Universidad de Alcalá

Práctica 2. Control Borroso (I)

Sistemas de Control Inteligente

Grado en Ingeniería Computadores
Grado en Ingeniería Informática Grado en Sistemas de Información

Universidad de Alcalá

En esta práctica, se estudiará el diseño de controladores borrosos para el control de posición de robots móviles. Se hará uso de Matlab y Simulink como en prácticas anteriores, siendo necesaria la instalación en este caso de la "Fuzzy Logic Toolbox". La práctica consta de dos partes:

Parte 1. Diseño de un control borroso de posición para un robot móvil.

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

Parte 1. Diseño de un control borroso de posición para un robot móvil.

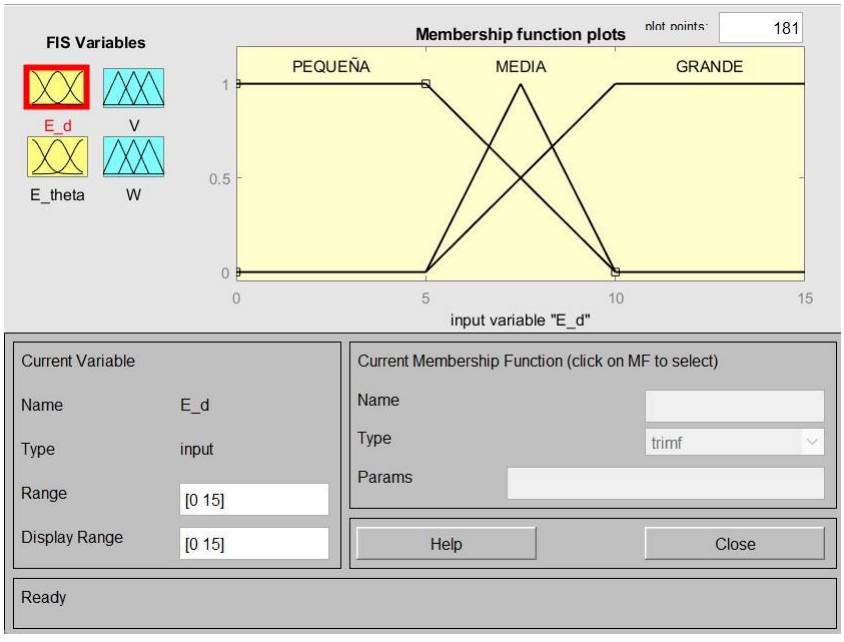
1.1. Desarrollo de la práctica

Como se indicó al comienzo del documento, el objetivo de esta primera parte de la práctica es diseñar un controlador de posición basado en la teoría de control borroso. Dicho controlador tiene como entradas los errores de distancia y ángulos comentados anteriormente y genera a su salida los valores de velocidad (angular y lineal) que serán entradas en el bloque de simulación del robot.

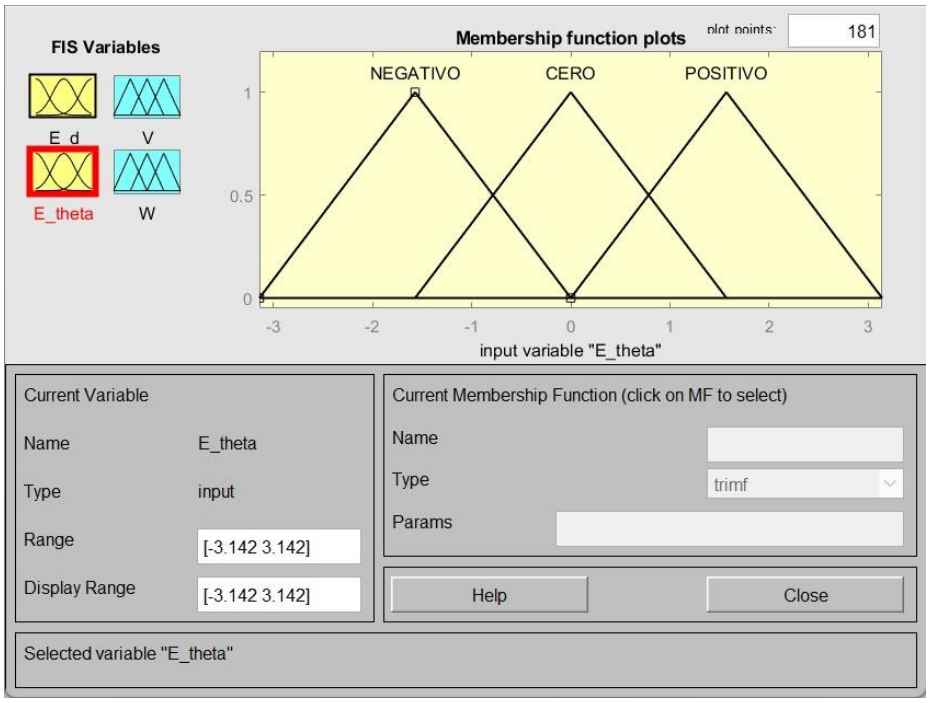
Para ello se piden los siguientes apartados:

Hemos creado un controlador borroso mediante el comando "fuzzy" o "fuzzyLogicDesigner", el cual hemos puesto en la terminal de Matlab. Una vez dentro hemos creado un controlador con 2 entradas (E_d y E_theta) y dos salidas (V y W). Las entradas corresponden con el error de distancia y el error de ángulo respectivamente. Por otro lado, las salidas corresponden a la velocidad lineal (V) y la velocidad angular (w). A continuación, se mostrará como es cada una de estas entradas y salidas:

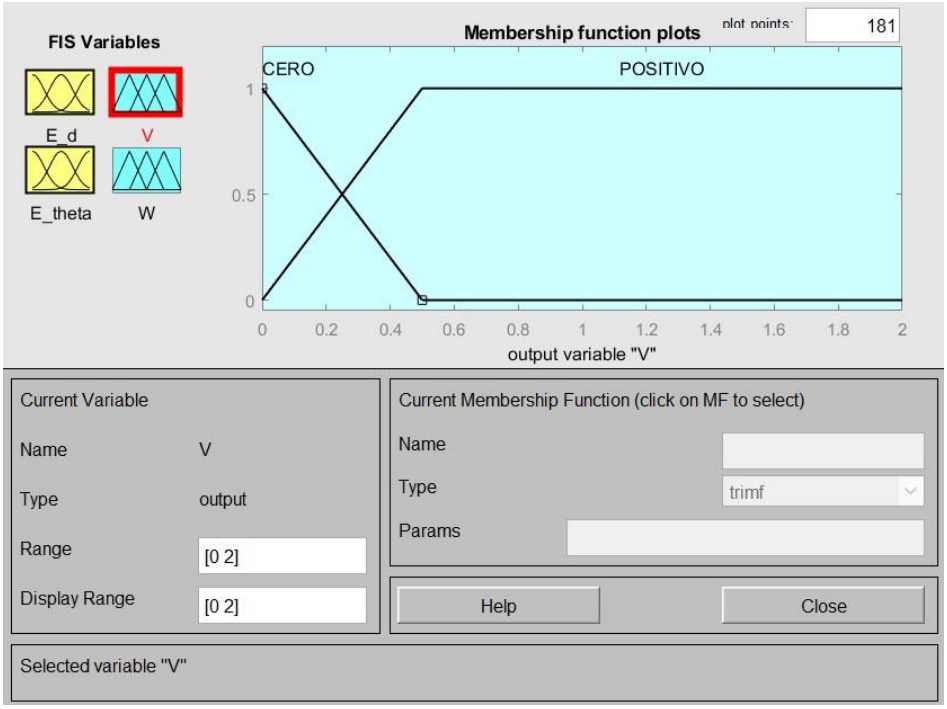
1. **E_d**: tiene un rango de valores de entre 0 hasta 15, y tiene tres tipos de errores:
   PEQUEÑA (función trapezoidal con valores 0 0 5 y 10, que corresponden con el inicio de la función, el primer máximo del trapecio, el segundo máximo del trapecio y el final de la función respectivamente), por otro lado, esta MEDIA (función triangular con valores 5 7.5 y 10, los cuales corresponden con el inicio, el máximo y el fin de la función respectivamente), y por último, esta GRANDE (función trapezoidal con valores 5 10 15 y 15). Se vería de la siguiente manera:



2. **E_theta**: tiene un rango de valores de entre -π hasta π, tiene tres tipos de errores: NEGATIVO, CERO y POSITIVO (Las tres son funciones triangulares). Se vería de la siguiente manera:

