Universidad de Alcalá

Escuela Politécnica Superior

Universidad de Alcalá

Práctica 2

Control Borroso

Sistemas de control inteligente

Laboratorio Lunes 12:00 – 14:00

Grado en Ingeniería Informática – Curso 2018/2019

Eduardo Graván Serrano – 03212337L

Marcos Barranquero Fernández – 51129104N

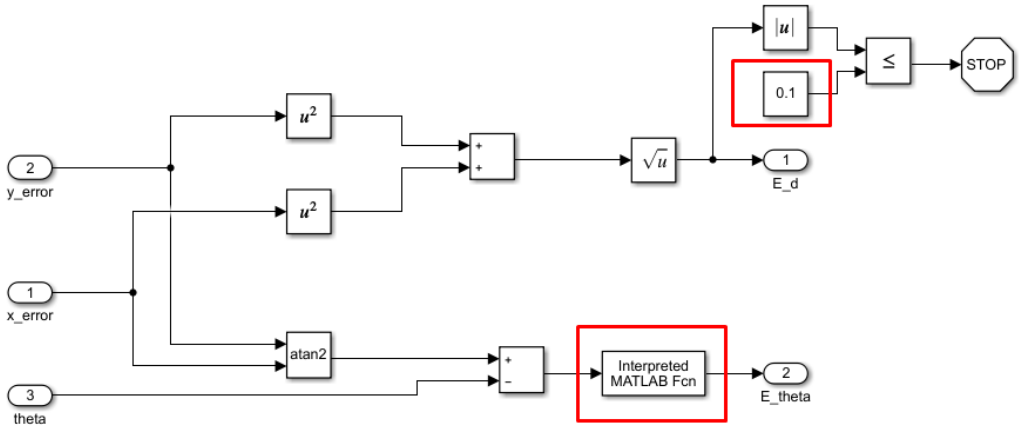
CONTENIDO

Consideraciones sobre prácticas previas 3
Parte 1 – Control borroso 3
Ejercicio 1 3
Ejercicio 2 6
Parte 2 – Control borroso con obstáculos 8
Ejercicio 1 8
Ejercicio 2 11

CONSIDERACIONES SOBRE PRÁCTICAS PREVIAS

Para hacer funcionar adecuadamente el bloque de control borroso, se ha añadido al bloque de cálculo de error un nuevo bloque de función de Matlab. La función que contiene es WrapToPi, que cierra los valores de salida en el rango de -Pi a +Pi.

También se ha modificado la condición del STOP en función de la distancia de error, de forma que pasa a detenerse cuando el error es menor de 0.1, en lugar de 0.01 como en prácticas anteriores.

