PRÁCTICA 2: Control Borroso

Sistemas de Control Inteligente

Alba Calvo Herrero
María Sanz Espeja

Grupo 11 - A3 lunes
Grado Ingeniería Informática

Índice

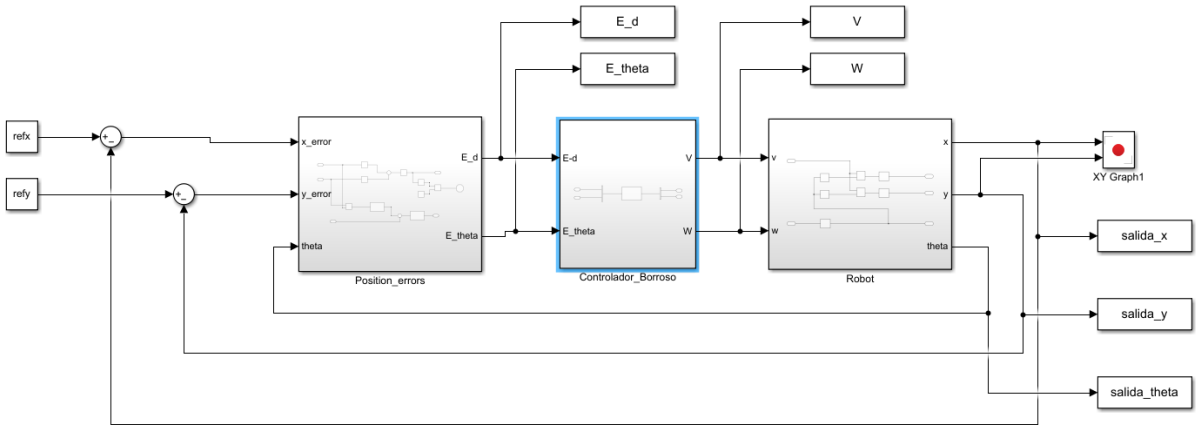
Introducción

Esta memoria explica la implementación y resultados de dos sistemas de control borroso desarrollados en MATLAB mediante Simulink para un robot móvil. La primera parte se centra en el control de posición del robot, mientras que la segunda añade una capa de complejidad con la evitación de obstáculos. Las simulaciones demuestran la aplicabilidad y eficacia del control borroso en escenarios de robótica.

Parte I. Diseño de un Control Borroso de Posición para un Robot Móvil

1. Objetivo y descripción del sistema.

En esta primera sección de la práctica, se nos solicita construir el circuito en Simulink siguiendo las instrucciones proporcionadas en el enunciado. La descripción general del sistema creado es la siguiente:



El modelo se ha guardado como *simulationControl.slx*

2. Desarrollo de la práctica

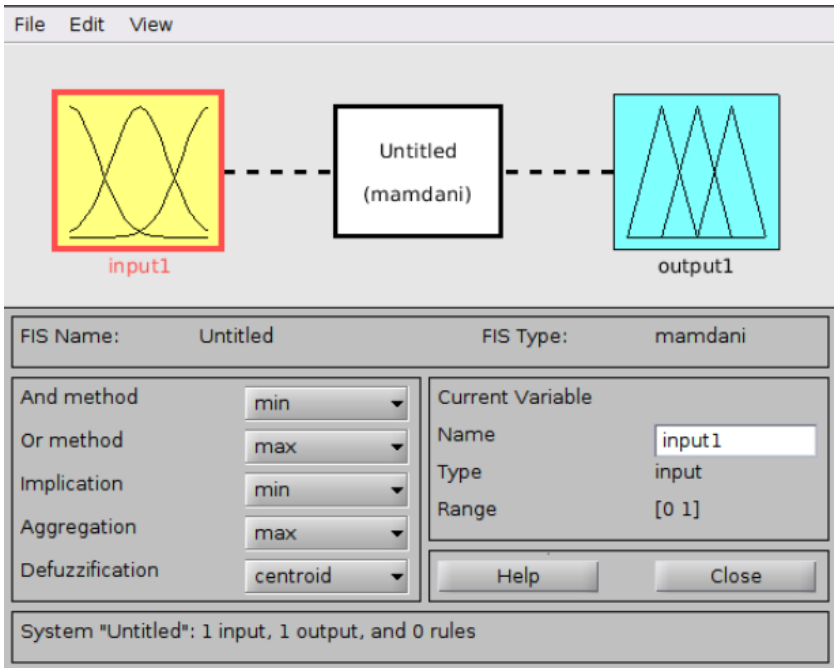
a) Ejecute el comando "fuzzy" en la consola de Matlab para crear un controlador borroso mediante la toolbox de Matlab de control borroso. Se abrirá la interfaz que se muestra en la Figura 4.
Mantenga las opciones por defecto para las operaciones de AND/OR y el desborrosificador (Defuzzification) y siga los siguientes pasos:

i. Cree un sistema borroso con dos entradas y dos salidas mediante el menú "Edit/Add Variable/input" y "Edit/Add Variable/output". Nombre las entradas como "E_d" y "E_theta" y las salidas como "V" y "W" tal y como se muestra en la Figura 5.

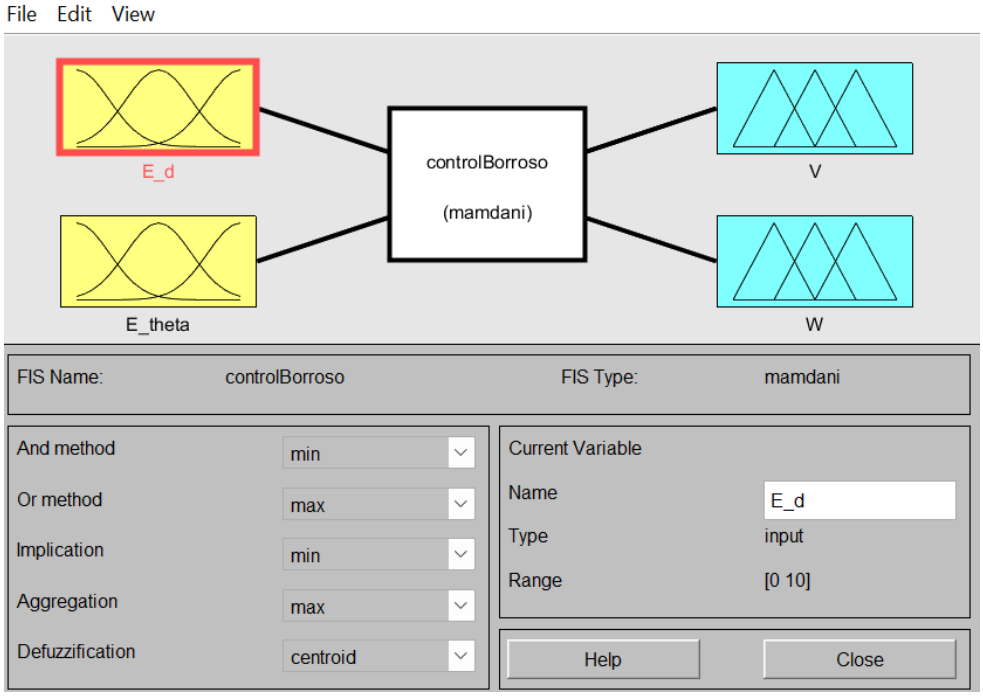
En la línea de comandos de matlab, se ejecuta el siguiente comando para crear el sistema borroso:

```
> fuzzy
```

Nos aparece la siguiente pantalla donde se pueden añadir variables y editarlas.



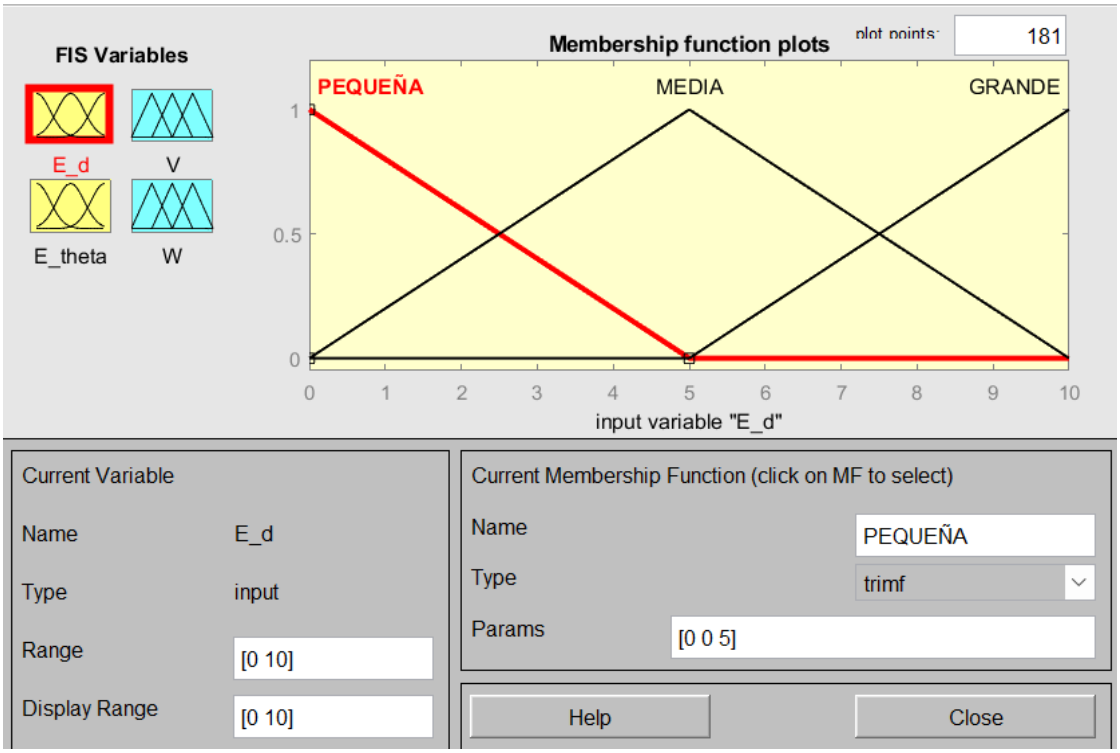
Creamos las entradas y las salidas:



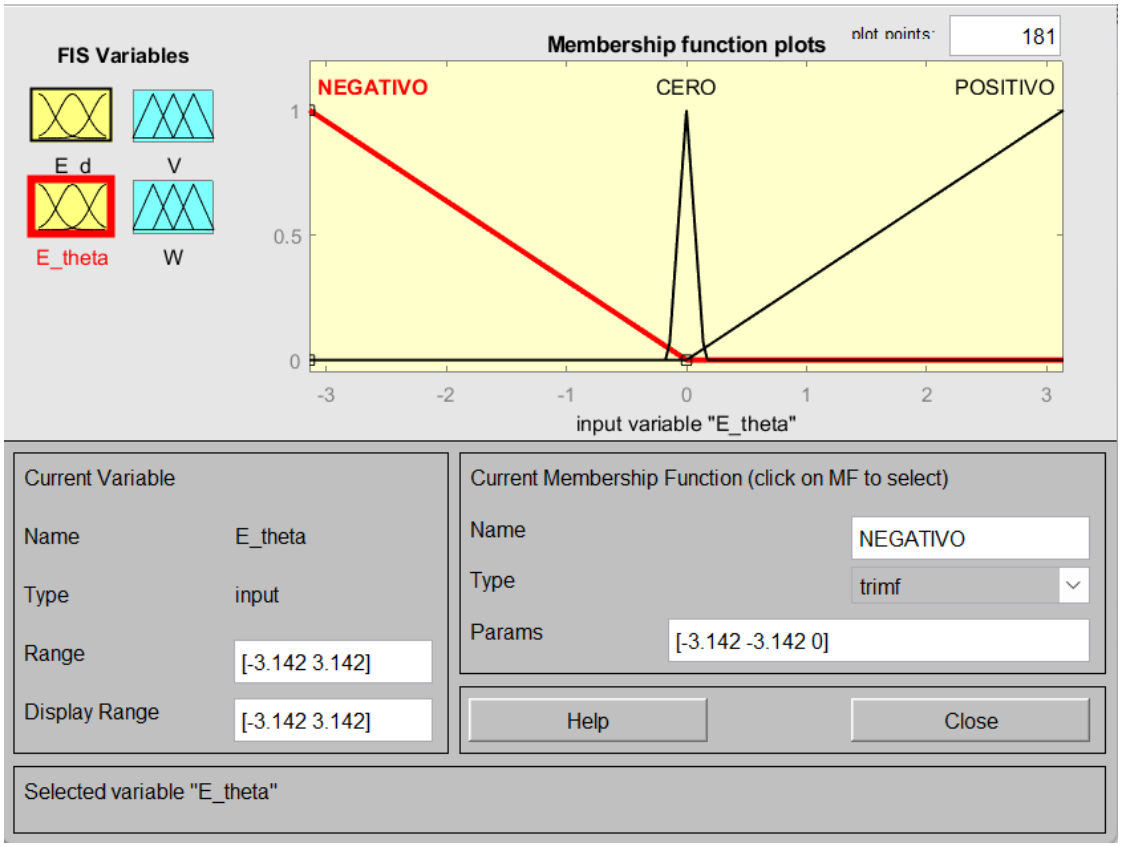
ii. Ajuste las variables de entrada para que dispongan de los siguientes rangos (véase Figura 6): Ed ∈ [0,15], Eo + ∈ [π–, π], V ∈ [0, 2], W ∈ [−1,1]

Se editan las variables de entrada y de salida pulsando encima de cada una de ellas. Los parámetros introducidos son los siguientes.

- **Entrada 1 ("E_d"):**
  - Rango de valores: [0, 10]
  - Número de funciones de membresía (MFs): 3
  - Funciones de membresía:
  - "PEQUEÑA" (trimf): [0, 0, 5]
  - "MEDIA" (trimf): [0, 5, 10]
  - "GRANDE" (trimf): [5, 10, 10]

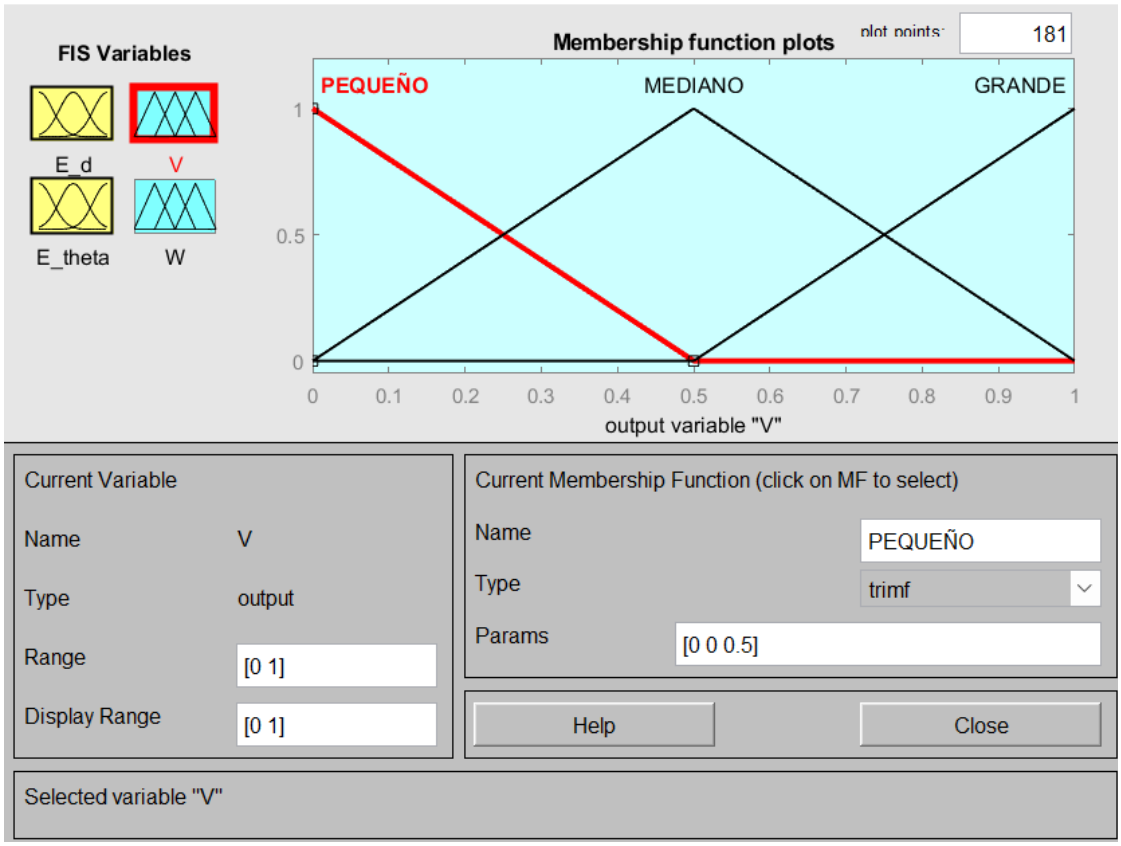


- **Entrada 2 ("E_theta"):**
  Se han tenido que cambiar los valores de pi. Al principio, se puso 3.14, sin embargo, muchas trayectorias no eran correctas, así que, finalmente se cambió a un valor más cercano de pi como se puede ver a continuación.
  - Rango de valores: [-3.14159265358979, 3.14159265358979]
  - Número de funciones de membresía (MFs): 3
  - Funciones de membresía:
  - "NEGATIVO" (trimf): [-3.142, -3.142, 0]
  - "POSITIVO" (trimf): [0, 3.142, 3.142]
  - "CERO" (trimf): [-0.15, 0, 0.15]

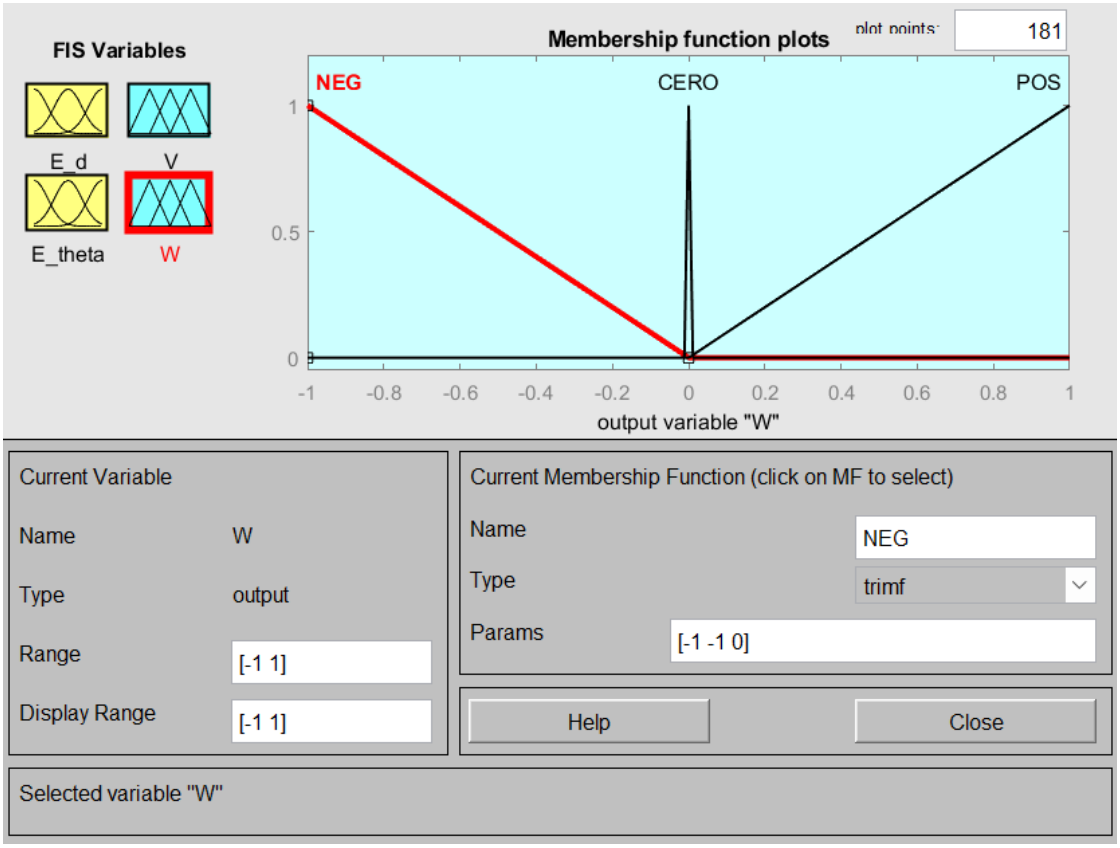


- **Salida 1 ("V"):**
  - Rango de valores: [0, 1]
  - Número de funciones de membresía (MFs): 3
  - Funciones de membresía:
  - "PEQUEÑO" (trimf): [0, 0, 0.5]
  - "MEDIANO" (trimf): [0, 0.5, 1]
  - "GRANDE" (trimf): [0.5, 1, 1]

Hemos cambiado el rango de la velocidad lineal de [0,2] a [0,1] porque nos daba error en las gráficas de la trayectoria, ya que, la velocidad lineal era demasiado rápida para seguir correctamente la trayectoria al destino.