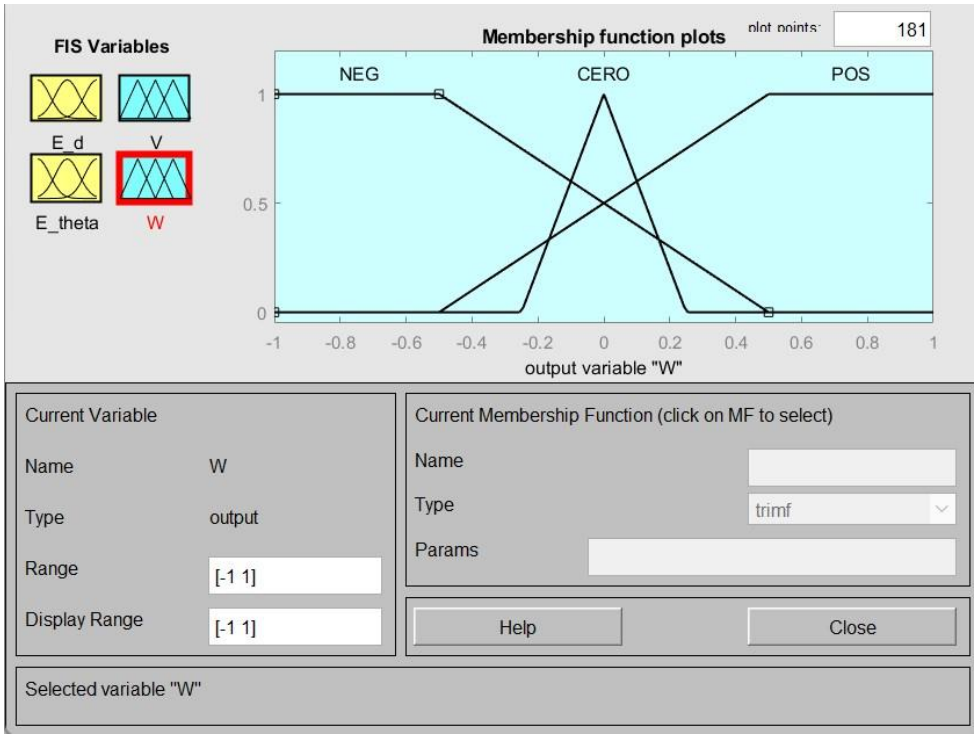


3. **V**: tiene un rango de valores de entre 0 y 2, tiene 2 tipos de acción: CERO (función trapezoidal que se usa para cuando queremos que el robot deje de moverse en línea recta) y POSITIVO (función trapezoidal que se usa para cuando queremos que avance el robot en línea recta). Otra opción que habíamos pensado era poner una función extra que correspondiera con NEGATIVO, que sería una marcha atrás del robot, pero no nos interesaba eso para el funcionamiento de la práctica. Se veía de la siguiente manera:

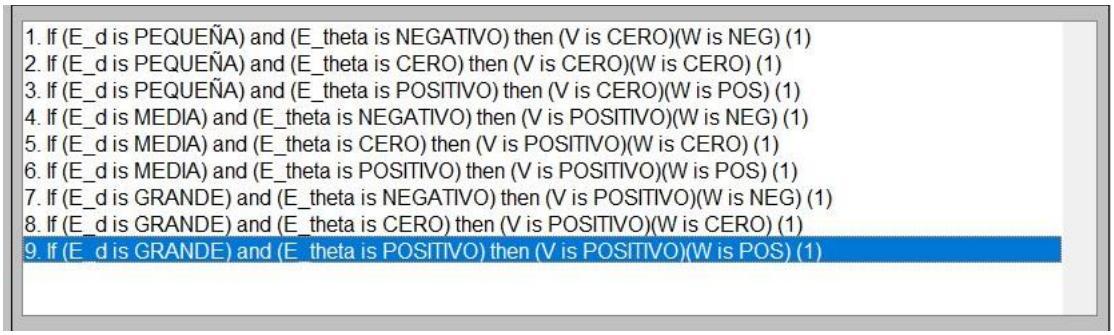


4. **W**: tiene un rango de valores de entre 0 y 1, tiene 2 tipos de acción: NEGATIVO (función trapezoidal que se usa para cuando queremos se mueva hacia la izquierda), CERO (función triangular que usamos para

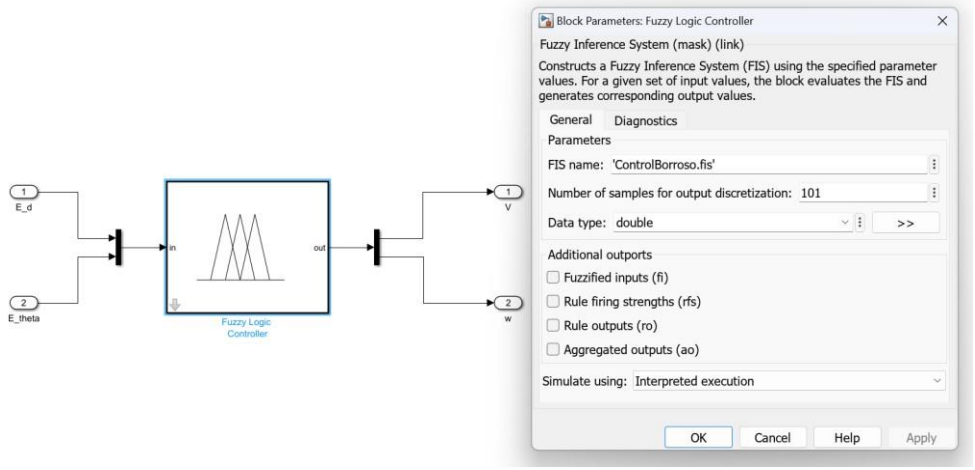
cuando no queremos que el robot tenga una velocidad angular, es decir, cuando no queremos que gire) y POSITIVO (función trapezoidal que se usa para cuando queremos que el robot se mueva hacia la derecha). Se veía de la siguiente manera:



Una vez se hayan ajustado las entradas y las salidas, se deben realizar las reglas del controlador, las cuales hemos pensado que las que mejor actuarían con nuestro diseño son las siguientes:



Una vez hayamos guardado el control borroso debemos importarlo en el modelo de Simulink como se indica en la siguiente imagen:

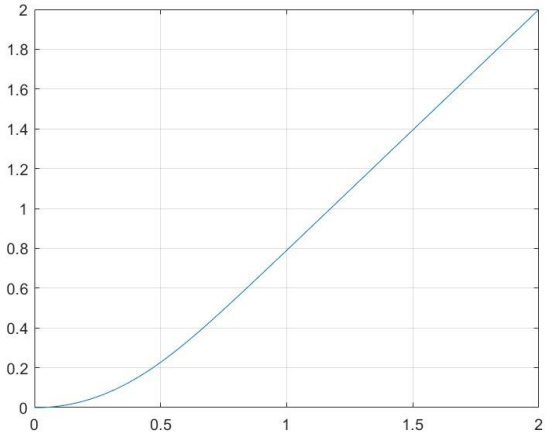


Con esto ya tenemos nuestro bloque controlador, y con esto tenemos lista la primera parte de la práctica.

a) 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.

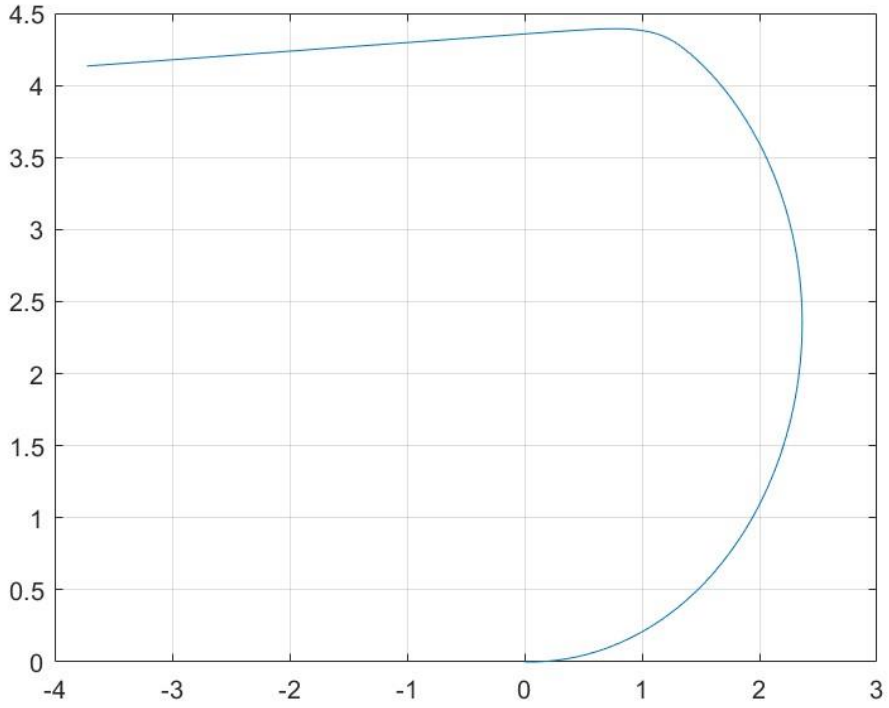
```
%Tiempo de muestreo
Ts=100e-3
% Referencia x-y de posicion
refx=10*rand-5; refy=10*rand-5; %
Ejecutar Simulacion
sim('PositionControl.slx')
% Mostrar
x=salida_x.signals.values;
y=salida_y.signals.values; figure;
plot(x,y); grid on;
hold on;
```

Antes de comenzar con el código, vamos a realizar una prueba para comprobar si el robot avanza bien a su destino, como, por ejemplo, refx=2 y refy=2.