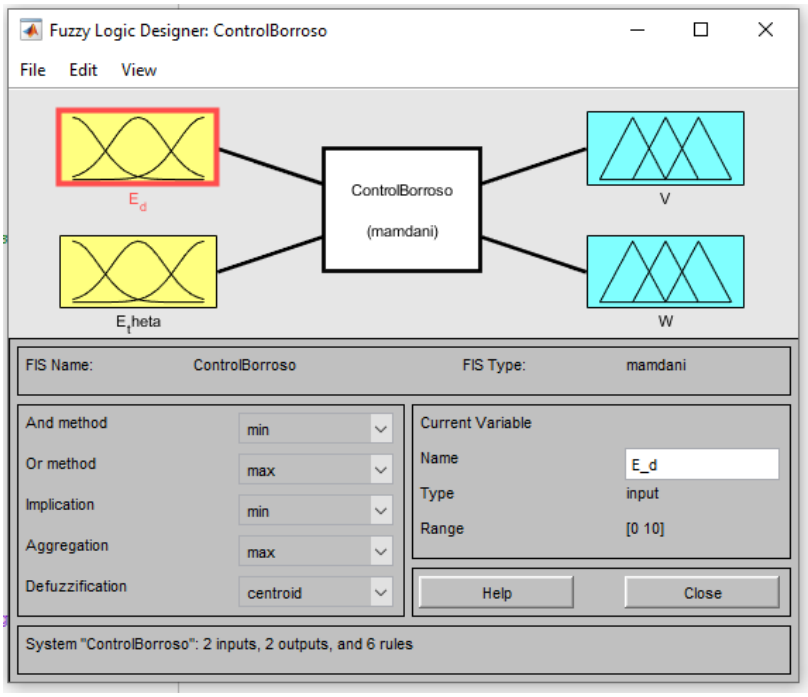


PARTE 1 – CONTROL BORROSO

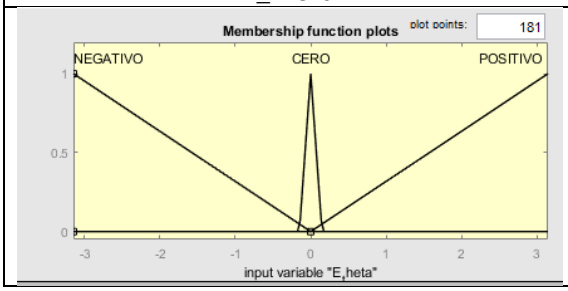
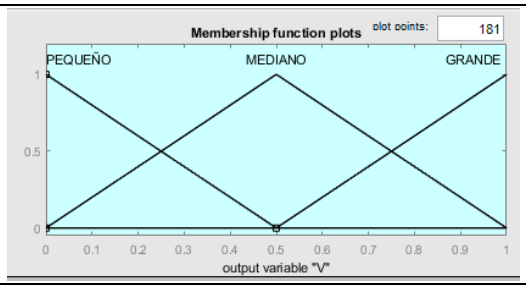
En esta parte se propone diseñar y emplear un controlador borroso para realizar los desplazamientos del robot de las prácticas anteriores.

EJERCICIO 1

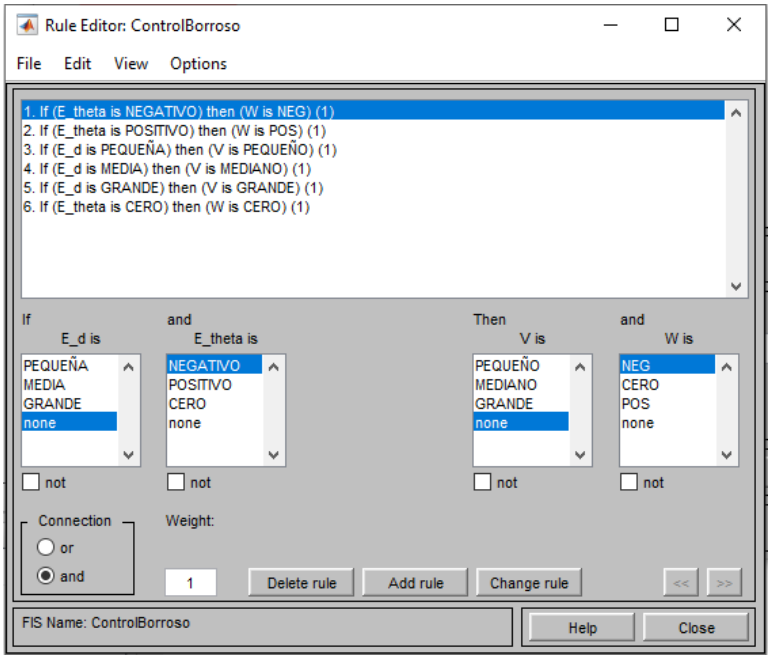
Se ha diseñado el siguiente controlador borroso, tal y como propone el enunciado:



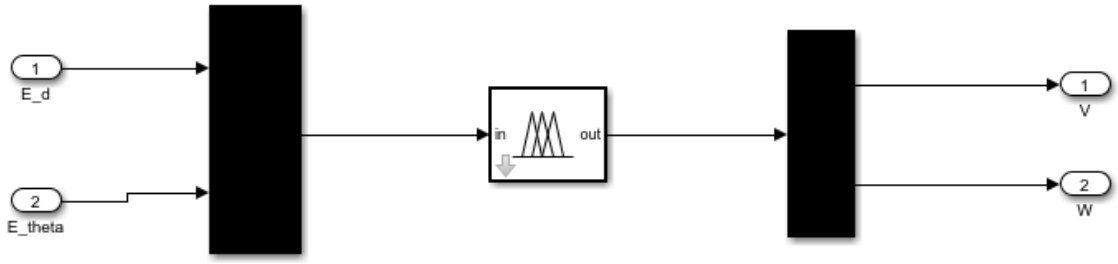
Además, se ha modificado la velocidad para que su rango vaya de 0 a 1, y el error de theta para poder corregir la trayectoria manteniendo una mayor precisión:

E_theta	V
	

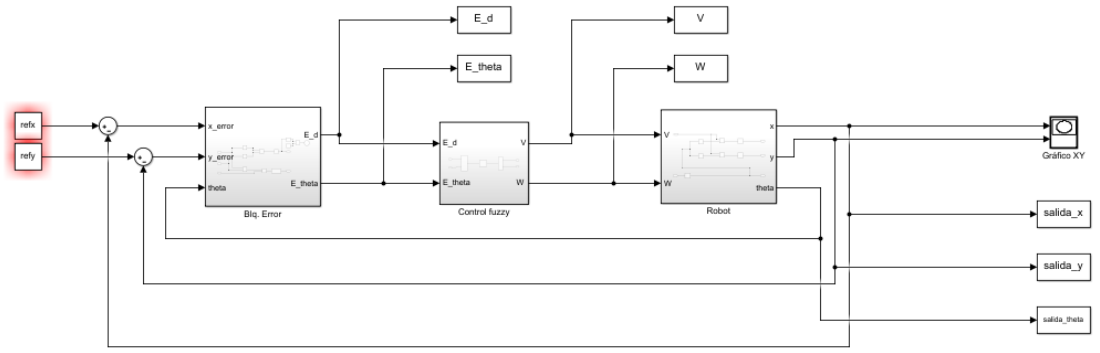
En el controlador, se encuentran definidas las siguientes reglas, que hemos separado en función de su ámbito, bien sea por distancia o ángulo:



Una vez guardado, se crea un nuevo proyecto en Simulink. Creamos un bloque que contiene el controlador borroso con las dos entradas y salidas:



Insertamos el bloque de control borroso el el archivo de control de posición de simulink:

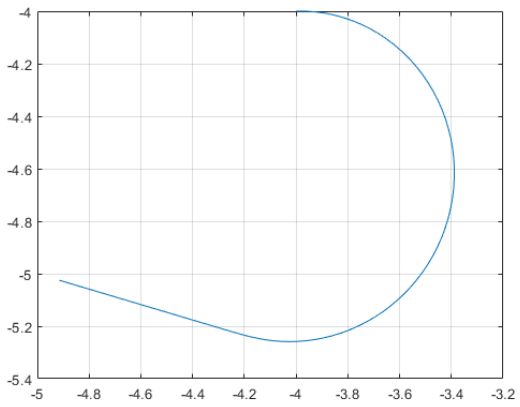


Y utilizamos el siguiente script para probar su funcionamiento:

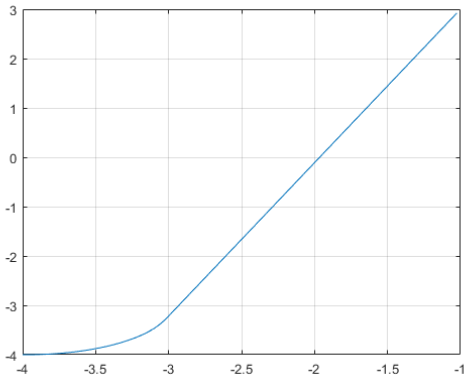
```
%Tiempo de muestreo
Ts=100e-3
% Referencia x-y de posicion
refx=5*rand-2.5
refy=5*rand-2.5

% Ejecutar Simulacion
sim('PositionControl.slx')
% Mostrar
x=salida_x.signals.values;
y=salida_y.signals.values;
figure;
plot(x,y);
grid on;
hold on;
```

Modificando los valores de refx y refy, podemos ir probando distintas situaciones para verificar que las reglas funcionan adecuadamente. Debemos tener en cuenta que el robot inicia el movimiento desde (-4, -4) y que si la distancia a recorrer es mayor que 10, obtendremos errores debido a salirse del dominio del controlador borroso.