- **Salida 2 ("W")**:
    - Rango de valores: [-1, 1]
    - Número de funciones de membresía (MFs): 3
    - Funciones de membresía:
    - "NEG" (trimf): [-1, -1, 0]
    - "CERO" (trimf): [0, 0, 0]
    - "POS" (trimf): [0, 1, 2]

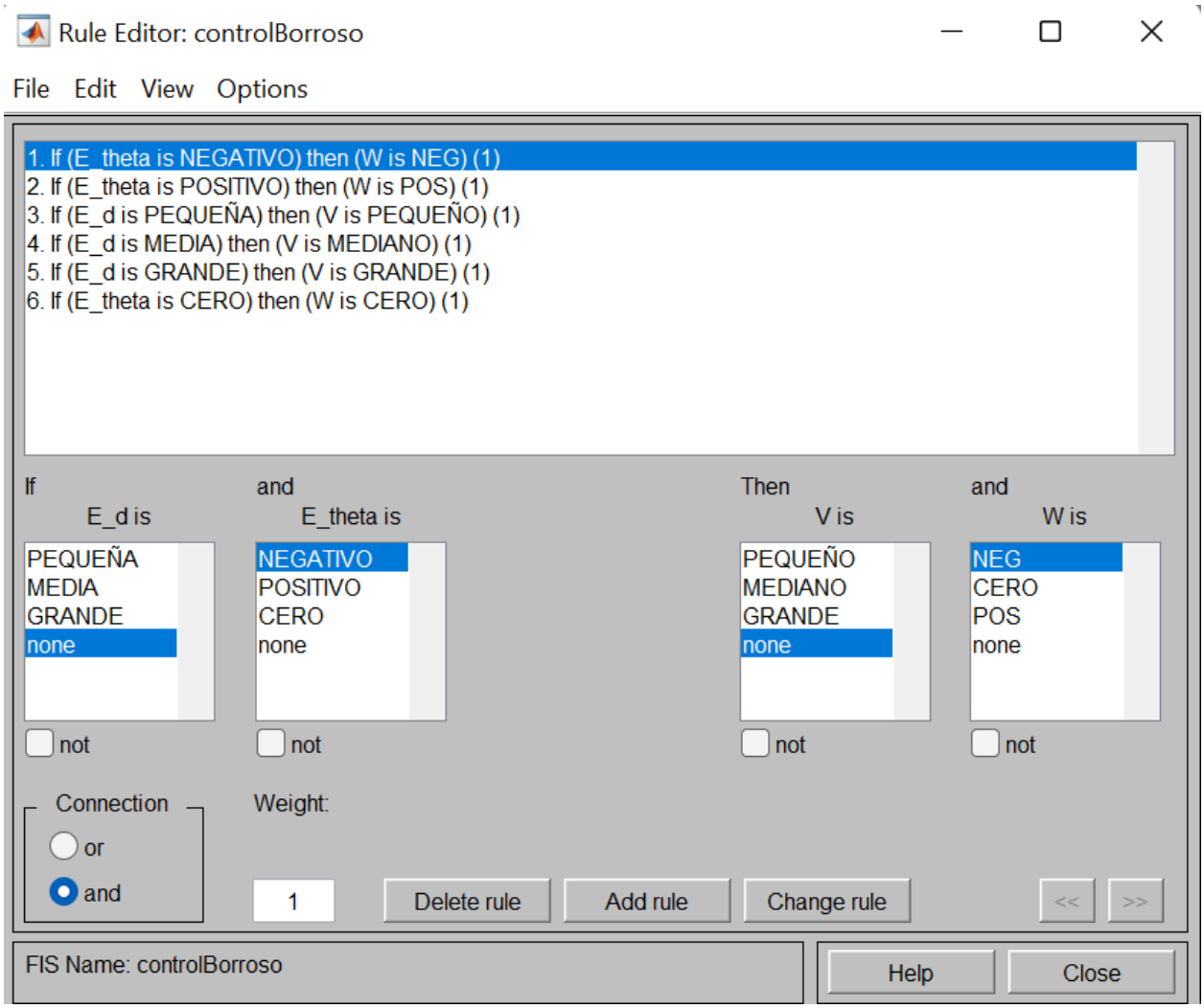


iii. Cree las funciones de pertenencia del tipo que considere necesarias en cada una de las variables definidas en la entrada y la salida (véase Figura 7). Utilice nombres para los conjuntos borrosos que tengan un significado claro como NEGATIVO, POSITIVO, GRANDE, PEQUEÑO, etc.

(Realizado en el paso anterior).

iv. Diseñe la tabla de reglas del controlador mediante la interfaz de diseño de reglas mostrada.

Dentro de la interfaz de rule editor, se han creado 6 reglas usando el conector AND:



La implementación del controlador borroso tiene como finalidad dirigir un robot móvil desde un punto de partida (-4, -4) hacia un destino generado aleatoriamente. Este controlador utiliza dos variables de entrada: el error de distancia (E_d) y el error angular (E_theta), y dos variables de salida: la velocidad lineal (V) y la velocidad angular (W). Las seis reglas del sistema borroso se han diseñado con el propósito de proporcionar una estrategia de control que permita al robot autocorregirse y alcanzar el punto destino de forma eficiente.

- Regla 1: **Si (E_theta es NEGATIVO) entonces (W es NEG)**. Esta regla se aplica cuando el robot necesita corregir su orientación girando hacia la izquierda, y por tanto, la velocidad angular W se establece en un valor negativo.
- Regla 2: **Si (E_theta es POSITIVO) entonces (W es POS)**. Contraria a la regla 1, esta regla se activa cuando el robot debe girar hacia la derecha, asignando un valor positivo a la velocidad angular W.
- Regla 3: **Si (E_d es PEQUEÑA) entonces (V es PEQUEÑO)**. Esta regla ajusta la velocidad lineal V a un valor bajo cuando el robot se encuentra cerca del destino, permitiendo una aproximación controlada y evitando sobreoscilaciones.

- Regla 4: **Si (E_d es MEDIA) entonces (V es MEDIANO)**. En el caso de que el robot esté a una distancia intermedia del objetivo, la velocidad lineal V se configura a un valor medio para avanzar de manera efectiva sin acercarse demasiado rápido.
- Regla 5: **Si (E_d es GRANDE) entonces (V es GRANDE)**. Si la distancia al destino es considerable, esta regla incrementa la velocidad lineal V para que el robot cubra la distancia de manera más rápida.
- Regla 6: **Si (E_theta es CERO) entonces (W es CERO)**. Esta regla se activa cuando el robot está correctamente orientado hacia el destino, y en consecuencia, no es necesario aplicar una velocidad angular W, manteniendo al robot en su trayectoria lineal hacia el objetivo.

La interacción de estas reglas permite que el robot ajuste de forma dinámica su trayectoria y velocidad en respuesta a los cambios en su posición y orientación relativas al destino.

v. Exporte el controlador diseñado a un fichero .fis mediante el menú "File/Export to File"

b) Una vez diseñado el controlador, se creará el esquema mostrado en la Figura 9 en el entorno Simulink para generar el bloque controlador mediante la opción "Create Subsystem from Selection" (véase Figura 10).

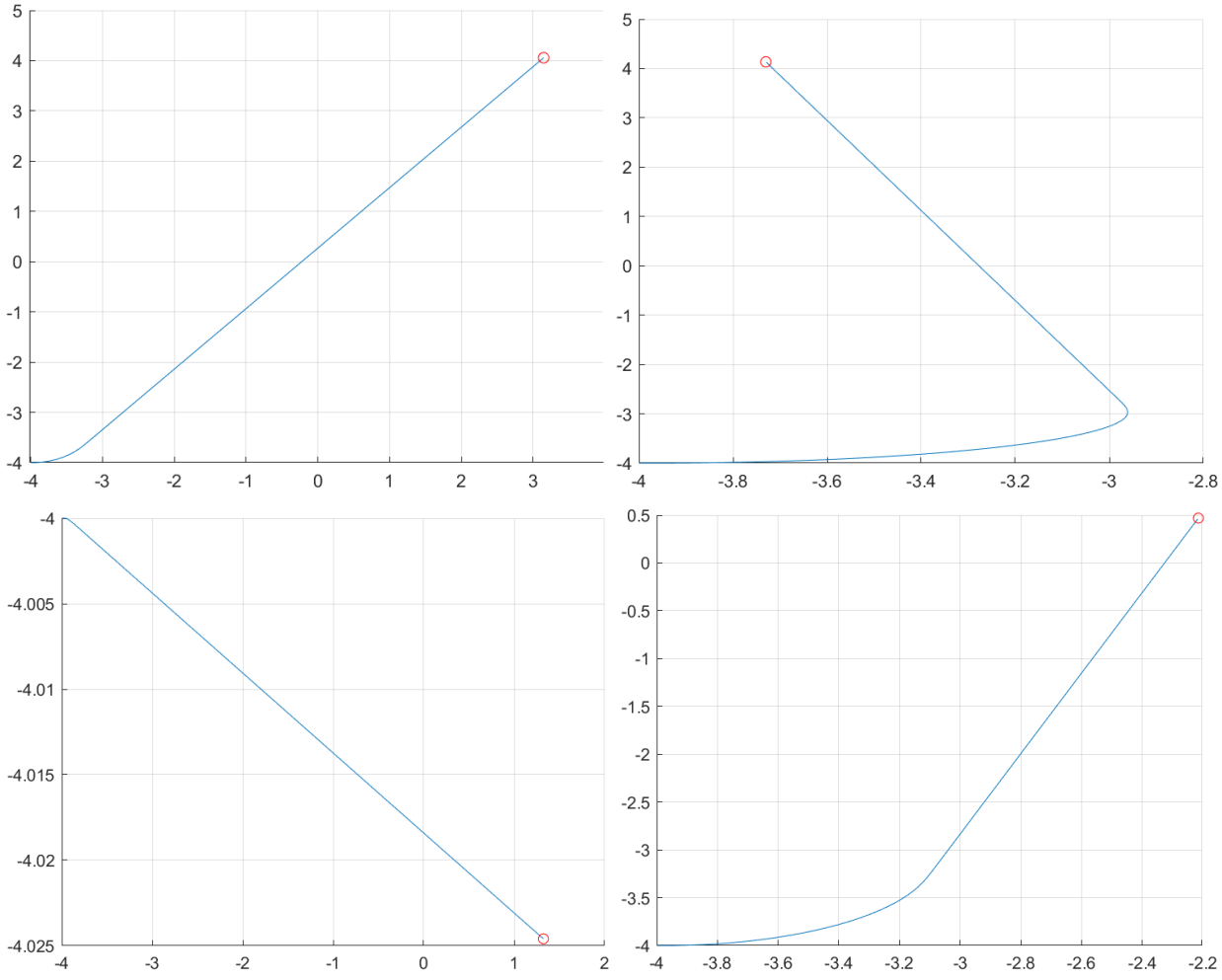
c) Introduzca el bloque controlador en el esquema de Simulink de la Figura 1 y guarde el modelo con el nombre "PositionControl.slx". Realice pruebas con posiciones refx y refy aleatorias para comprobar que el controlador se comporta adecuadamente. Para ello genere un script de Matlab que permita automatizar ese proceso y mostrar mediante la función plot la trayectoria seguida por el robot. Ejecute dicho script varias veces o introduzca un bucle en el código proporcionado.