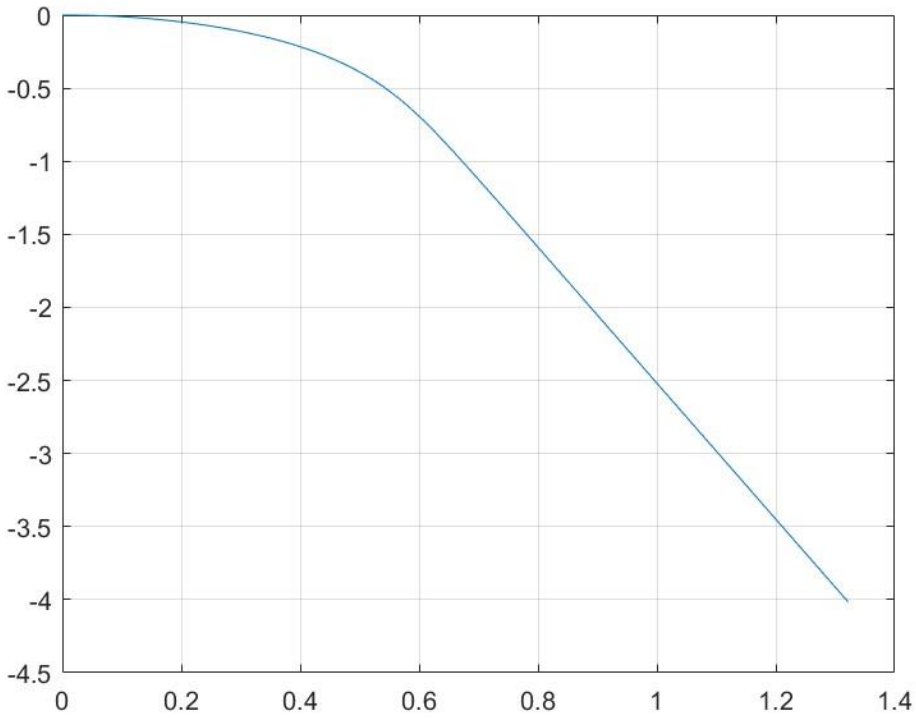


Como podemos observar, el robot llega a la posición deseada. A continuación, se mostrarán una sería de pruebas realizadas con distintas posiciones finales aleatorias.

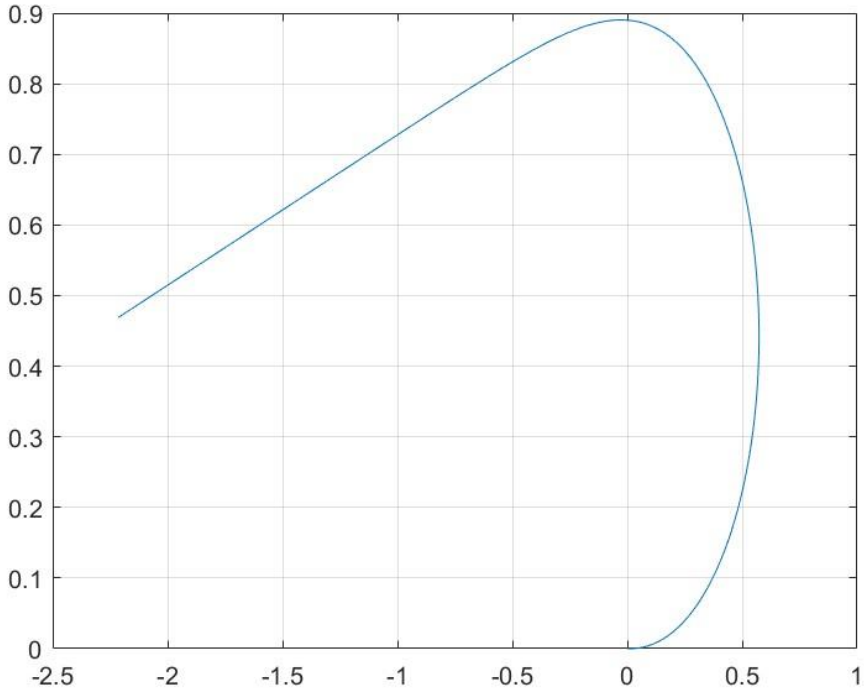
- **Refx = -3.730131837064939 y Refy = 4.133758561390193**



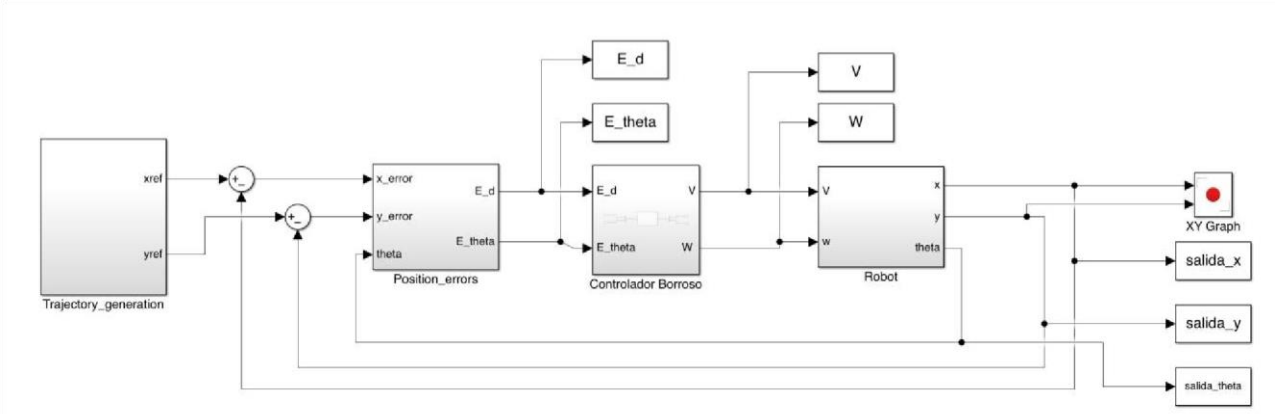
- **Refx = 1.323592462254095 y Refy = -4.024595950005905**



- **Refx = -2.215017811329516 y Refy = 0.468815192049838**



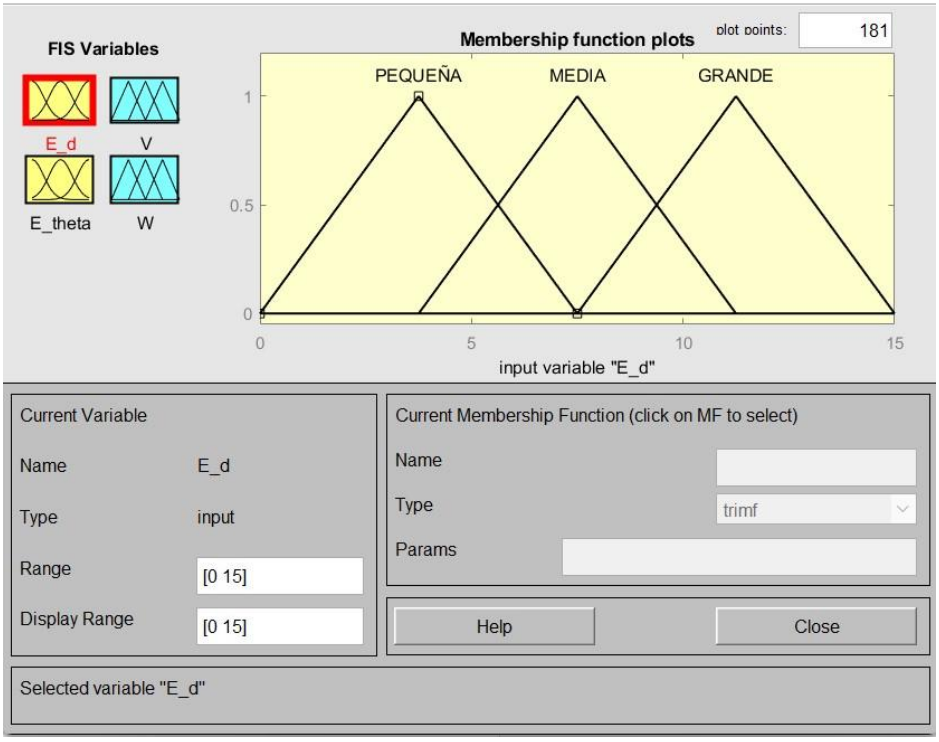
b) Sustituya las referencias de posición (bloques refx y refy) por el generador de trayectorias proporcionado ("Trajectory_generator.slx") y compruebe el funcionamiento del sistema que se muestra en la imágen. Realice los cambios en el diseño del controlador borroso que sean necesarios para que el robot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