Por ejemplo, vemos que para (-5, -5) consigue corregir la ruta adecuadamente:



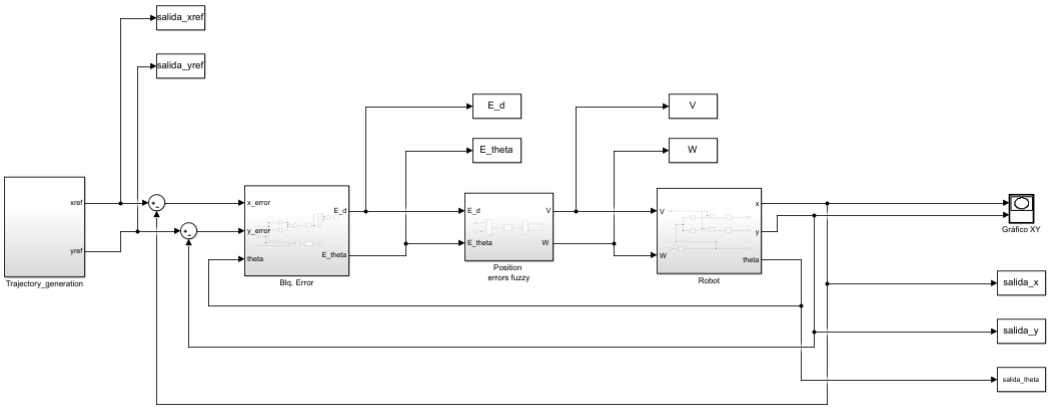
O que consigue corregir y avanzar en línea recta para llegar a (-1, 3):



EJERCICIO 2

En este ejercicio se propone emplear el controlador borroso diseñador para realizar la trayectoria del archivo de control de trayectoria de simulink.

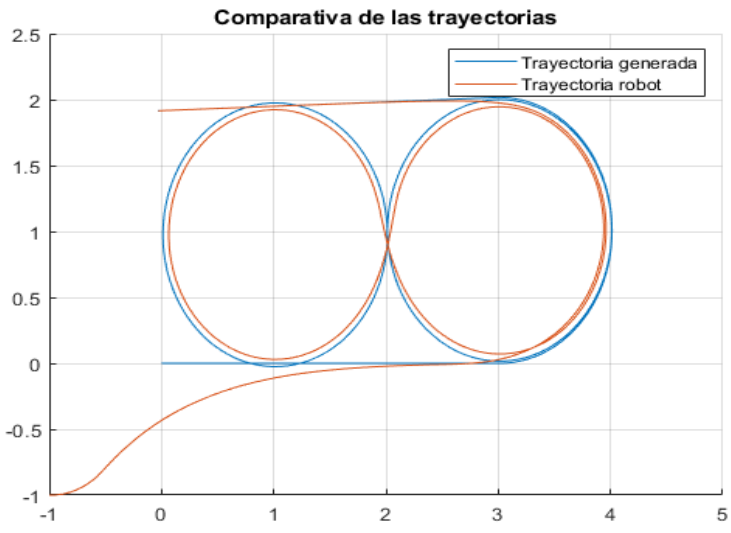
Sustituimos el bloque de control por el bloque de control borroso con nuestro controlador borroso:



Y empleamos el siguiente script para verificar que el robot sigue la trayectoria:

```
clear all; close all;

% Inicializamos las variables necesarias para el sistema
Ts = 0.1;

x_0 = 0;
y_0 = 0;
th_0 = 0;

% Hacemos la simulacion
sim('TrajectoryControl.slx');

% Recogemos los datos de trayectoria que envia el modelo desde simulink
trayectoria_x = salida_xref.signals.values';
trayectoria_y = salida_yref.signals.values';

x = salida_x.signals.values';
y = salida_y.signals.values';

% Pintamos ambas trayectorias en un figure
figure(1);
hold on;
tray_original = plot(trayectoria_x, trayectoria_y);
tray_robot = plot(x, y);
hold off;
grid on;
legend([tray_original tray_robot], {'Trayectoria generada', 'Trayectoria robot'});
title('Comparativa de las trayectorias');
```

Observamos que el robot consigue realizar la trayectoria con bastante precisión:

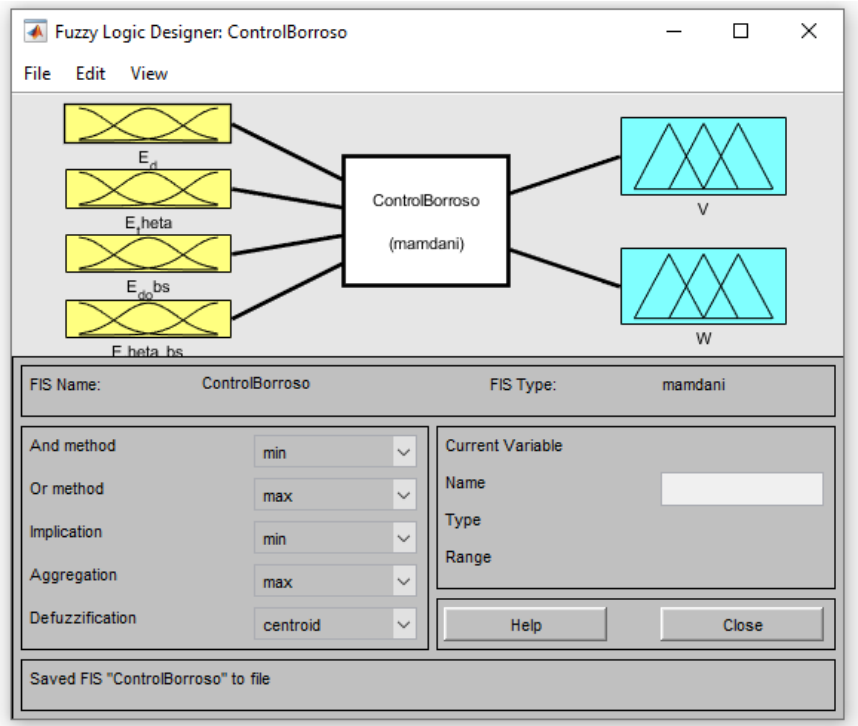


PARTE 2 – CONTROL BORROSO CON OBSTÁCULOS

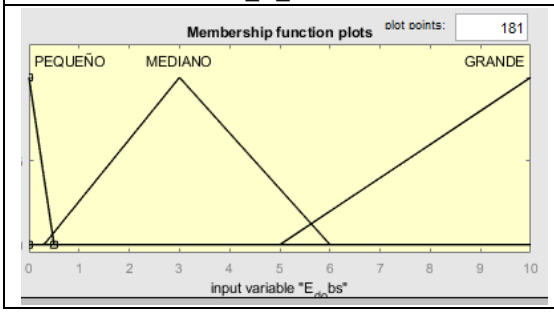
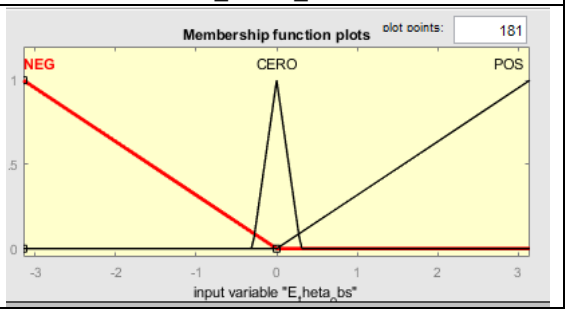
En esta parte se propone modificar el controlador borroso creado en la parte 1 para poder realizar trayectorias esquivando un obstáculo.

EJERCICIO 1

Partiendo del controlador de la parte 1, se ha diseñado el siguiente controlador borroso:

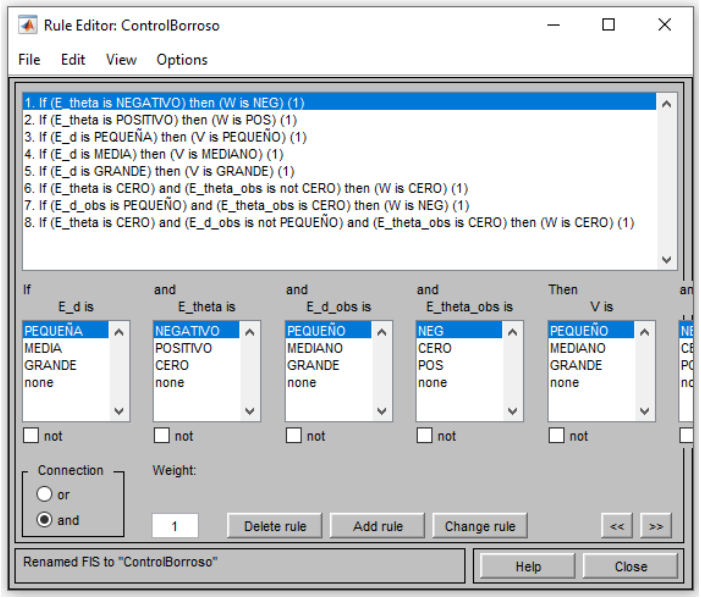


Se han diseñado las funciones de pertenencia del error de distancia y de ángulo del obstáculo:

E_d_obs	E_theta_obs
	

Se han establecido así las funciones de trayectoria para poder realizar la corrección lo más cerca posible del obstáculo, de forma que la trayectoria final se vea lo menos afectada posible por dicho obstáculo.