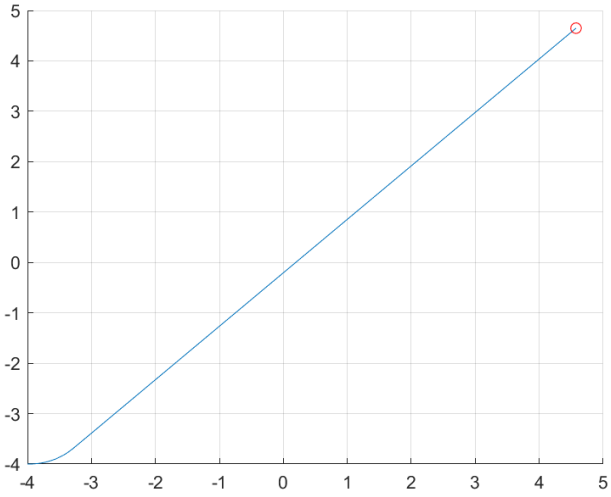
Se crea el script de Matlab, llamado *Trayectoria.m* que realiza el número de pruebas que se quieran hacer con posiciones destino aleatorias en cada iteración. El destino será mostrado en las gráficas mediante un círculo rojo.

```
% Tiempo de muestreo
Ts = 100e-3;
% Número de pruebas que deseas realizar
num_pruebas = 5;
for i = 1:num_pruebas
   % Generar referencia x-y de posición aleatoria
   refx = 10 * rand - 5;
   refy = 10 * rand - 5;
   % Ejecutar Simulación
   sim('simulationControl.slx');

   % Obtener los datos de salida de la simulación
   x = salida_x.signals.values;
   y = salida_y.signals.values;

   % Mostrar la trayectoria seguida por el robot
```

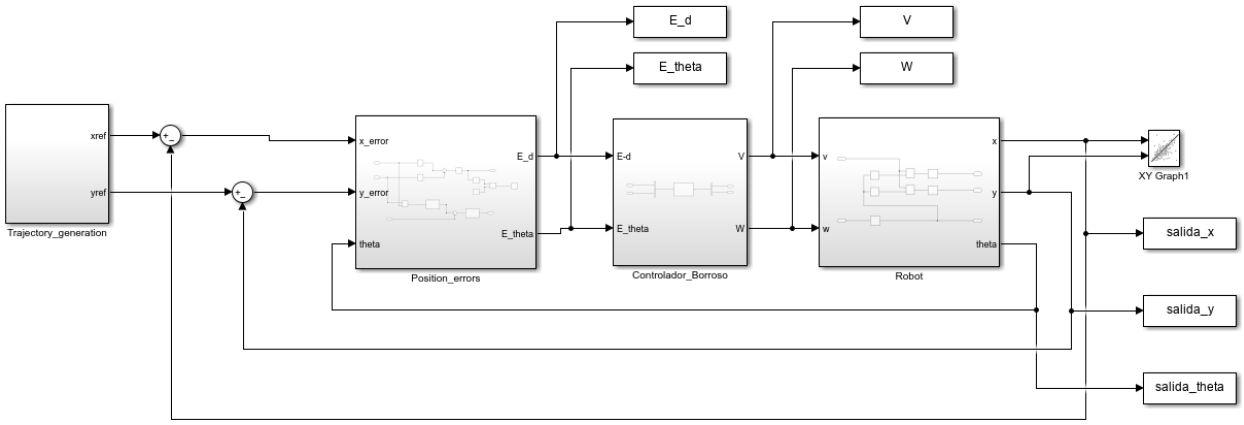
```
    figure;
    hold on;
    plot(x, y);
    plot(refx,refy,'ro');
    grid on;


    % Esperar antes de la siguiente prueba
    pause(1);
end
```

Se comprueba cómo el robot empieza el movimiento desde la posición (-4,-4) y llega hasta (refx,refy) en las cinco iteraciones, teniendo en ocasiones que girar y cambiar su trayectoria.

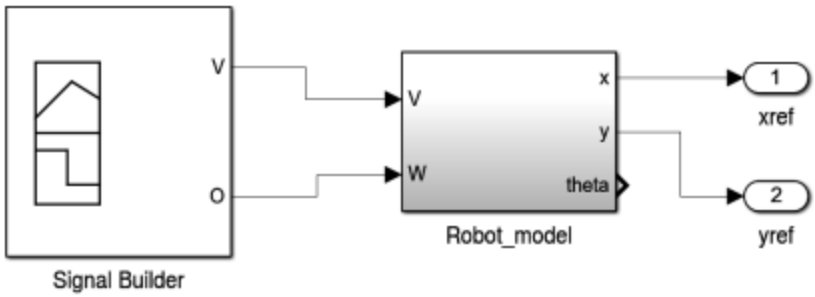




d) Sustituya las referencias de posición (bloques refx y refy) por el generador de trayectorias proporcionado ("Trajectory_generator.slx") y compruebe el funcionamiento del sistema que se muestra en la Figura 11. Realice los cambios en el diseño del controlador borroso que sean necesarios para que el robot sea capaz de seguir la trayectoria.

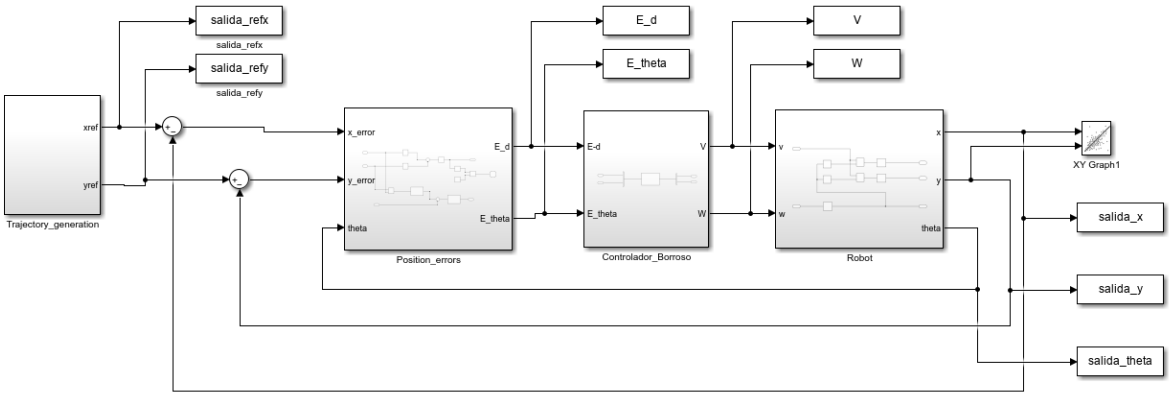


La Figura 13 muestra la arquitectura del subsistema "Trajectory_generation", que generará la trayectoria que debe seguir el robot móvil. Este subsistema se proporciona dentro de la documentación de la práctica en el fichero "Trajectory_generator.slx". Para utilizar este bloque es necesario inicializar los parámetros iniciales de la trayectoria (x_0, y_0, th_0) y el periodo de muestreo (Ts). Debe tener en cuenta que, si la trayectoria y el robot comienzan lo suficientemente cerca como para que se cumpla la condición de parada, se terminará la simulación.



El bloque "Signal Builder" se utiliza para generar las señales de velocidad lineal y angular. El bloque "Robot_model" es idéntico al utilizado para modelar el robot, añadiendo variables a los integradores para cambiar la posición (variables x0 e y0) y orientación (theta0) iniciales de la trayectoria. Este bloque y la trayectoria generada se muestran a continuación.

Se han añadido las salidas salida_refx y salida_refy de la siguiente forma:

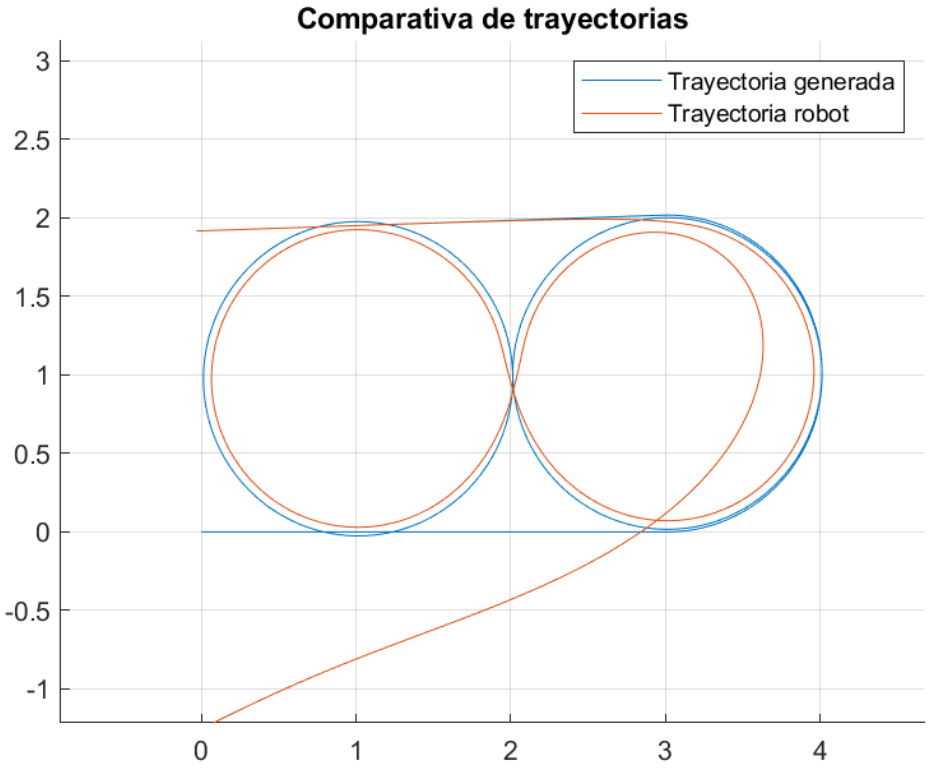


Se ejecuta el script ejercicio2.m. Este script muestra las trayectorias de la trayectoria generada y la trayectoria del robot en una gráfica de manera que se puedan comparar. Se inicializan los valores de x_0, y_0 y th_0 a 0. Así no se cumplirá la condición de parada, ya que, está alejado de la posición de salida.