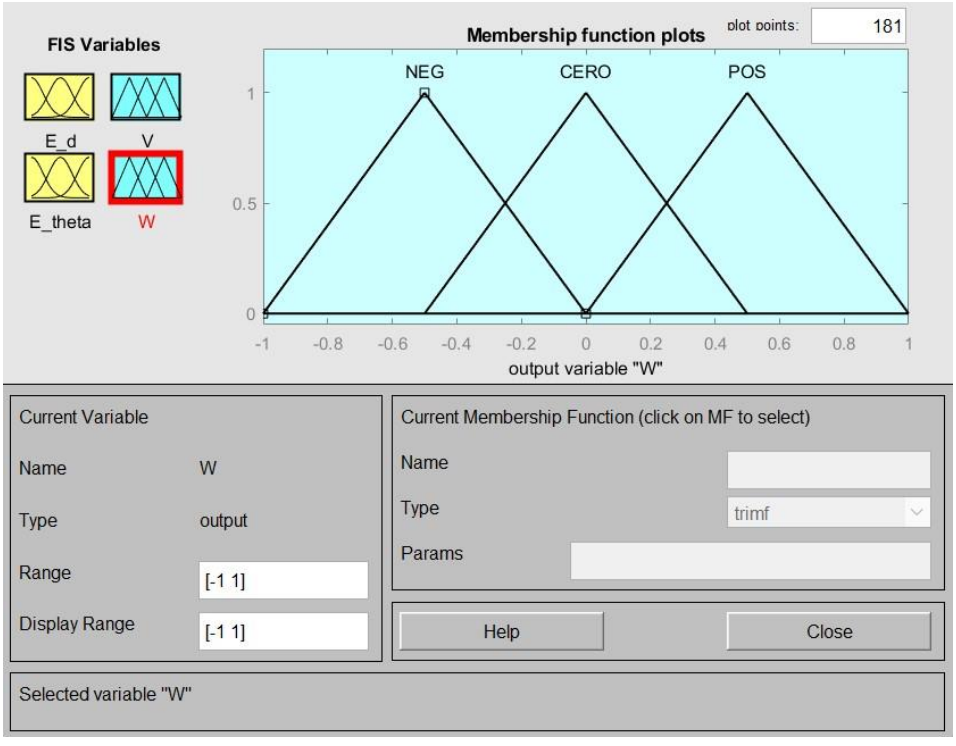
La imagen muestra la arquitectura del subsistema "Trajectory_generation", que generará la trayectoria que debe seguir el robot móvil. Este subsistema se proporciona dentro de la documentación de la práctica en el fichero "Trajectory_generator.slx". Para utilizar este bloque es necesario inicializar los parámetros iniciales de la trayectoria (x_0, y_0, th_0) y el periodo de muestro (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.

Para esta parte del trabajo hemos modificado el controlador borroso del apartado anterior, concretamente hemos modificado el input de E_d y el output W de las siguientes maneras:

- **E_d**: hemos cambiado PEQUEÑA y GRANDE a funciones triangulares en vez de funciones trapezoidales y los valores de MEDIA, a continuación, se muestra cómo queda:



- **W**: hemos cambiado PEQUEÑA y GRANDE a funciones triangulares en vez de funciones trapezoidales y los valores de MEDIA, a continuación, se muestra cómo queda:

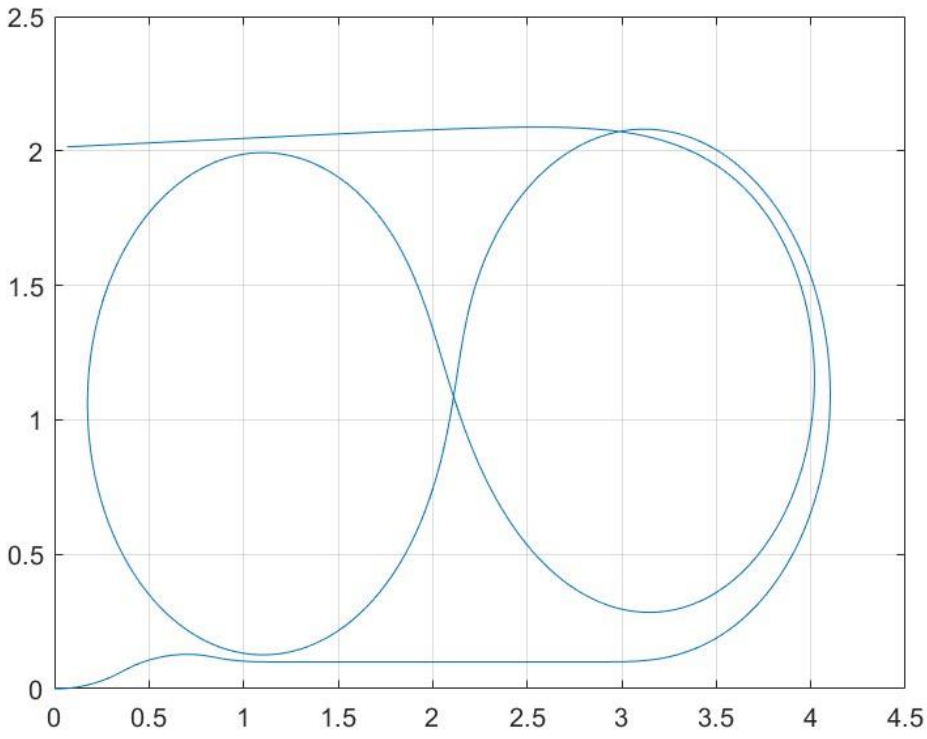


Tras estas modificaciones en el controlador borroso guardamos el nuevo controlador y lo insertamos en "Trajectory_generator.slx". Ahora creamos un script en Matlab el cual debe tener el siguiente código:

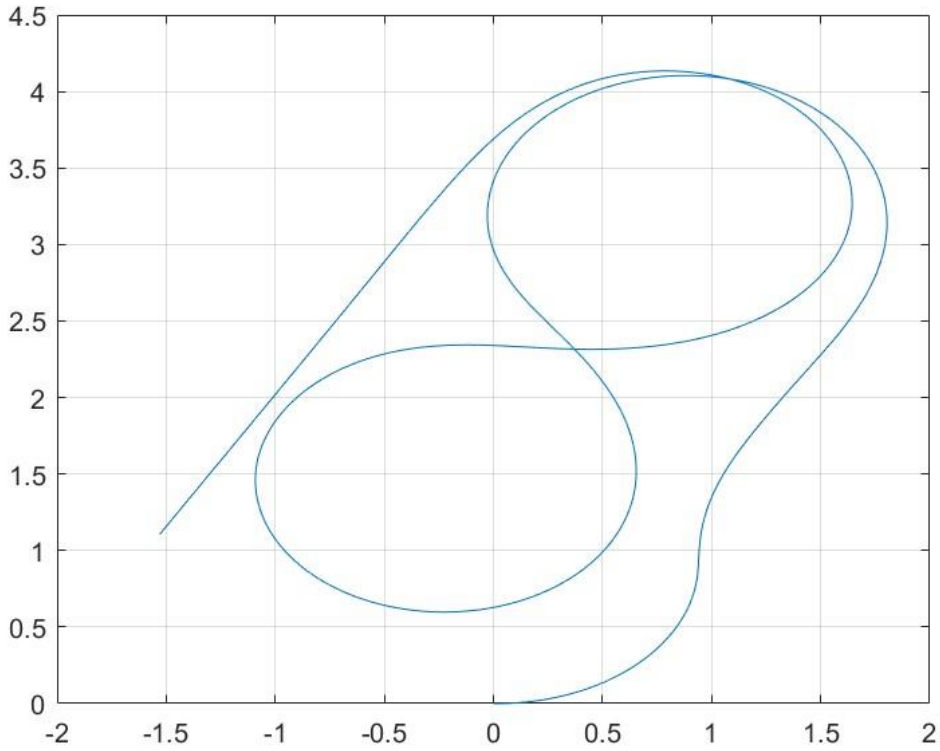
```
%Tiempo de muestreo
Ts=100e-3
% Referencia x-y de posicion x_0 =
0.1; y_0 = 0.1; th_0 = 0;
% Ejecutar Simulacion
sim('Trayectory_PositionControl.slx')
% Mostrar
x=salida_x.signals.values;
y=salida_y.signals.values; figure;
plot(x,y); grid on; hold on;
```

Tanto x_0 y y_0 no están inicializadas en 0 (posición inicial del robot) ya que si comienza en esta posición el robot terminará sin haber hecho ningún recorrido, por tanto, inicializamos la posición de este lo más cercano posible a 0 pero sin que esta sea 0. Por otro lado, th_0 esta inicializado en 0 debido a que esta es la orientación del robot, por lo que, para que sea lo más aproximado posible al objetivo, este valor debe estar inicializado a 0. A continuación se mostrarán distintos resultados con distintas th_0 inicializadas.

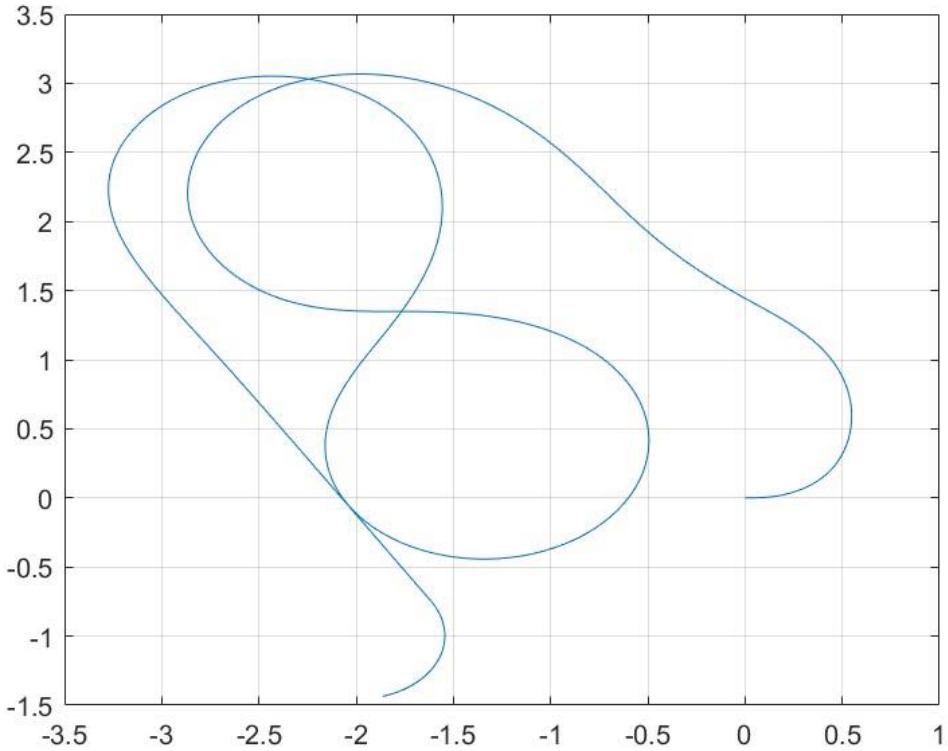
- **th_0 = 0**: En esta ejecución podemos observar que la orientación de la figura es la correcta y se aproxima bastante el objetivo que buscamos.



- **th_0 = 1**: Esta ejecución sigue el mismo recorrido que la anterior, pero el problema es que no está bien orientada.



- **th_0 = 2**: Al igual que la figura anterior, el problema es la orientación.



Universidad de Alcalá

Práctica 3. Control Borroso (II)

Sistemas de Control Inteligente

Grado en Ingeniería Computadores Grado en Ingeniería Informática Grado en Sistemas de Información

Universidad de Alcalá

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.

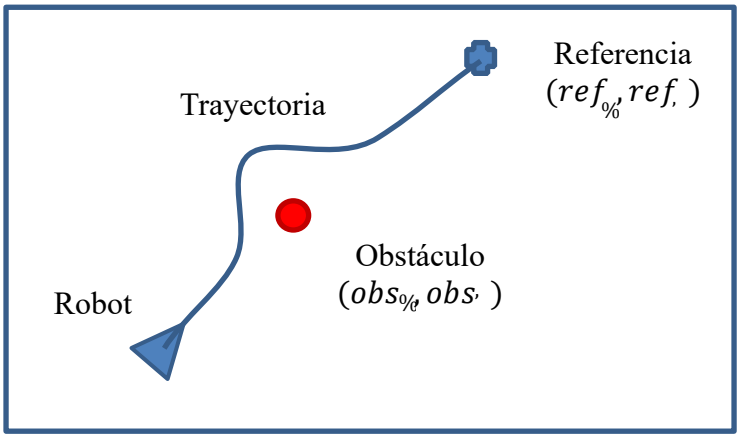


Figura 1. Diagrama de funcionamiento del control de posición con evitación de obstáculos.

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:

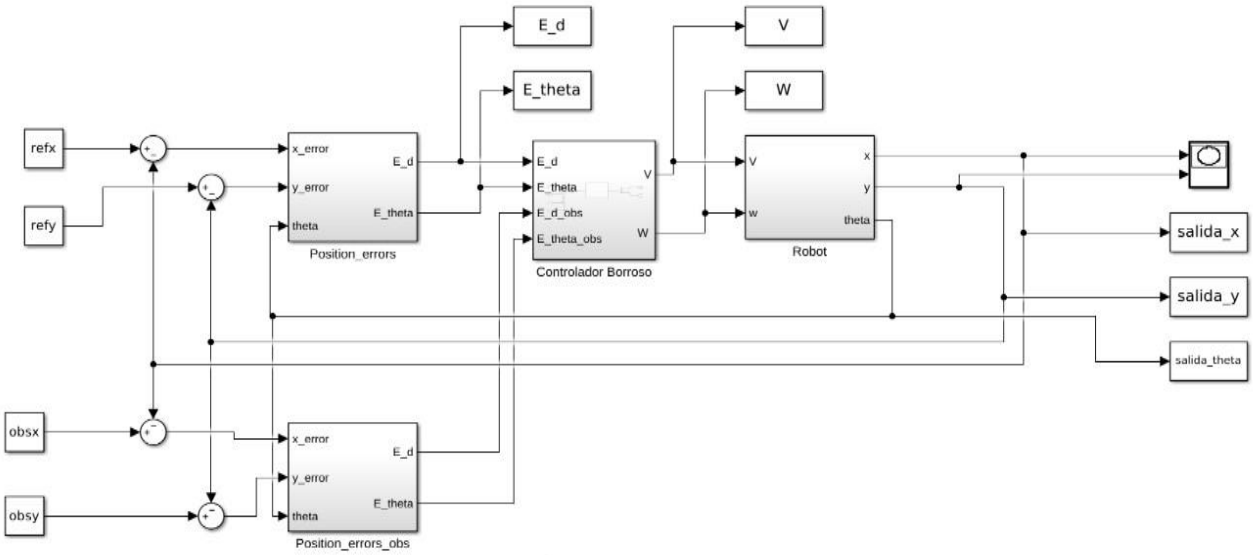


Figura 2. Esquema general de un control de posición con evitación de obstáculos

1. Desarrollo de la práctica

El controlador realizado tendrá como entradas los errores de distancia y ángulo a la referencia de posición (E_{-}, $E_{/}$) y los errores de distancia y ángulo a la posición del obstáculo ($E_{-,012}$, $E_{/,012}$). 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 3.

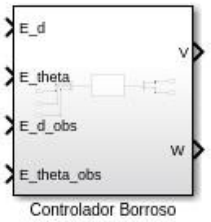


Figura 3. Bloque Simulink del controlador borroso con evitación de obstáculos.

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).

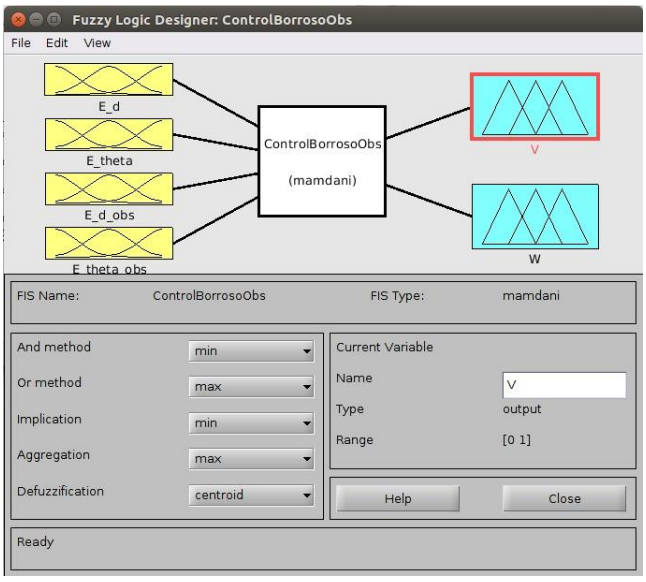
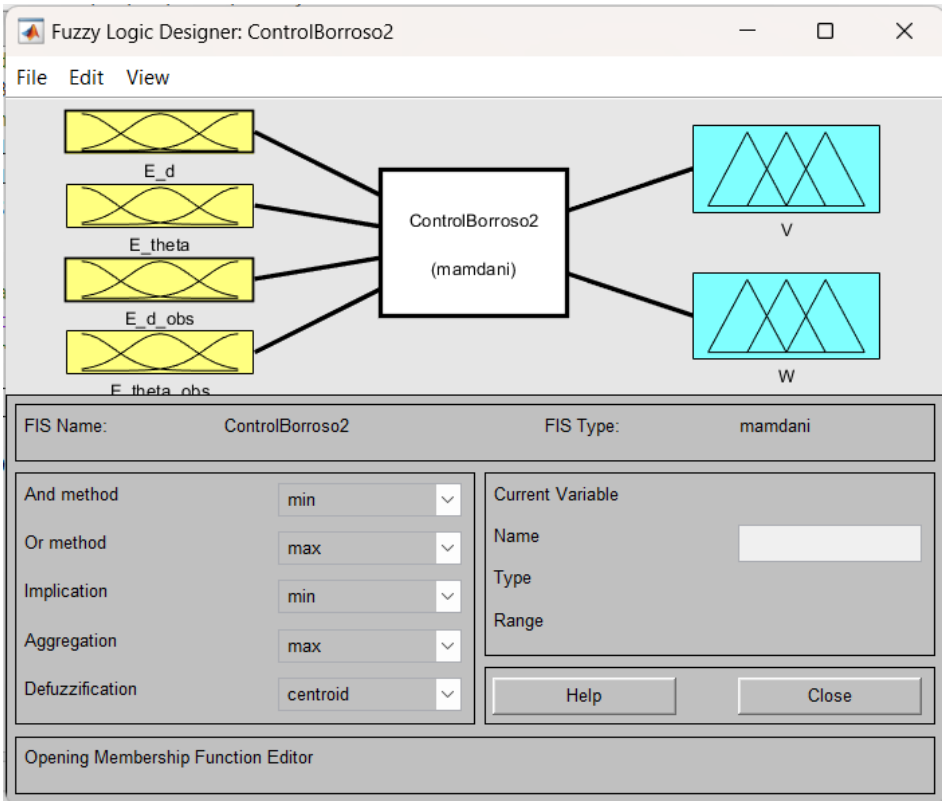


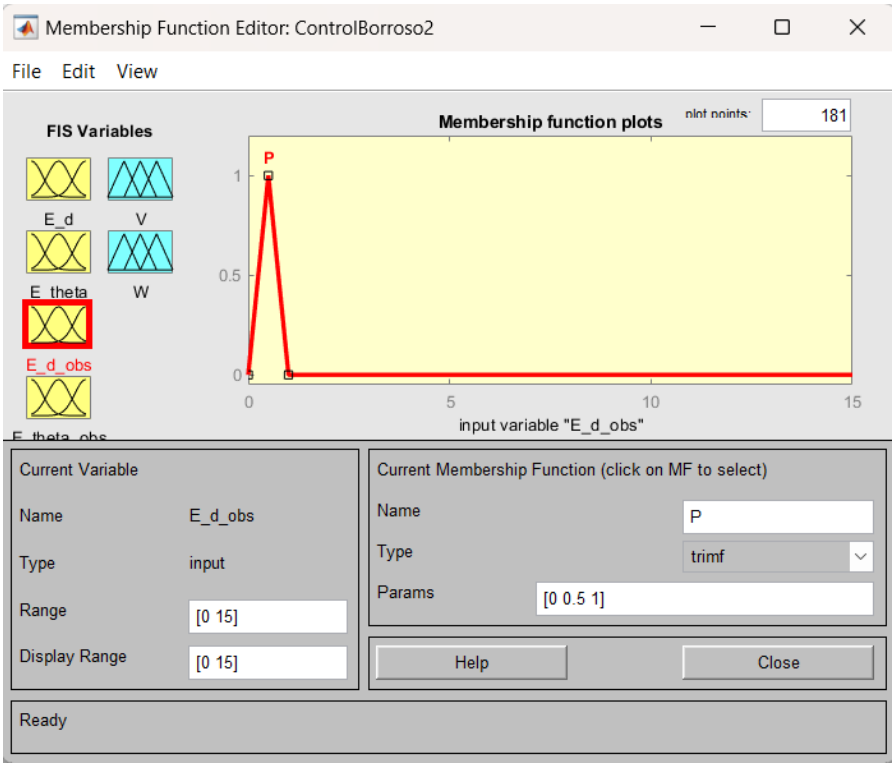
Figura 4. Diseño del control borroso mediante la toolbox de Matlab

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.

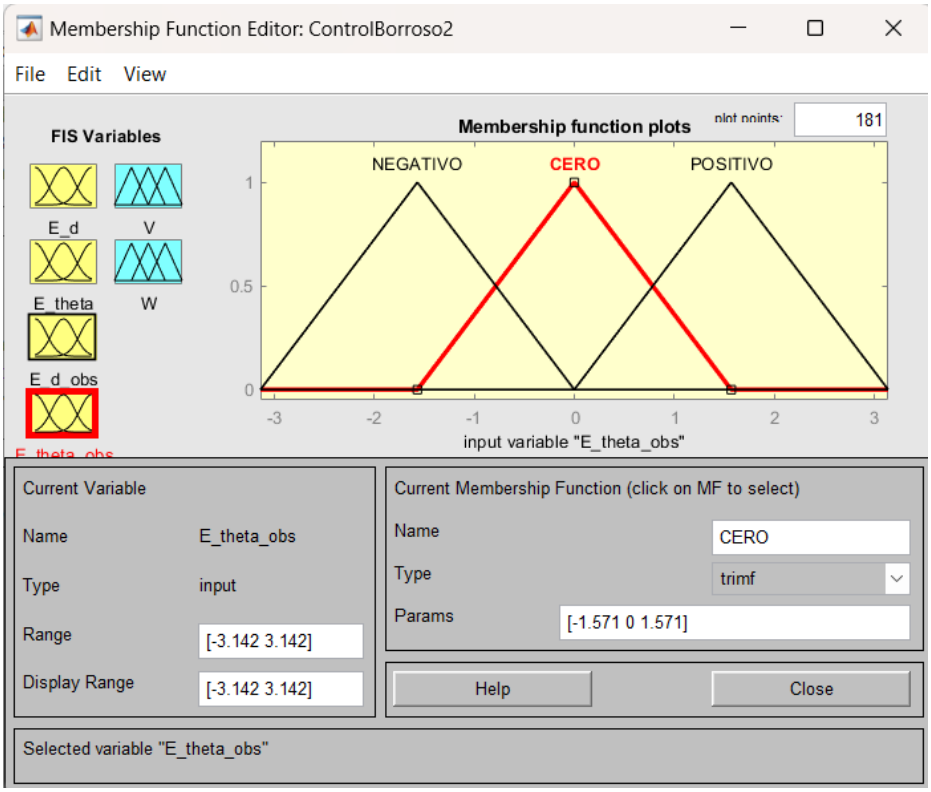
Creamos las nuevas funciones de pertenencia:



- **E_d_obs**: es la distancia restante al obstáculo desde nuestra posición actual. En este caso tan solo nos importa si estamos cerca del objeto para empezar a variar nuestra trayectoria, por lo que sólo hemos tenido en cuenta una función de pertenencia P para cuando la distancia al objeto sea de 0.5, ya que hemos considerado que el obstáculo tenía diámetro 1.



- **E_theta_obs**: esta regla es completamente igual que la del E_theta pero en vez de con el destino final, con el obstáculo.



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).

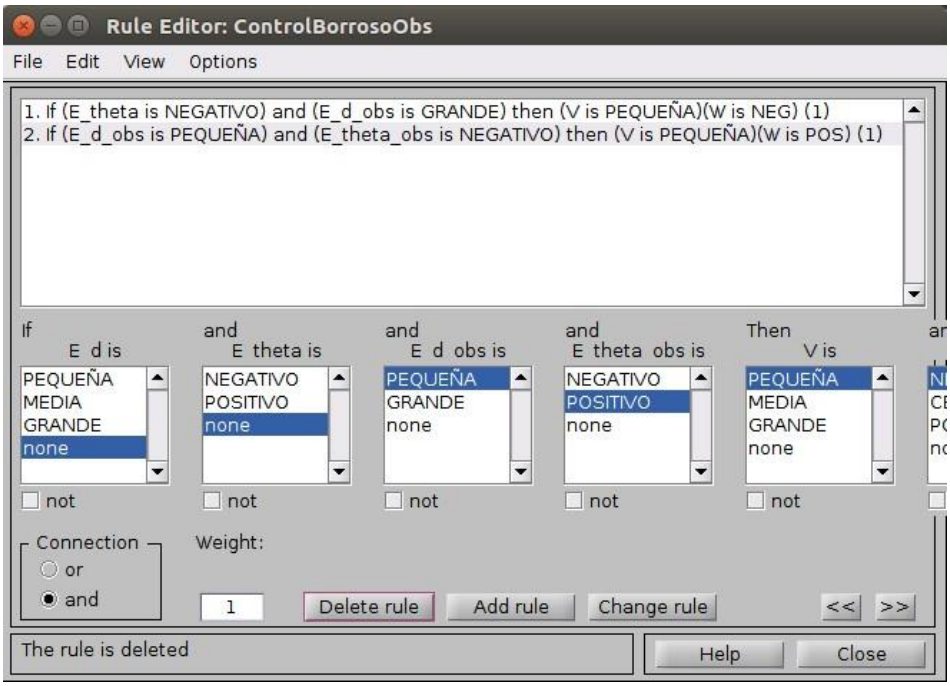
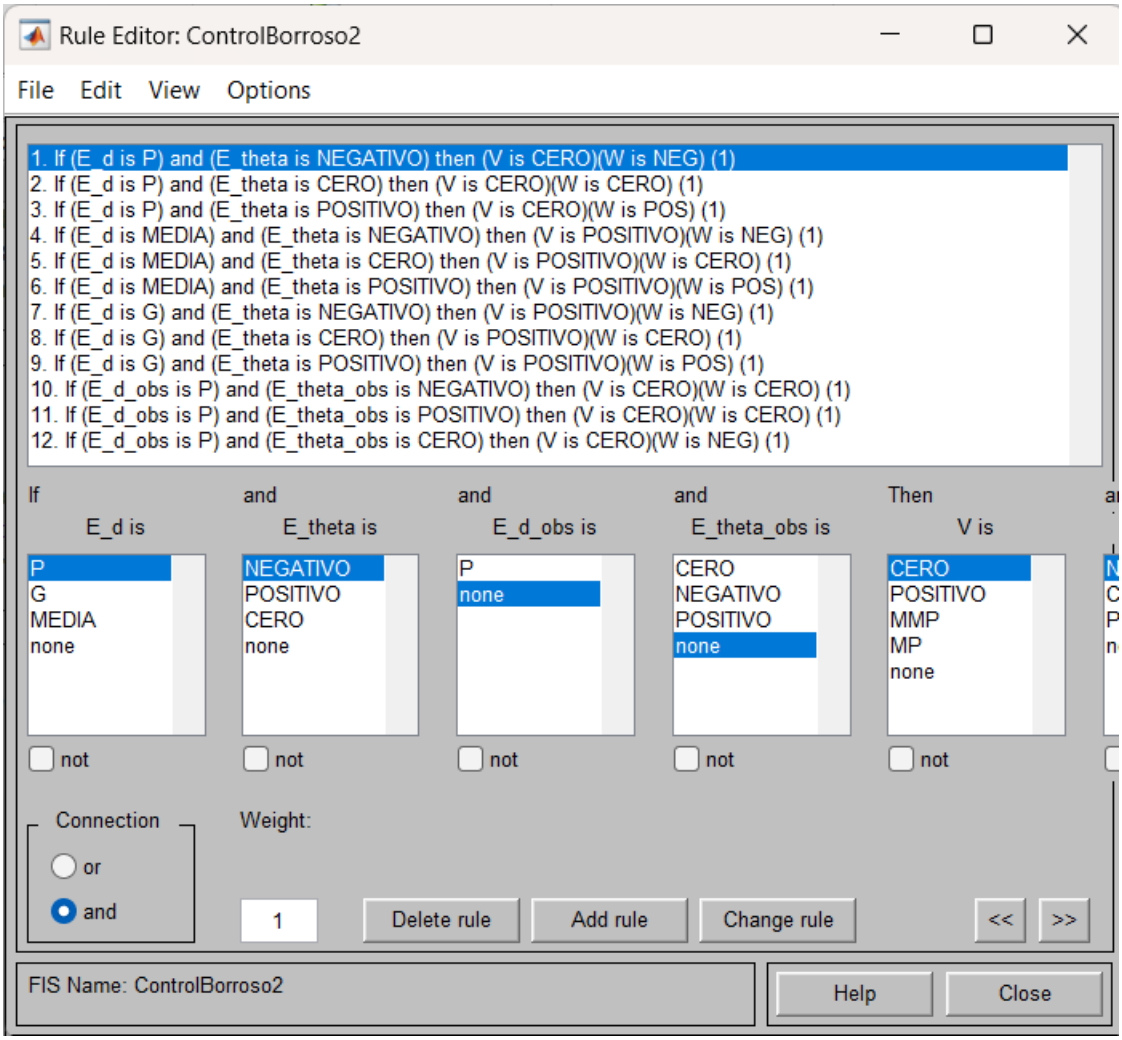


Figura 5. Diseño de reglas para el controlador de evitación de obstáculos



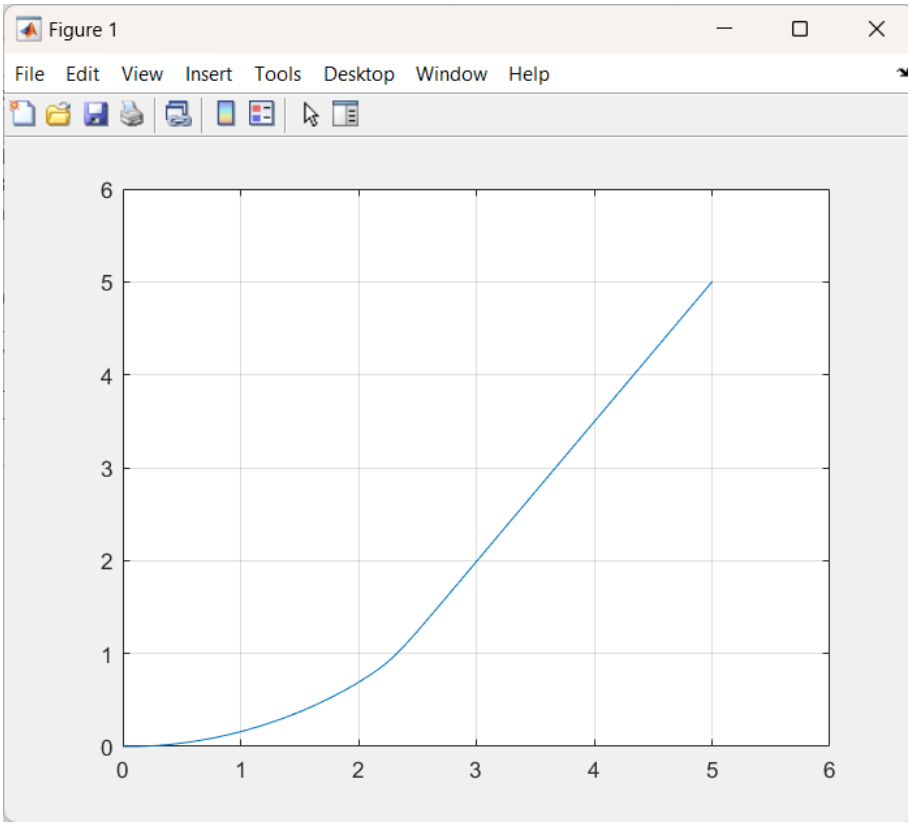
Esta fue la tabla de reglas que creamos, que básicamente son las 9 anteriores mas 3 que tienen en cuenta el obstáculo.
Las reglas 10 y 11 son para que si estamos a un lado del objeto, centremos de nuevo la trayectoria hacia el destino y la regla 12 se encarga de que si nuestra trayectoria es justo hacia el obstáculo (CERO), cambiamos el ángulo para virar y esquivarlo.

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.

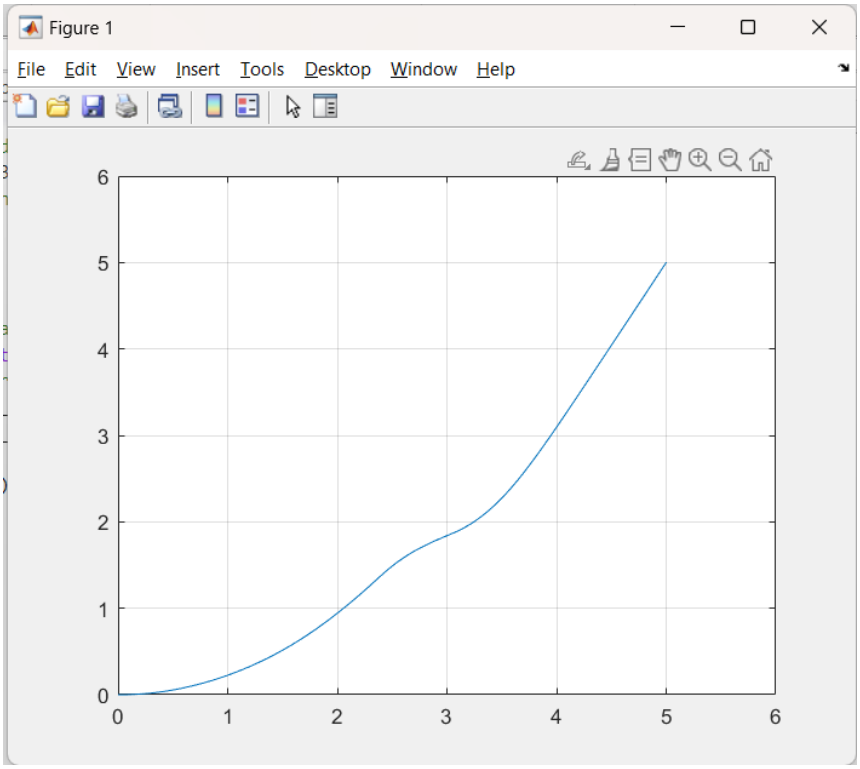
```
%Tiempo de muestreo
Ts=100e-3
% Referencia x-y de posicion
refx=5;     refy=5; obsx=2.5;
obsy=2.5;
% Ejecutar Simulacion sim('EvitarObstaculo.slx')
% Mostrar
x=salida_x.signals.values;
y=salida_y.signals.values;
figure;
plot(x,y); grid
on; hold on;
```

Hemos sustituido la ubicación del obstáculo de (2.5, 2.5) a (3, 2) ya que nuestra trayectoria no pasaba por el primer punto, pero sí por este segundo.
Veamos la comparación de la trayectoria en una simulación sin y con el objeto:

- Sin objeto: Vemos que pasa exactamente por (3, 2)

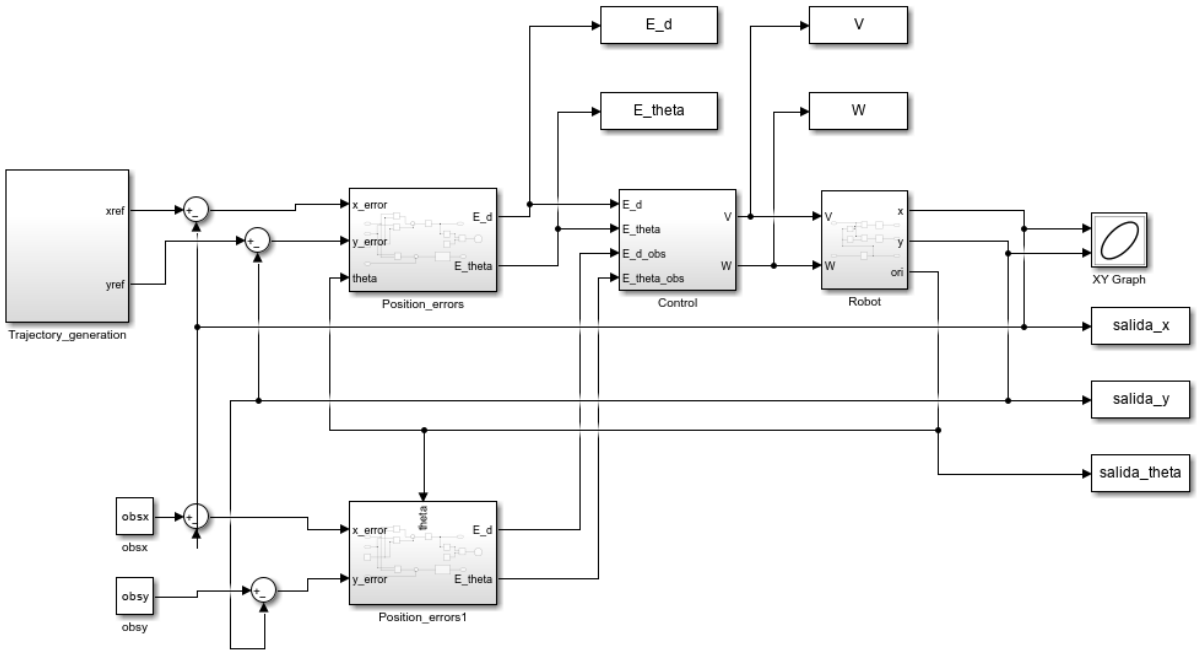


- Con objeto: Vemos que elude el punto (3, 2), bordeando el objeto imaginario.



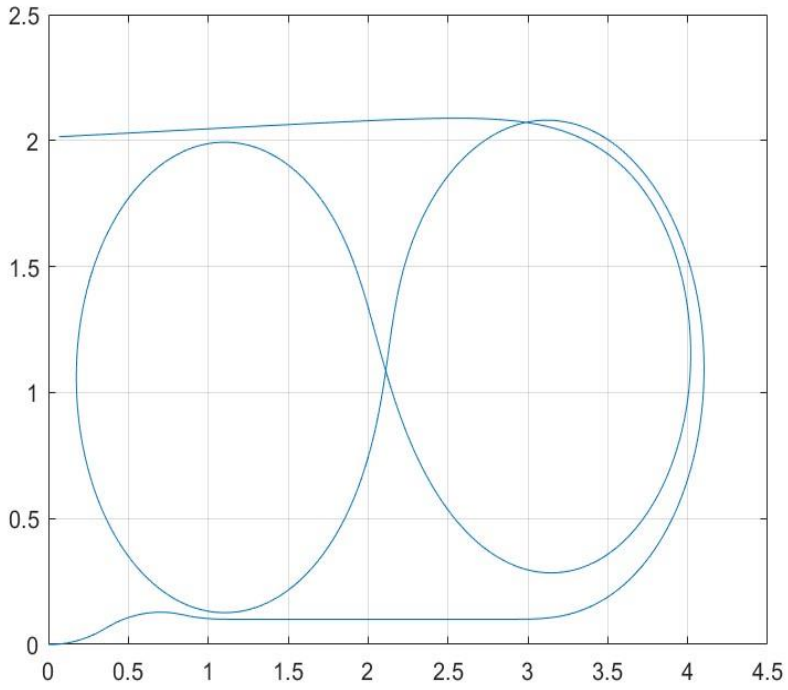
e) Sustituya los bloques refx y refy del diagrama de la Figura 2, por el sistema generador de trayectorias utilizado en la primera parte de la práctica. Proponga valores de obsx y obsy en los que el robot esté obligado a esquivar el obstáculo y muestre las trayectorias seguidas por el robot en este caso.

Quedaría de la siguiente forma:



De nuevo observemos que evita el obstáculo, veamos una imagen de la trayectoria con y sin obstáculo.

- Sin obstáculo: la trayectoria circunda el punto (3,2)



- Con obstáculo: vemos que evita dicho punto.