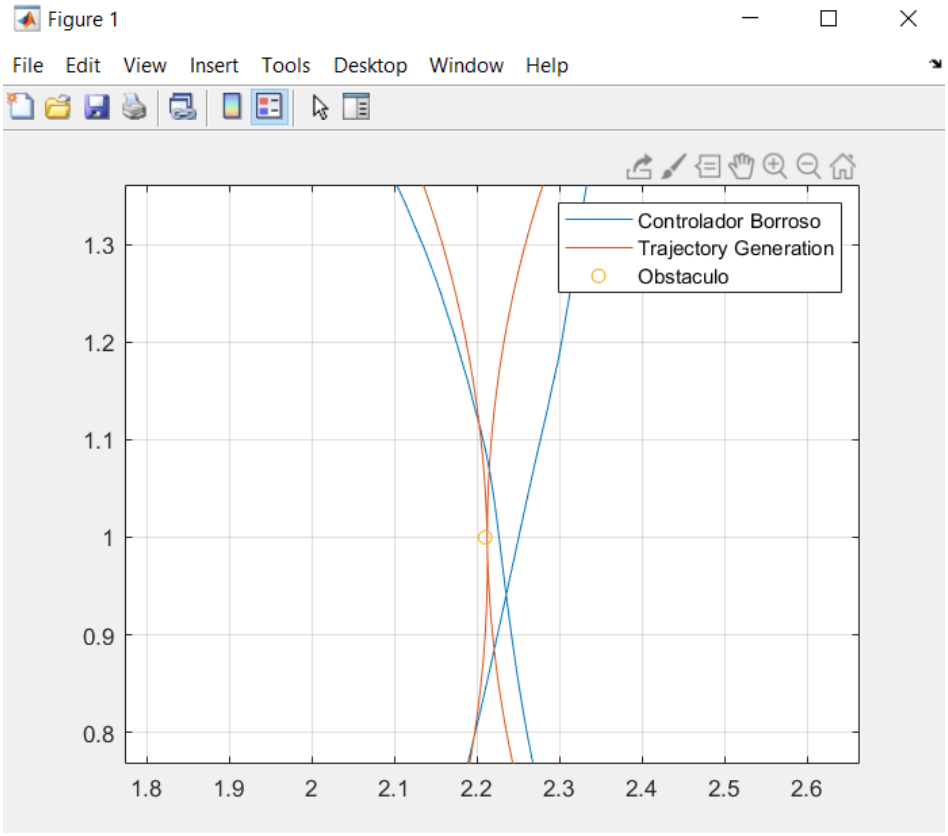

Figura 18: Resultado ampliado de esquivar el obstáculo $O_2(2.2, 1)$.

Una vez más el Controlador Borroso ha sido capaz de seguir la trayectoria a la perfección, desviándose ligeramente alrededor del obstáculo.

Como conclusión hemos conseguido desarrollar un **Controlador Borroso** capaz de desplazar un Robot hasta una referencia, evitando cualquier obstáculo que encuentre en su camino hacia esta.