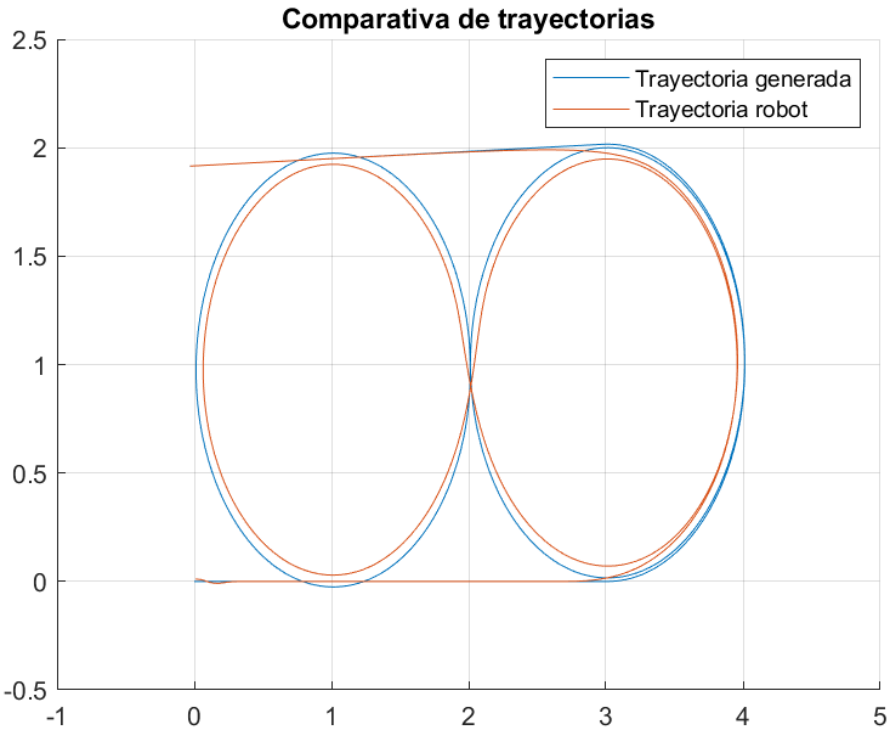
```
clc;
% Tiempo de muestreo
Ts = 100e-3;
x_0 = 0;
y_0 = 0;
th_0 = 0.0;
% Ejecutar Simulación
sim('trajectoryControl.slx');
% Guardamos datos de trayectoria de simulink
trayectoria_x = salida_refx.signals.values';
trayectoria_y = salida_refy.signals.values';
```

```
% Obtener los datos de salida de la simulación
x = salida_x.signals.values;
y = salida_y.signals.values;
% Mostrar la trayectoria seguida por el robot
figure(1);
hold on;
tray = plot(trayectoria_x,trayectoria_y);
tray_robot = plot(x, y);
hold off;
grid on;
legend([tray tray_robot], {'Trayectoria generada', 'Trayectoria robot'});
title('Comparativa de trayectorias')
```

Se observa que las trayectorias son muy similares:
En el primer caso, la posición inicial del robot es (-4,-4) y se observa que no es igual a la que imita pero sí que acaba en el mismo punto. Esto es debido al punto de inicio, ya que nuestro robot coge como referencia el punto final y cuanto más alejado esté del punto inicial será menos similar a la trayectoria de las gafas.



Si cambiamos la posición inicial del robot a **(0.01,0.01)** en vez de (-4,-4) se observa que sigue la trayectoria a la perfección, ya que, el punto inicial es más cercano a la trayectoria que quiere imitar.


Comparativa de trayectorias
Trayectoria generada
Trayectoria robot
2.5
2
1.5
1
0.5
0
-0.5
-1
0
1
2
3
4
5

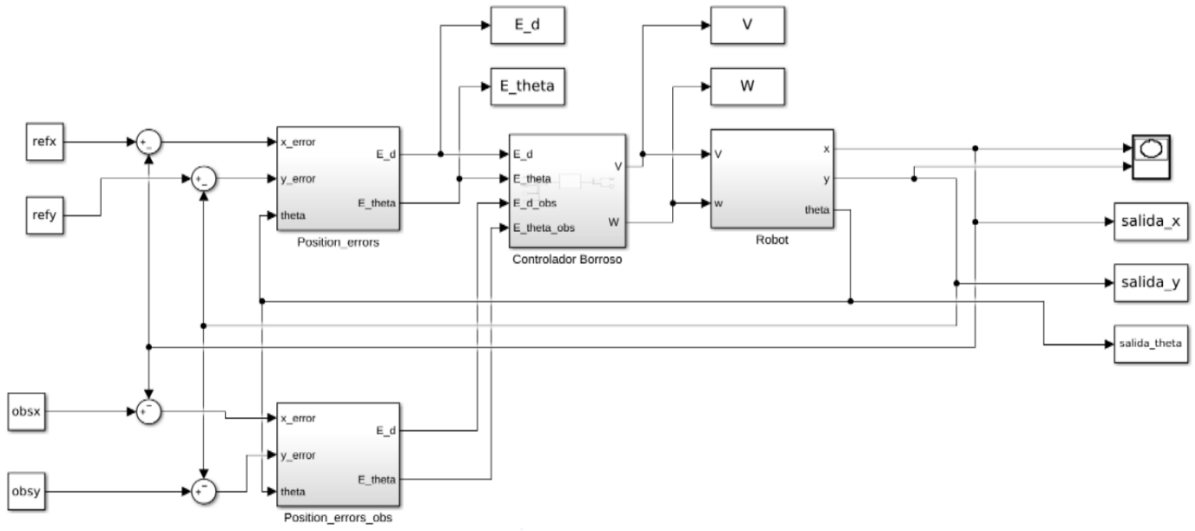
Parte 2. Diseño de control borroso de posición con evitación de obstáculos

1. Objetivo y descripción del sistema

En esta segunda parte de la práctica de Control Borroso, se plantea extender el diseño del control de posición de un robot móvil realizado anteriormente para que incluya la evasión de un obstáculo de posición y dimensiones conocidas dentro del entorno utilizado en la primera parte de la práctica.

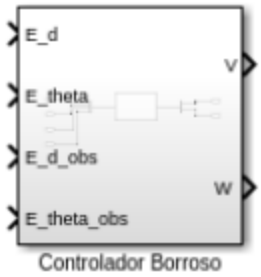
El objetivo de esta parte de la práctica es diseñar un controlador borroso de manera que el robot sea capaz de alcanzar una posición determinada por las referencias de posición refx y refy al mismo tiempo que se evita colisionar con un obstáculo que se encuentra en la posición obsx y obsy.

El esquema de control propuesto se describe mediante el diagrama de bloques de la Figura 2 en el entorno Matlab/Simulink:



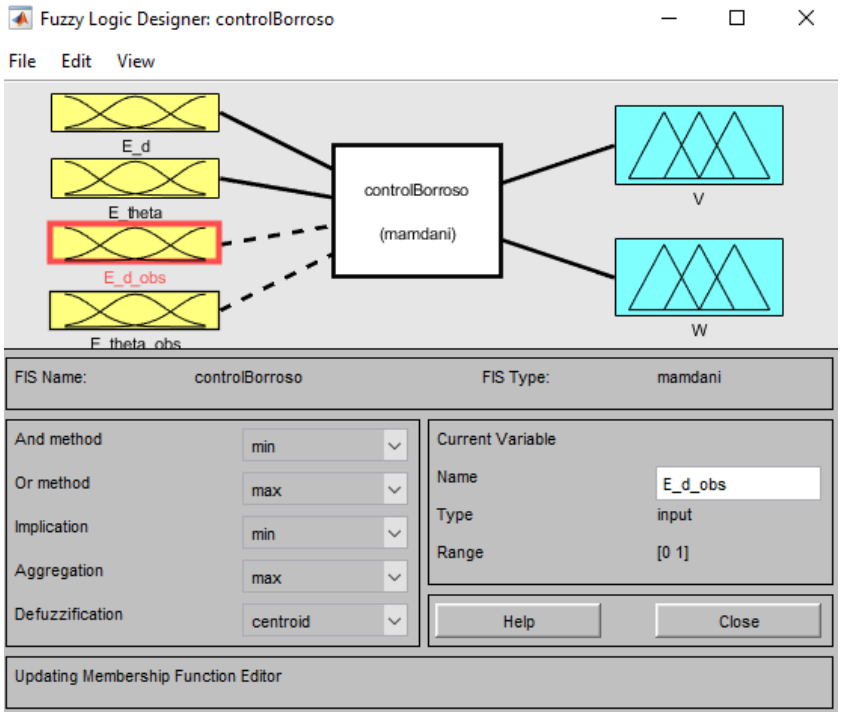
2. Desarrollo de la práctica

El controlador realizado tendrá como entradas los errores de distancia y ángulo a la referencia de posición (Ed; Etheta) y los errores de distancia y ángulo a la posición del obstáculo (Ed,obs; Etheta,obs). El controlador genera a su salida los valores de velocidad (angular y lineal) que serán entradas en el bloque de simulación del robot. El controlador borroso diseñado se implementará en un bloque de Simulink como el mostrado en la Figura.



Para ello se piden los siguientes apartados:

a) Defina un controlador de 4 entradas y 2 salidas como el mostrado a continuación. Se recomienda partir del controlador diseñado en la primera parte de la práctica (ver Figura 4).



b) Diseñe las funciones de pertenencia de las nuevas entradas "E_d_obs" y "E_theta_obs" y modifique las que crea convenientes del resto de entradas y salidas del controlador.

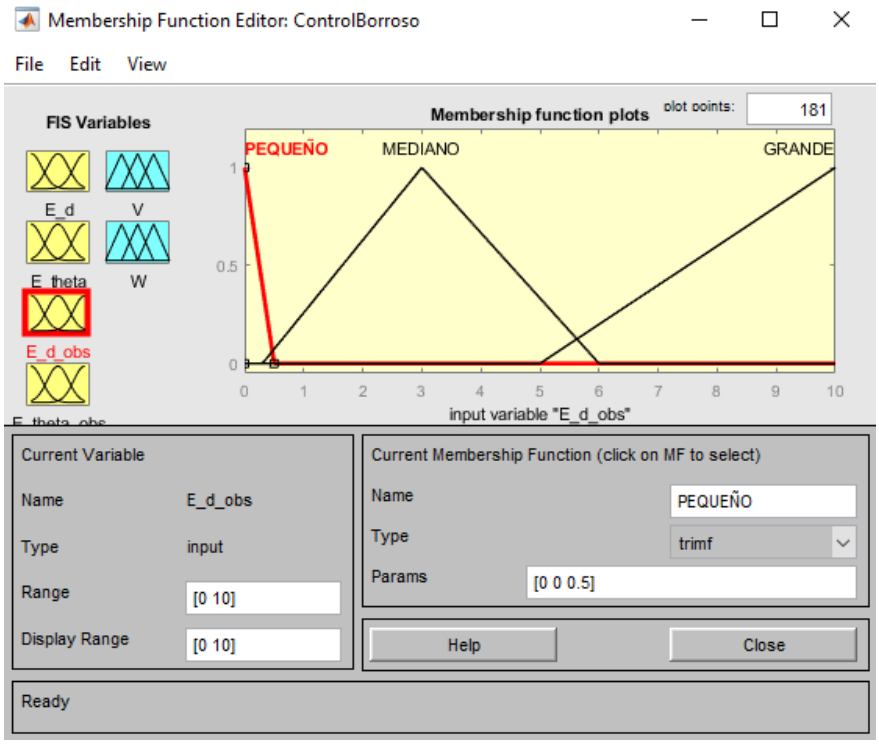
A continuación, se explican los valores introducidos para las nuevas entradas:

- Entrada **E_d_obs** (Error de distancia a obstáculos):

  Para la evasión de obstáculos, se considera la distancia a estos. Se establecen tres categorías para la función de membresía que guiarán la respuesta del robot ante posibles colisiones.