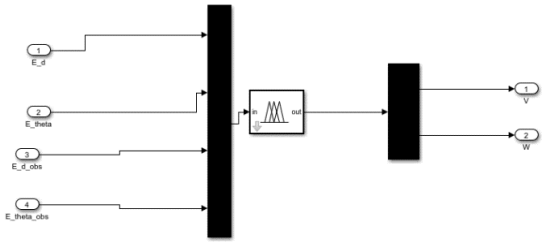
Contamos con el siguiente conjunto de reglas:



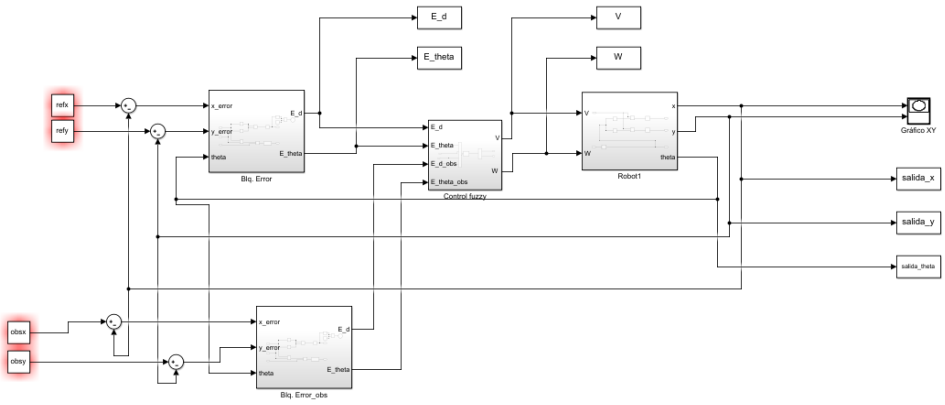
Como se puede observar, hemos añadido reglas para corregir la trayectoria respecto al obstáculo.

- La regla 7 implica que cuanto se esté al lado del objeto y vayamos directos a él, corrijamos la trayectoria con W negativo, es decir, girando a la izquierda.
- La regla 8 implica que aunque vayamos dirección al obstáculo, no nos molestemos en corregir hasta que estemos cerca.

Una vez más creamos el bloque de control borroso de la misma forma que en el apartado anterior:



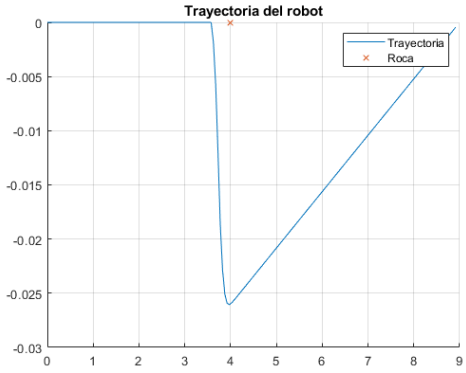
Lo añadimos al archivo simulink de control de posición:



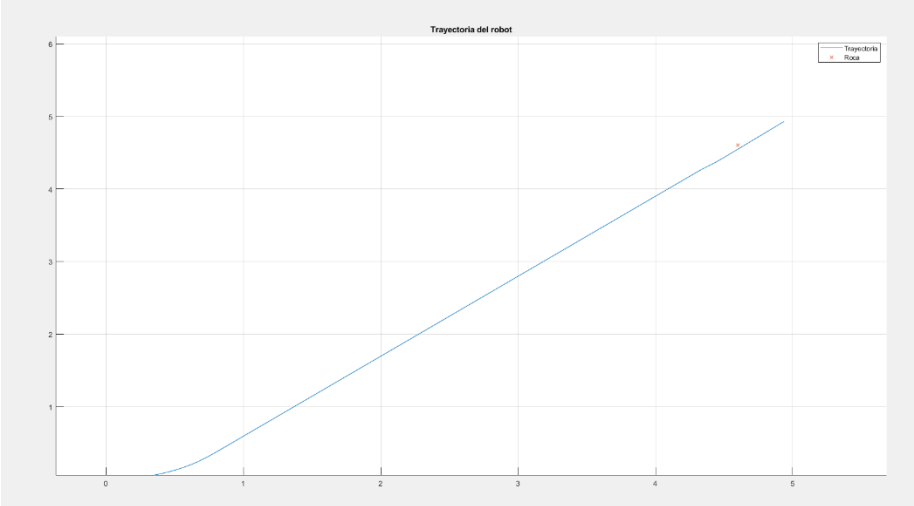
Contamos con el siguiente script para comprobar su funcionamiento:

```
clear all; close all;
%Tiempo de muestreo
Ts=100e-3
% Referencia x-y de posicion
refx=9
refy=0
obsx = 4
obsy = 0
% Ejecutar Simulacion
sim('PositionControl.slx')
% Mostrar
x=salida_x.signals.values;
y=salida_y.signals.values;
figure;
hold on;
tray_original = plot(x,y);
tray_roca = plot(obsx, obsy, 'x');
grid on;
hold off;
legend([tray_original tray_roca], {'Trayectoria', 'Roca'});
title('Trayectoria del robot');
```

Podemos ir jugando con el punto objetivo (refx, refy) y el punto donde se encuentra el obstáculo (obsx, obsy), para verificar su correcto funcionamiento. Para el objetivo (9,0) y el obstáculo en (4,0), vemos que consigue esquivarlo:

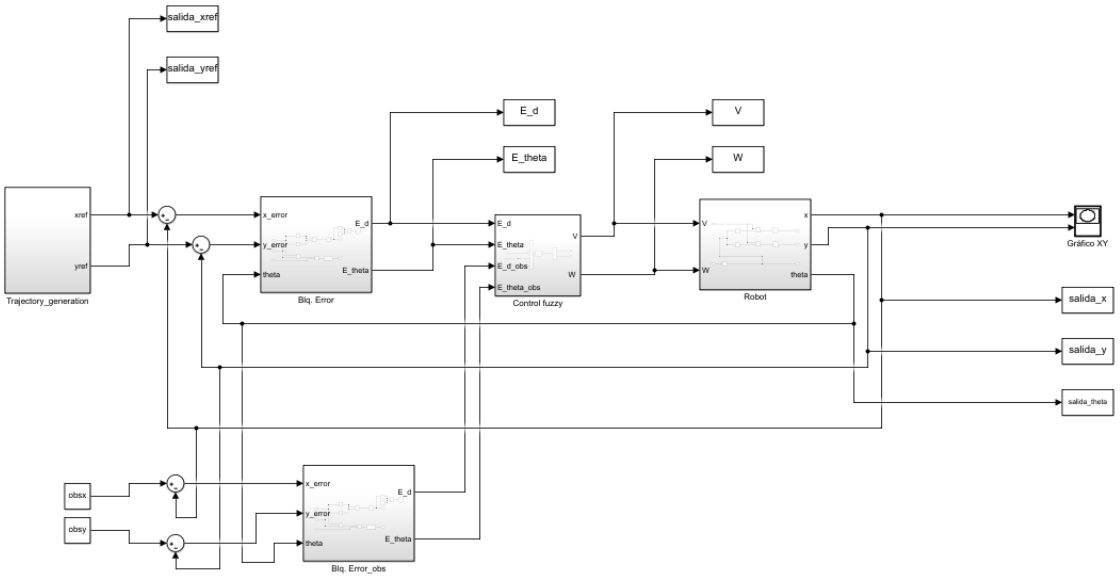


E incluso consigue esquivarlo con bastante precisión cuando el objetivo se encuentra en (5,5) y el obstáculo en (4'6, 4'6):