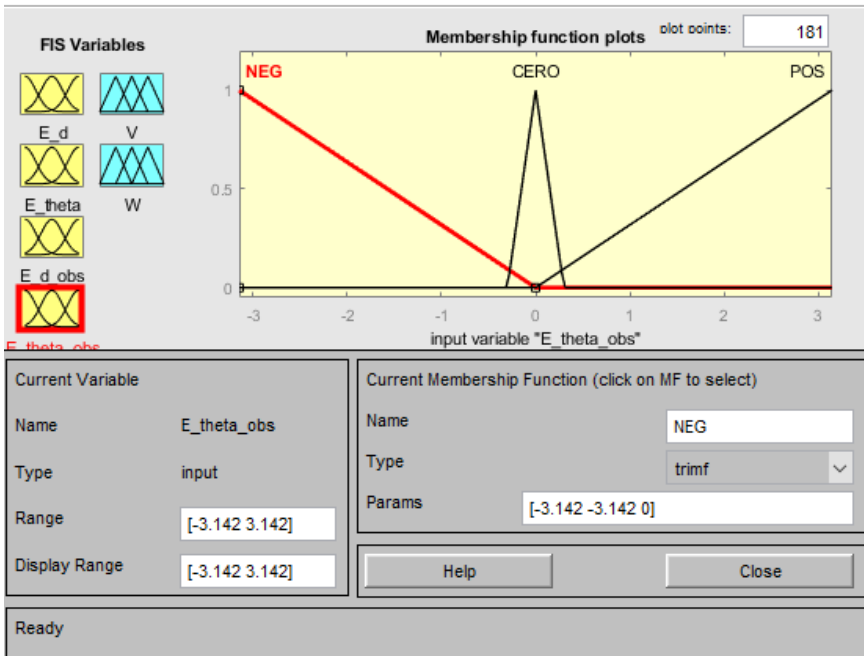
  - Rango de valores: [0, 10]

- Número de funciones de membresía (MFs): 3
- Funciones de membresía:
- "PEQUEÑO" (trimf): [0, 0, 0.5]: Con la membresía "PEQUEÑO", se indica una proximidad inmediata a un obstáculo, requiriendo una reacción rápida y posiblemente un detenimiento completo para evitar impactos.
- "MEDIANO" (trimf): [0.3, 3, 6]: prepara al robot para maniobrar con cautela alrededor de obstáculos que están a una distancia moderada, ajustando su trayectoria de forma proactiva.
- "GRANDE" (trimf): [5, 10, 10]: Para distancias "GRANDE", se interpreta que el obstáculo está lejos o fuera del camino inmediato, permitiendo que el robot mantenga su velocidad y trayectoria actual sin cambios drásticos.



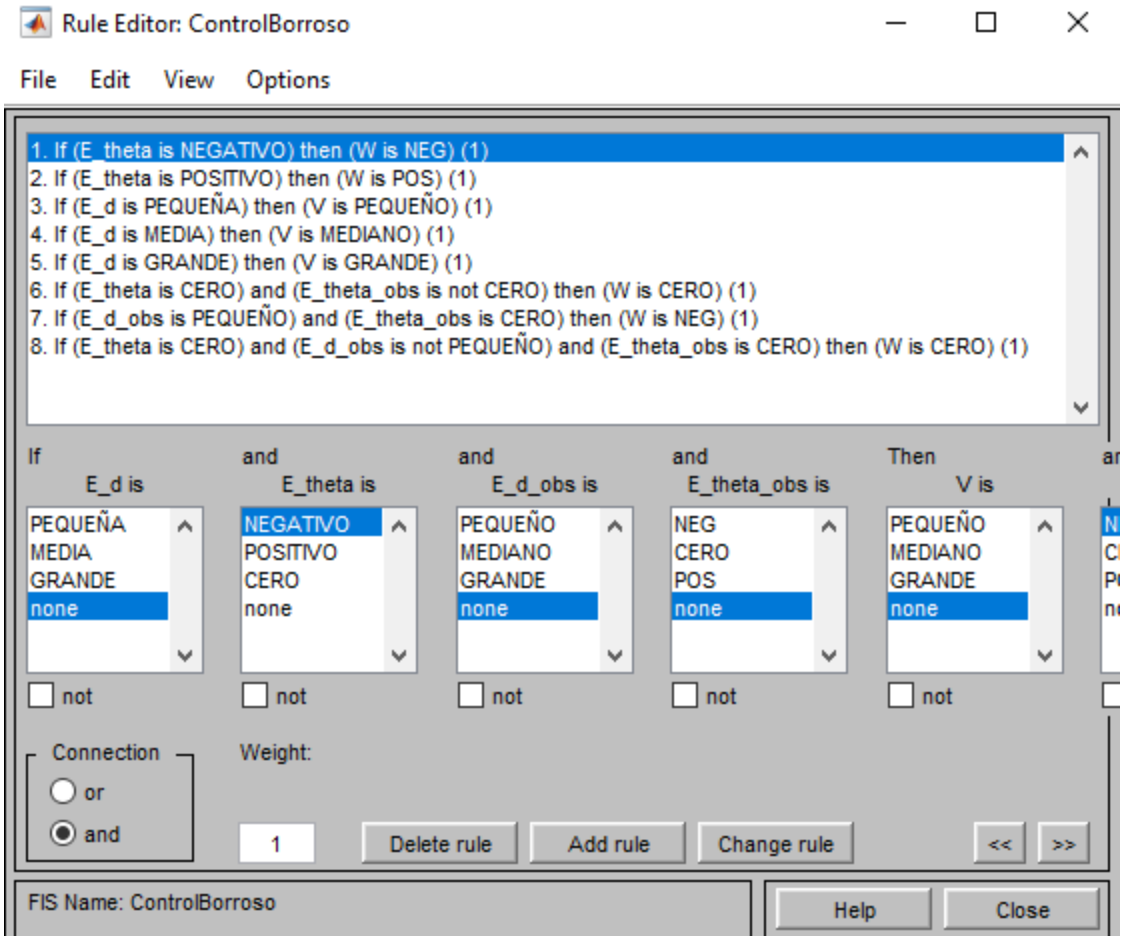
- Entrada **E_theta_obs** (Error angular respecto a obstáculos):
  - La orientación relativa a los obstáculos es crucial para la maniobra y evasión eficiente en la navegación del robot. Se utilizan valores precisos de pi para asegurar la correcta interpretación de la dirección angular en relación con los obstáculos detectados.
  - Rango de valores: [-3.14159265358979, 3.14159265358979]
  - Número de funciones de membresía (MFs): 3
  - Funciones de membresía:

- NEG: La función [-3.142 -3.142 0]: La función "NEG" se aplica cuando el obstáculo se detecta en un ángulo negativo respecto a la orientación actual del robot, indicando la necesidad de girar hacia la derecha para evitarlo.
- CERO: Con [-0.3 0 0.3]: La función "CERO" se usa cuando el obstáculo está casi directamente en la trayectoria del robot, implicando que se requiere un ajuste muy fino o ninguna acción para mantener el rumbo.
- POS: La función [0 3.142 3.142]: La función "POS" entra en juego cuando el obstáculo se encuentra a un ángulo positivo, lo que significa que el robot debe girar hacia la izquierda para esquivar el obstáculo y continuar hacia su destino.



c) Cree la tabla de reglas del nuevo controlador. Se puede utilizar la opción "none" para ignorar algunas de las entradas del controlador en una regla (ver Figura 5).

Introducimos las reglas en la interfaz:



A continuación, se explican las nuevas reglas (6,7,8):

- Regla 6: **Si (E_theta es CERO) y (E_theta_obs es NEG) entonces (W es MEDIANO)**. Esta regla ajusta la velocidad angular a un valor medio para realizar un giro moderado hacia la derecha, posiblemente para esquivar un obstáculo.

- Regla 7: **Si (E_d_obs es PEQUEÑO) y (E_theta_obs es POS) entonces (V es PEQUEÑO)**. Se emplea cuando hay un obstáculo muy cercano y en el camino directo del robot, requiriendo una reducción de la velocidad para maniobrar con cuidado.

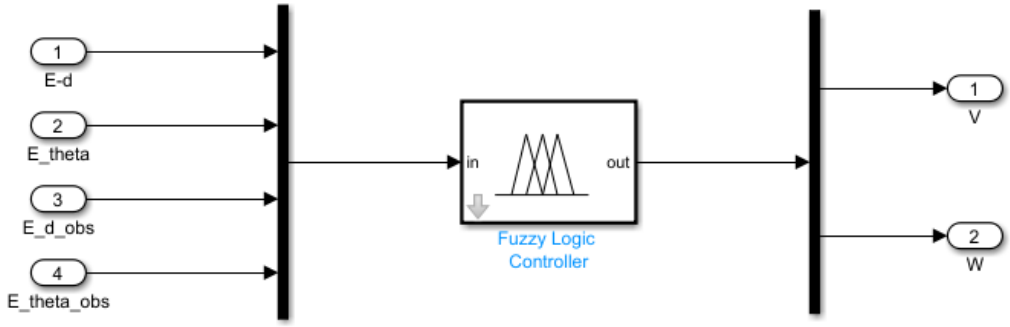
- Regla 8: **Si (E_theta es CERO) y (E_theta_obs es POS) entonces (W es MEDIANO)**. Esta regla se aplica cuando el robot está bien orientado hacia el objetivo pero necesita realizar un ajuste moderado hacia la izquierda para evitar un obstáculo.

Estas reglas están diseñadas para que el robot pueda adaptar de manera inteligente tanto su velocidad como su curso basándose en lo que percibe en tiempo real acerca de su ubicación y cualquier obstáculo cercano. Esto ayuda a asegurar que el robot se mueva de forma segura y eficaz, alcanzando su destino sin contratiempos.

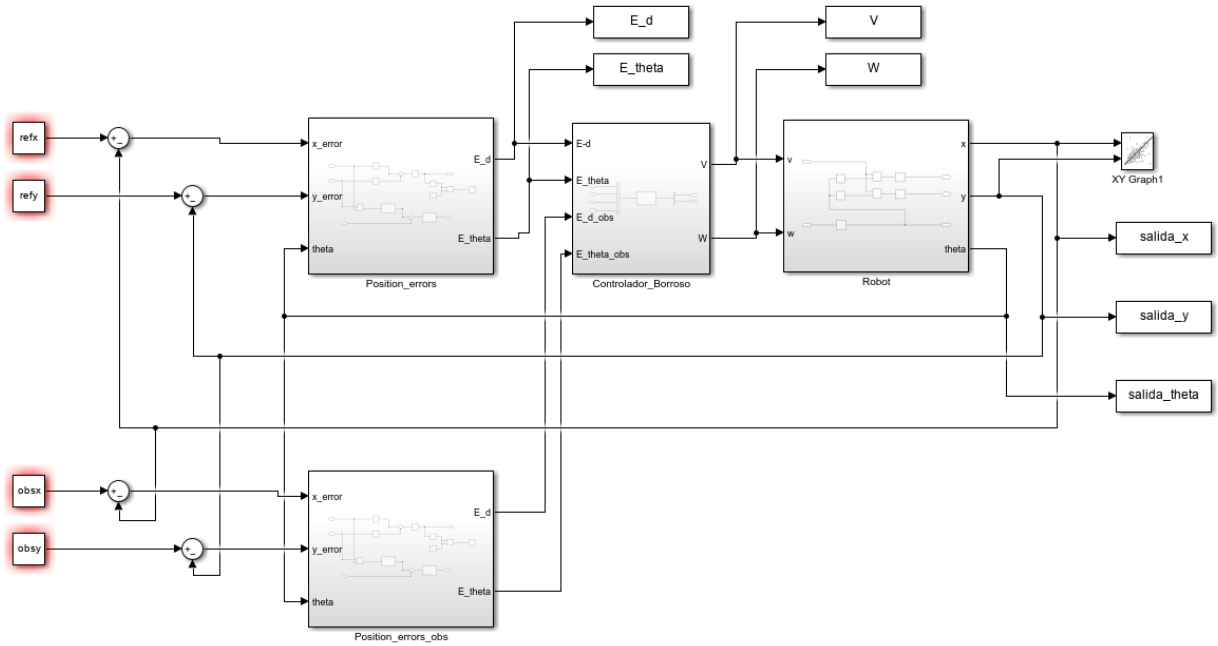
d) Evalúe el controlador diseñado utilizando el diagrama de bloques de la Figura 2. Llame a ese diagrama 'EvitarObstaculo.slx' y proponga valores de las variables refx, refy obsx y obsy en los que el robot esté obligado a esquivar el obstáculo y muestre las trayectorias seguidas por el robot.

Se crea el nuevo controlador borroso con las nuevas entradas en el multiplexor.

M



Modificamos la estructura de SimulationControl.slx con el nuevo sistema propuesto:

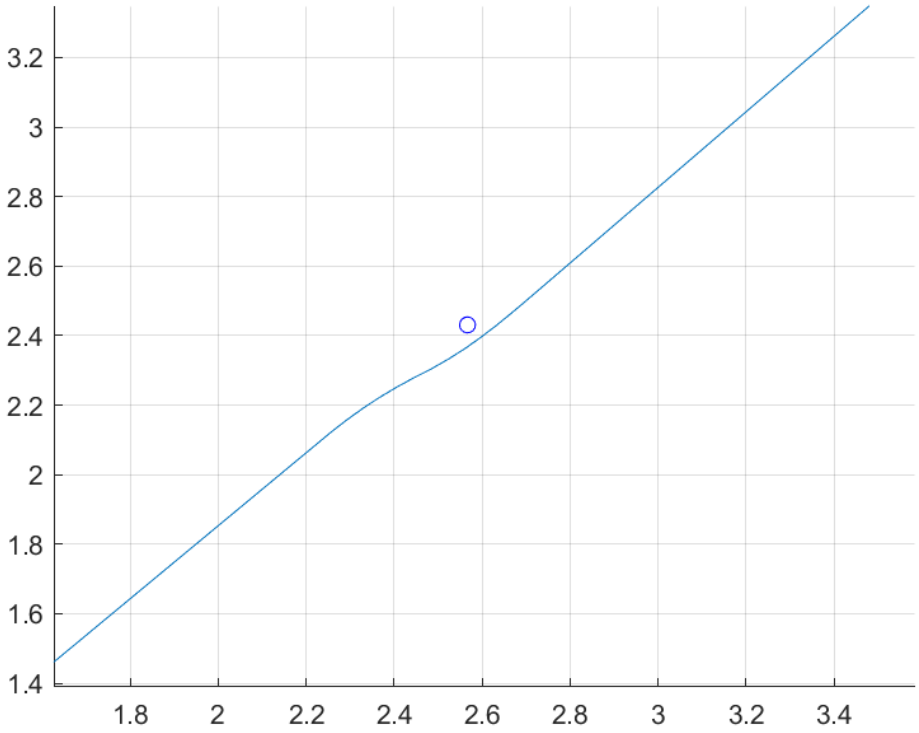


Generamos el script Trayectoria.m:

```
% Tiempo de muestreo
Ts = 100e-3;
% Generar referencia x-y de posición aleatoria
refx = 5;
refy = 5;
obsx = 2.56689;
```

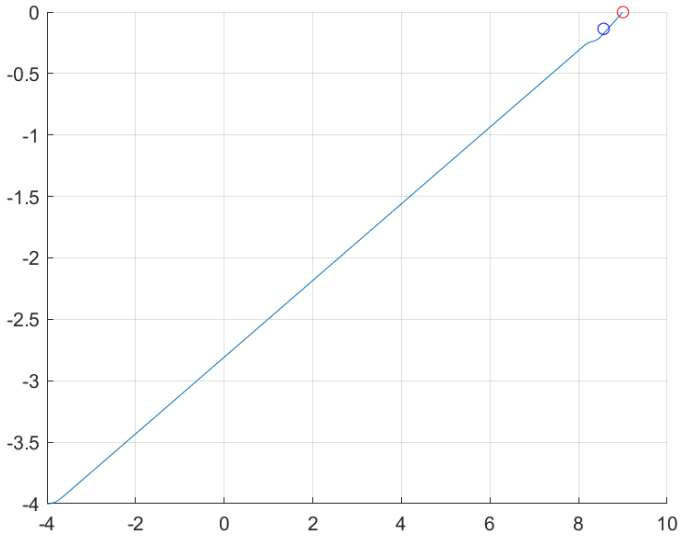
```
obsy = 2.43081;
% Ejecutar Simulación
sim('EvitarObstaculo.slx');
% Obtener los datos de salida de la simulación
x = salida_x.signals.values;
y = salida_y.signals.values;
% Mostrar la trayectoria seguida por el robot
figure;
hold on;
plot(x, y);
plot(obsx,obsy,'bo');    %Obstáculo = círculo azul
plot(refx,refy,'ro');    %Destino = círculo rojo
grid on;
% Esperar antes de la siguiente prueba
pause(1);
```

Se genera la gráfica con la trayectoria, donde se confirma que evita el obstáculo:

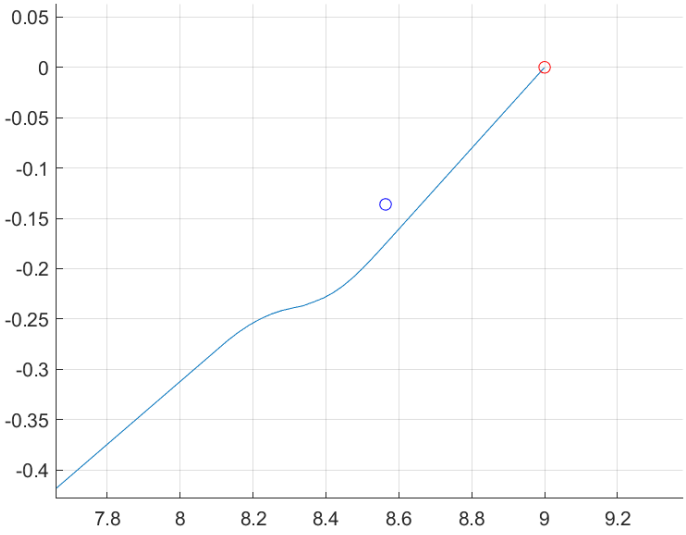


Generamos una gráfica con los siguientes valores. El obstáculo está muy cerca del destino. Sin embargo, el robot consigue esquivarlo y llegar a su destino.

```
% Generar referencia x-y de posición aleatoria
refx = 9;
refy = 0;
obsx = 8.56356;
obsy = -0.136158;
```

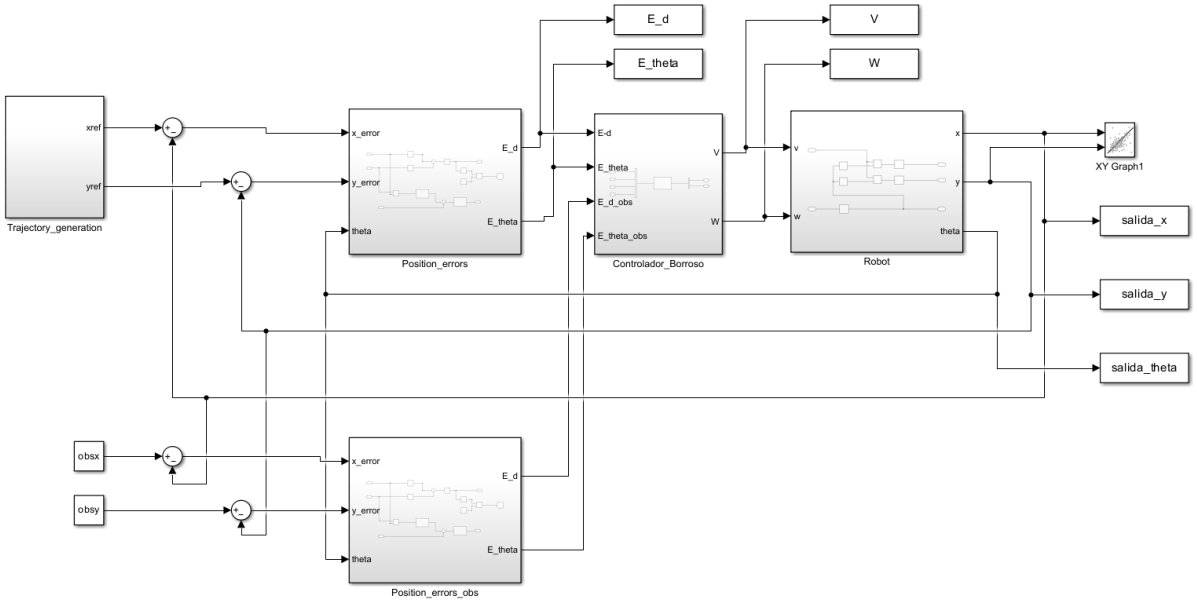


Hacemos zoom para comprobarlo:



e) Sustituya los bloques refx y refy del diagrama de la Figura 2, por el sistema generador de trayectorias utilizado en la primera parte de la práctica. Proponga valores de obsx y obsy en los que el robot esté obligado a esquivar el obstáculo y muestre las trayectorias seguidas por el robot en este caso.

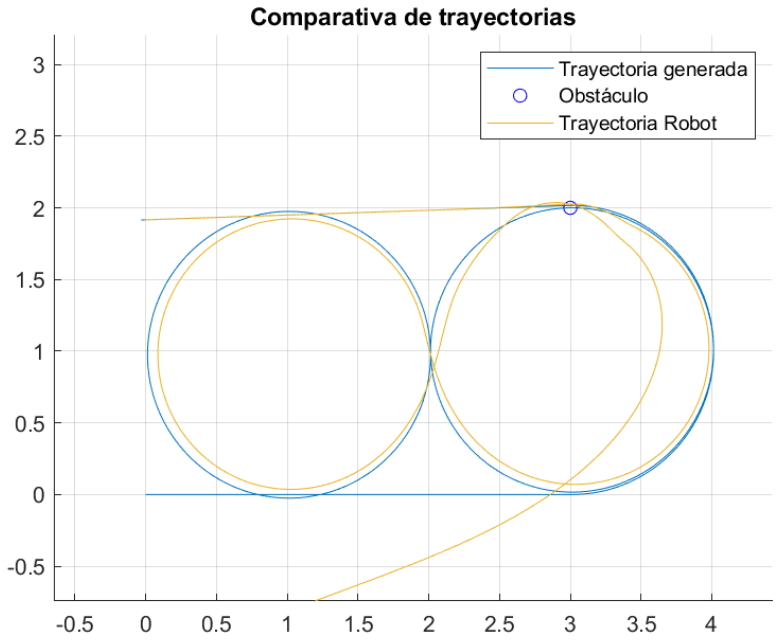
Se sustituye refx y refy por trajectory_generator como antes. Se guarda el archivo como EvitarObstaculos_2.slx.



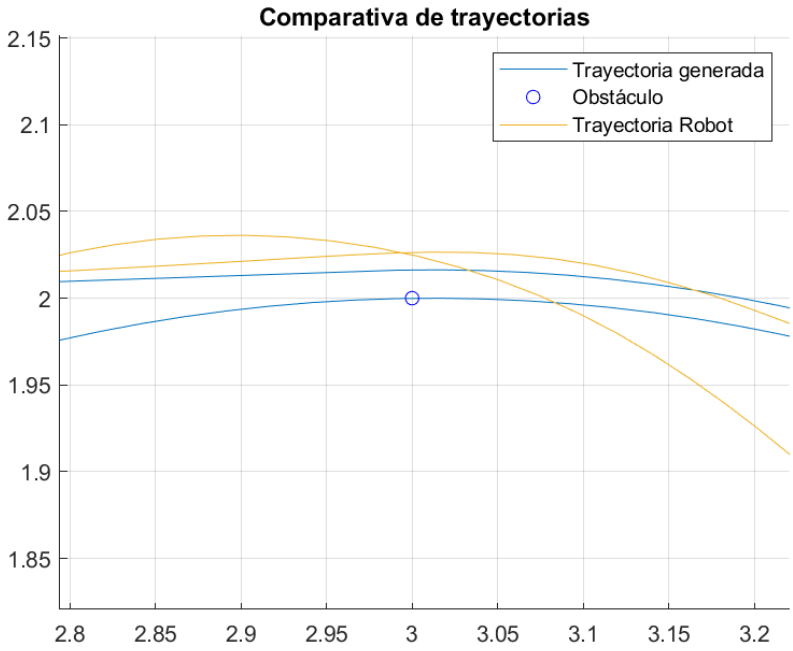
Se ejecuta el script Trayectoria_2.m para comparar las trayectorias generadas, y observar cómo el robot evita el obstáculo:

```
% Tiempo de muestreo
Ts = 100e-3;
% Generar referencia x-y de posición aleatoria
obsx = 3;
obsy = 2;
x_0 = 0.02;
y_0 = 0;
th_0 = 0;
% Ejecutar Simulación
sim('EvitarObstaculo_2.slx');
% Obtener los datos de salida de la simulación
trayectoria_x = salida_refx.signals.values;
trayectoria_y = salida_refy.signals.values;
x = salida_x.signals.values;
y = salida_y.signals.values;
% Mostrar la trayectoria seguida por el robot
figure(1);
hold on;
tray_original = plot(trayectoria_x, trayectoria_y);
obstaculo = plot(obsx,obsy,'bo');    %Obstáculo = círculo azul
tray_robot = plot(x, y);
hold off;
grid on;
legend([tray_original obstaculo tray_robot], {'Trayectoria
generada','Obstáculo','Trayectoria Robot'});
title('Comparativa de trayectorias');
```

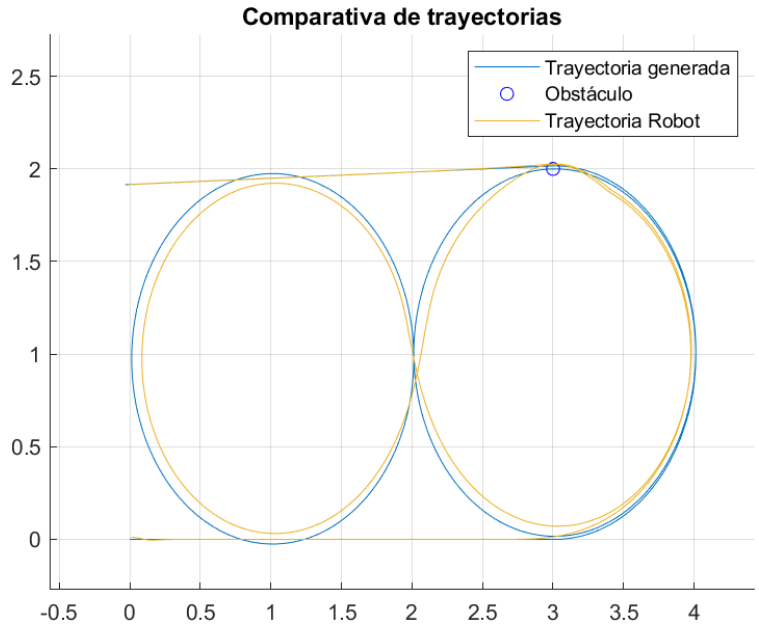
Se observa que sucede como en la anterior parte, el punto de inicio del robot es el (-4,-4) y al comienzo el robot no realiza la trayectoria igual pero sí acaba en el mismo punto. Además, el robot evita el obstáculo y continúa con la trayectoria.



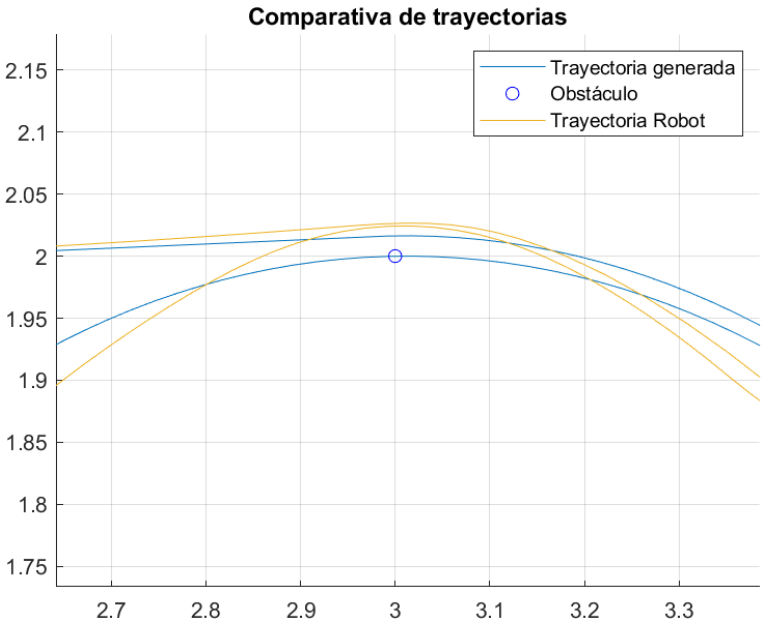
Se realiza zoom para comprobar que evita el obstáculo:



Se añade otro ejemplo donde el punto de inicio del robot es (0.1,0.1) en vez de (-4,-4) y cómo la trayectoria es más similar a la generada pero desviándose para evitar al obstáculo.



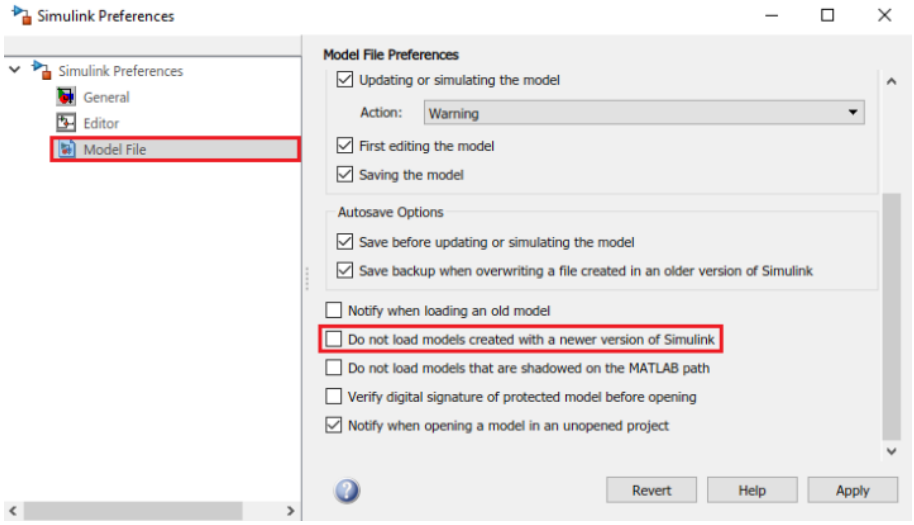
Se comprueba el resultado haciendo zoom:



Problemas encontrados y soluciones

Tuvimos problemas con las versiones de Matlab y Simulink, ya que, se realizó en las versiones R2022a, R2022b e incluso desde Matlab Online (R2023b). Para poder abrir y editar los modelos desde las distintas versiones se exportó a la versión anterior que se necesitaba en ese momento. Sin embargo, seguían saliendo mensajes de error al intentar abrirlos, por lo que, tuvimos que permitir cargar modelos creados con una versión más reciente de Simulink. Para ello, en la barra de herramientas de MATLAB, en la pestaña INICIO, pulsamos en Preferencias -> Simulink -> Abrir Preferencias de Simulink.

En el panel de Archivo de Modelo desmarcamos la opción "No cargar modelos creados con una versión más reciente de Simulink".



Además, tuvimos que bajar la velocidad lineal para que el robot continuase la trayectoria correctamente, evitando los obstáculos en su caso, a tiempo.