EJERCICIO 2

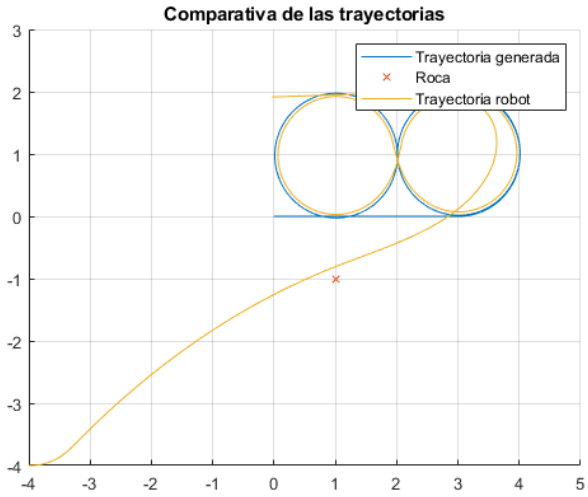
Implementamos el bloque de control borroso en el generador de trayectorias.



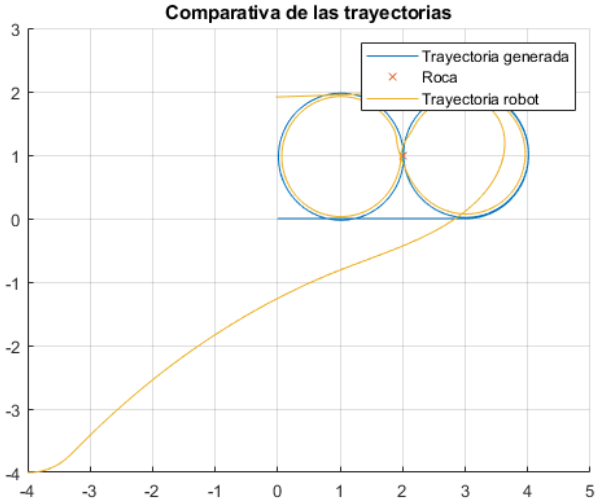
Utilizamos el siguiente script para comprobar el correcto funcionamiento:

```
clear all; close all;

% Inicializamos las variables necesarias para el sistema
Ts = 0.1;

x_0 = 0;
y_0 = 0;
th_0 = 0;

obsx = 1;
obsy = -1;

% Hacemos la simulacion
sim('TrajectoryControl.slx');

% Recogemos los datos de trayectoria que envia el modelo desde simulink
trayectoria_x = salida_xref.signals.values';
trayectoria_y = salida_yref.signals.values';

x = salida_x.signals.values';
y = salida_y.signals.values';

% Pintamos ambas trayectorias en un figure
figure(1);
hold on;
tray_original = plot(trayectoria_x, trayectoria_y);
tray_roca = plot(obsx, obsy, 'x');
tray_robot = plot(x, y);
hold off;
grid on;
legend([tray_original tray_roca tray_robot], {'Trayectoria generada','Roca', '
Trayectoria robot'});
title('Comparativa de las trayectorias');
```

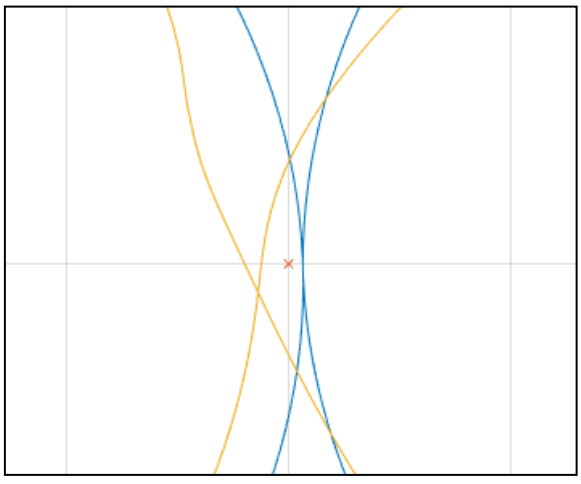
Y podemos probar a poner el obstáculo en medio de la trayectoria para ver que el robot lo esquiva adecuadamente. Para la posición (1, -1), vemos que lo esquiva:



Para la posición (2,1) podemos ver en el gráfico como consigue esquivarlo en ambas vueltas:



Si hacemos zoom, podemos ver que realmente lo esquiva. Si queremos que deje más espacio podemos jugar con el error de distancia ampliando el conjunto borroso que termina la distancia pequeña.



Finalmente, observamos que si ponemos el obstáculo en (1,2), también lo esquiva satisfactoriamente: