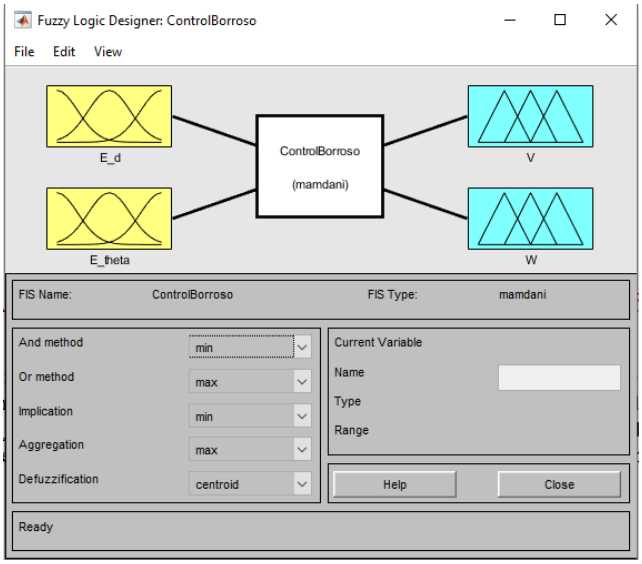
PRÁCTICA 2

Patricia Cuesta Ruiz

Víctor Gamonal Sánchez

CONTROL BORROSO (I): Diseño de un control borroso de posición para un robot móvil

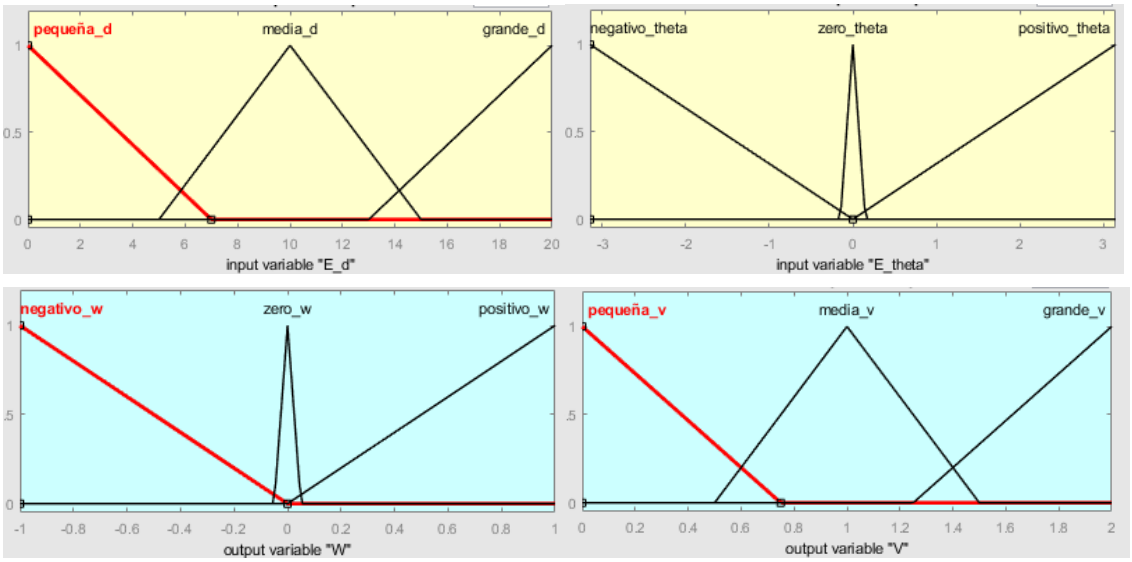
Para la realización de esta parte, hemos seguido los pasos indicados en el enunciado de la práctica. En primer lugar hemos creado un sistema borroso de tipo Mamdani con dos entradas (E_d para el error de la distancia y E_theta para el error del ángulo) y dos salidas (V para la velocidad lineal y W para la velocidad angular) como se muestra en la siguiente figura:



Como se puede apreciar y como también especifica el enunciado, hemos dejado los valores de **T-norma** (mínimo) y **S-norma** (máximo) por defecto, además del **desborrosificador** (centroide). Además, hemos establecido las variables de entrada para que dispongan de los siguientes rangos de valores:

- **E_d**: [0, 20]
- **E_theta**: [-π, π]
- **V**: [0, 2]
- **W**: [-1, 1]

Las funciones de pertenencia las hemos diseñado con la mayor lógica que hemos considerado posible, atribuyendo en este caso a las variables **E_d** y **V** los conjuntos borrosos *negativo*, *media* y *grande* y para las variables **E_theta** y **W** los conjuntos borrosos *negativo*, *zero* y *positivo* como se pueden ver a continuación:



Esto nos ha servido para posteriormente diseñar la tabla de reglas que hemos implementado de la siguiente manera:

E_theta \ E_d	**pequeña_d**	**media_d**	**grande_d**
negativo_theta	pequeña_v	mediana_v	grande_v
zero_theta	pequeña_v	mediana_v	grande_v
positivo_theta	pequeña_v	mediana_v	grande_v

*Tabla de reglas para la salida **V***

E_theta \ E_d	**pequeña_d**	**media_d**	**grande_d**
negativo_theta	negativo_w	negativo_w	negativo_w
zero_theta	zero_w	zero_w	zero_w
positivo_theta	positivo_w	positivo_w	positivo_w

*Tabla de reglas para la salida **W***

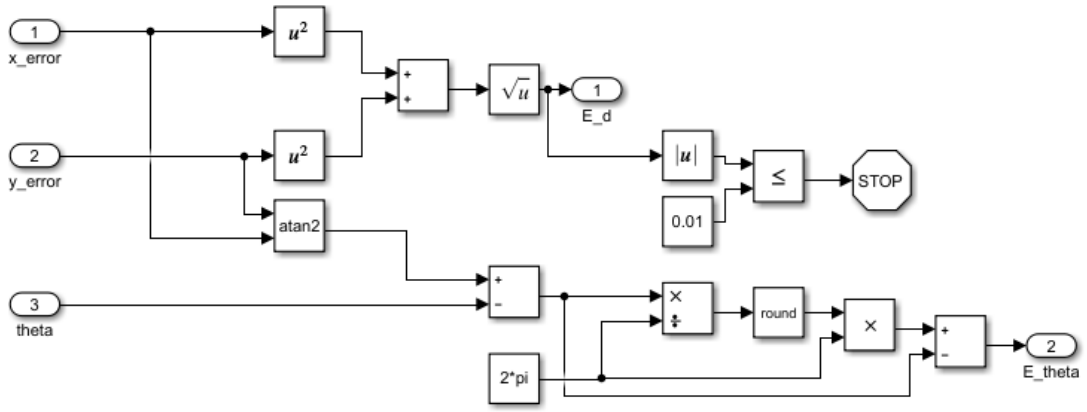
Cabe destacar que para establecer dichos conjuntos borrosos hemos probado distintos valores de *V* para regular las reglas de *W*, y viceversa, hasta encontrar los más adecuados.

Aquí se puede contemplar cada una de las reglas detalladas:

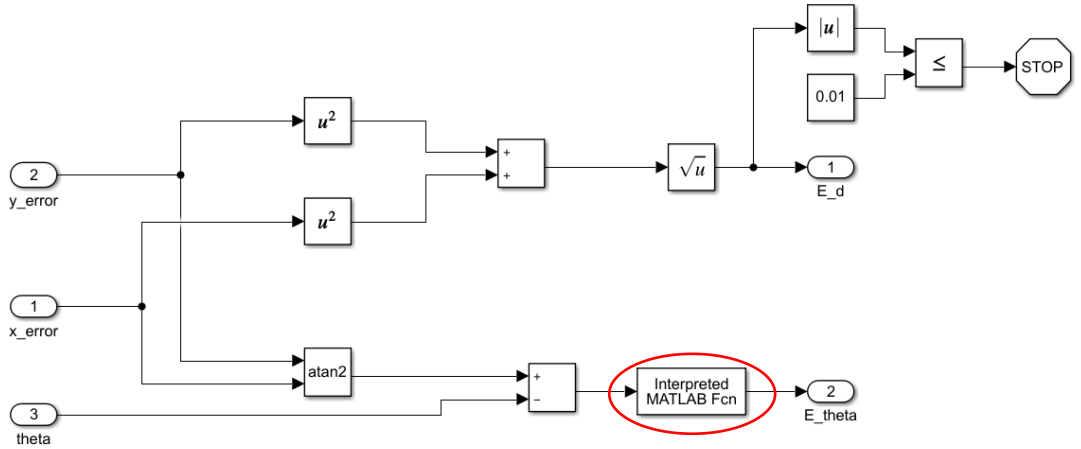
1. If (E_d is pequeña_d) then (V is pequeña_v)(W is zero_w) (1)
2. If (E_d is media_d) then (V is media_v)(W is zero_w) (1)
3. If (E_d is grande_d) then (V is grande_v)(W is zero_w) (1)
4. If (E_theta is negativo_theta) then (W is negativo_w) (1)
5. If (E_theta is positivo_theta) then (W is positivo_w) (1)
6. If (E_theta is zero_theta) then (W is zero_w) (1)

Además, en el subsistema *position errors* hemos realizado los siguientes cambios respecto a prácticas anteriores.

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



Original

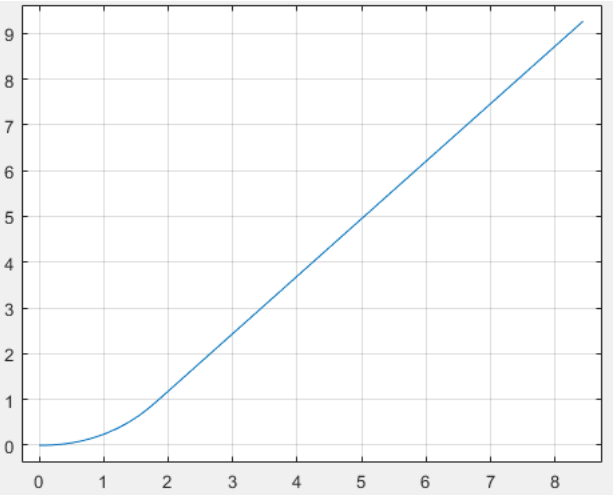


Nuevos cambios

Una vez creado el controlador borroso, como ya hemos visto, lo hemos sustituido en nuestro archivo *PositionControl.slx* por el antiguo controlador y hemos realizado distintas pruebas para verificar que el robot realiza una trayectoria de manera adecuada; en este caso, mostramos el resultado de este tras la ejecución del código facilitado en el enunciado de la práctica (para valores aleatorios de *refx* y *refy*).

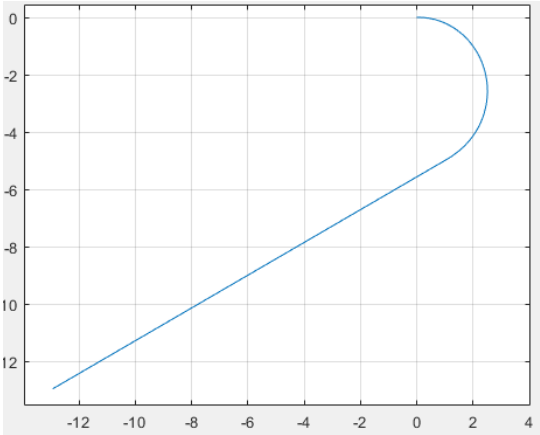
```
clc; clear all;

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



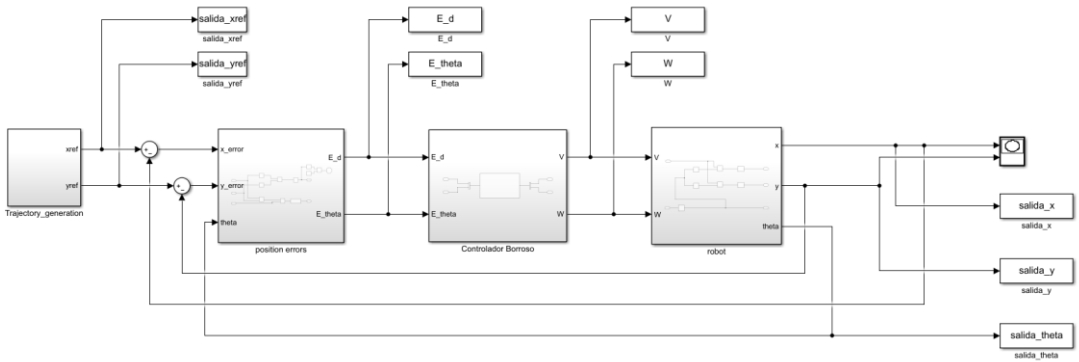
Además, vamos modificando los valores de refx y refy para ir comprobando distintas situaciones y ver que las reglas funcionan correctamente.

Por ejemplo, con los valores (-13, -13) la ruta sería:



Para la realización de la segunda parte de la práctica se propone implementar el controlador borroso diseñado en la parte anterior añadiendo ahora el bloque de control de trayectoria de la práctica anterior.

Al sustituirlo el sistema quedaría de la siguiente forma:



Y ejecutando el código del siguiente script:

```
clc; clear all;

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

x_0 = 0.02;
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('Comparación de trayectorias');
```

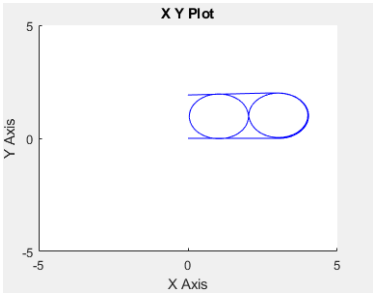
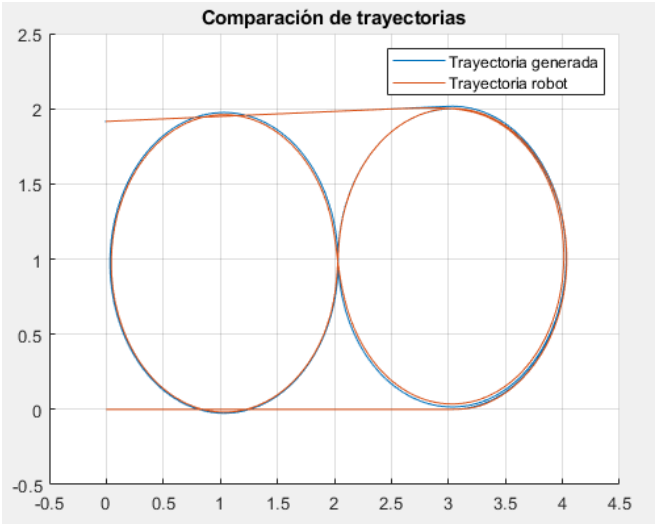


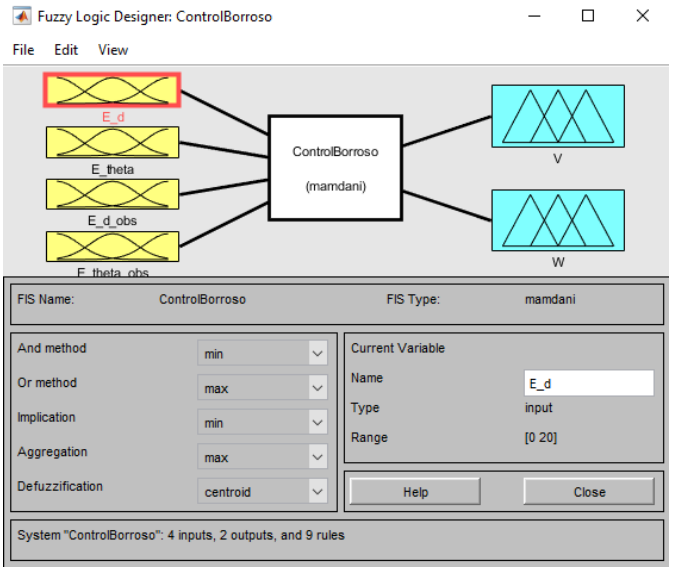
Gráfico de trayectoria realizada por el robot

La trayectoria del robot seguida por el robot comparada con la trayectoria generada sería:



CONTROL BORROSO (II): Diseño de control borroso de posición con evitación de obstáculos

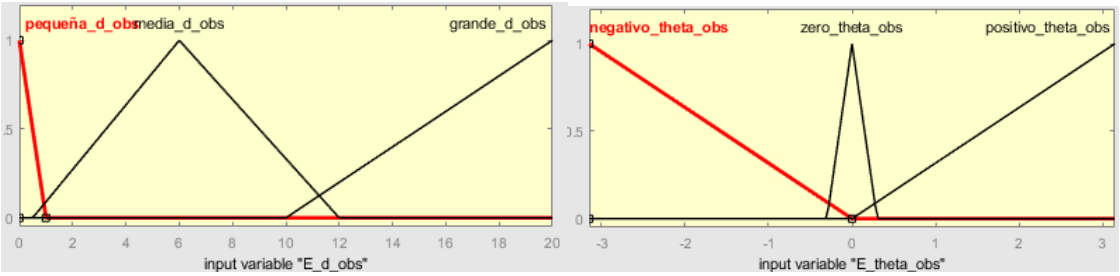
Para la realización de esta segunda parte de la práctica hemos tenido que añadir al sistema creado en la parte anterior dos nuevas entradas (*E_d_obs* y *E_theta_obs*) con lo que tendríamos así un sistema de tipo **Mamdani** de cuatro entradas y dos salidas.



Los valores de **T-norma** (mínimo) y **S-norma** (máximo) se mantienen, al igual que el **desborrosificador** (centroide). Además, para las nuevas entradas los rangos que les hemos asignado son:

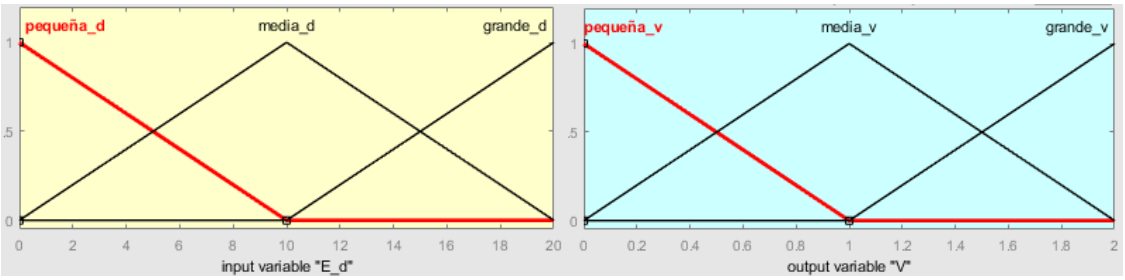
- **E_d_obs**: [0, 20]
- **E_theta_theta**: [-π, π]

En cuanto a las funciones de pertenencia de estas entradas, serían:



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

Además, hemos modificado las funciones de pertenencia de las variables *E_d* y *V* respecto a la parte 1 de la práctica para obtener un correcto funcionamiento del robot, quedando estas de la siguiente forma:



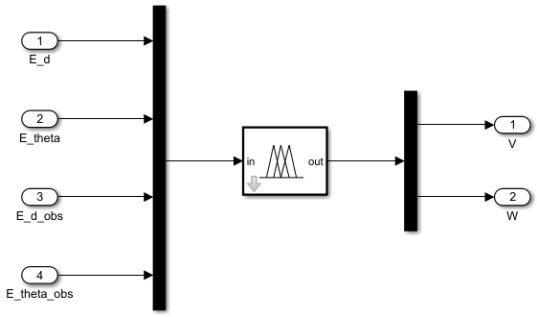
También hemos añadido nuevas reglas que serían:

1. If (E_d is pequeña_d) then (V is pequeña_v) (1)
2. If (E_d is media_d) then (V is media_v) (1)
3. If (E_d is grande_d) then (V is grande_v) (1)
4. If (E_theta is negativo_theta) then (W is negativo_w) (1)
5. If (E_theta is positivo_theta) then (W is positivo_w) (1)
6. If (E_theta is zero_theta) then (W is zero_w) (1)
7. If (E_theta is zero_theta) and (E_theta_obs is not zero_theta_obs) then (W is zero_w) (1)
8. If (E_d_obs is pequeña_d_obs) and (E_theta_obs is zero_theta_obs) then (W is negativo_w) (1)
9. If (E_theta is zero_theta) and (E_d_obs is not pequeña_d_obs) and (E_theta_obs is zero_theta_obs) then (W is zero_w) (1)

Como se puede observar, hemos añadido reglas para corregir la trayectoria:

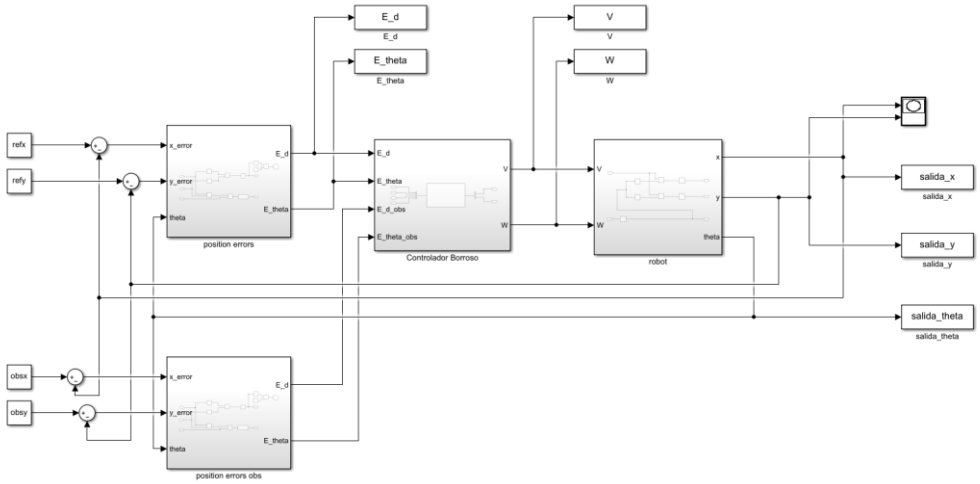
- La **regla 8** implica que cuando esté al lado del objeto y vaya directo a él se corrija la trayectoria con W negativo, es decir, girando a la izquierda.
- La **regla 9** implica que aunque vayamos en dirección al obstáculo, el robot no la corrija hasta que no esté cerca.

En cuanto al diseño del sistema en **Simulink**, tal y como especifica el enunciado de la práctica, también hemos realizado las modificaciones pertinentes en lo que al controlador se refiere:



Bloque del nuevo controlador borroso

Quedando de esta manera el sistema habiendo añadido también el bloque de *position errors* para el obstáculo:



Una vez realizados todos estos cambios, procedemos a **comprobar el funcionamiento de nuestro robot**; esto lo conseguimos haciendo uso del código facilitado en el enunciado de la práctica y modificando los distintos valores para las posiciones *refx* y *obs_y* para ver cómo este es capaz de evitar el obstáculo. Estos son los resultados:

```
clear all; clc;

%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;
hold on;
tray_original = plot(x,y);
tray_obs = plot(obsx, obsy, 'x');
grid on;
hold off;
legend([tray_original tray_obs], {'Trayectoria', 'Obstáculo'});
title('Trayectoria del robot');
```

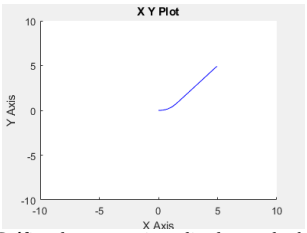


Gráfico de trayectoria realizada por el robot

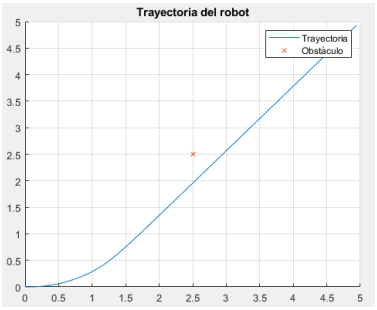


Gráfico de trayectoria del robot evitando el obstáculo

Y así quedaría modificando los valores por unos aleatorios:

```
clear all; clc;

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

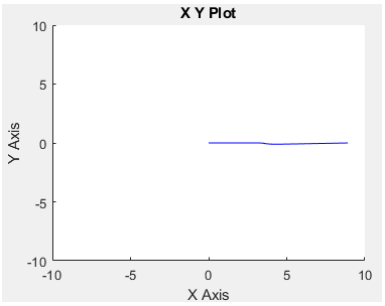


Gráfico de trayectoria realizada por el robot

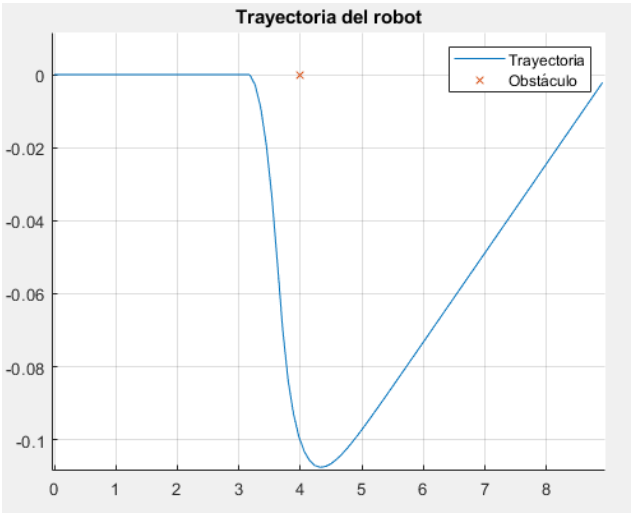
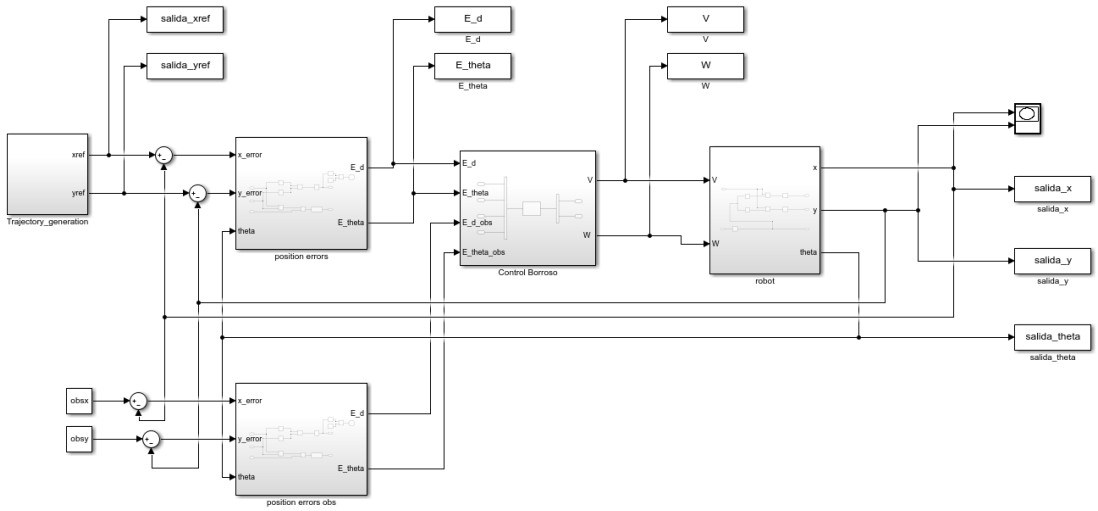


Gráfico de trayectoria del robot evitando el obstáculo

Por último, tan sólo nos queda comprobar el funcionamiento de nuestro robot implementando el bloque de **generador de trayectorias** que ya conocemos, quedando el sistema como se muestra a continuación:



Para ello hemos utilizado el siguiente código en un script de MATLAB en el que hemos realizado distintas pruebas variando los distintos puntos en los que se encuentra el obstáculo; empezamos, por ejemplo, para el **punto (3,2)**:

```
clear all; clc;

% Inicializamos las variables necesarias para el sistema
Ts = 100e-3;

x_0 = 0.02;
y_0 = 0;
th_0 = 0;

obsx = 3;
obsy = 2;

% 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_obstaculo = plot(obsx, obsy, 'x');
tray_robot = plot(x, y);
hold off;
grid on;
legend([tray_original tray_obstaculo tray_robot], {'Trayectoria generada', 'Obstáculo', 'Trayectoria robot'});
title('Comparación de trayectorias');
```

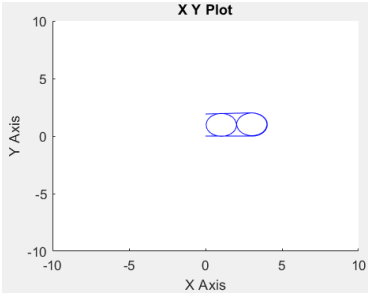
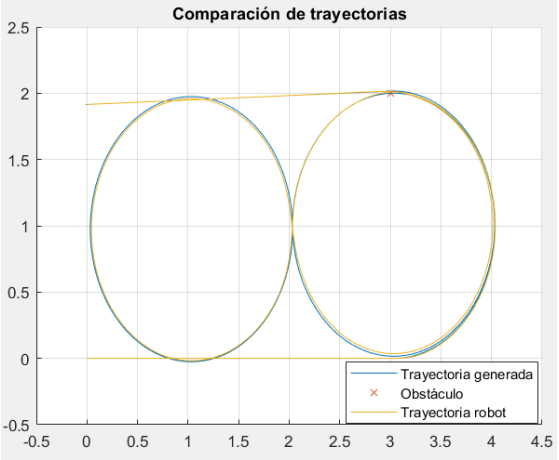
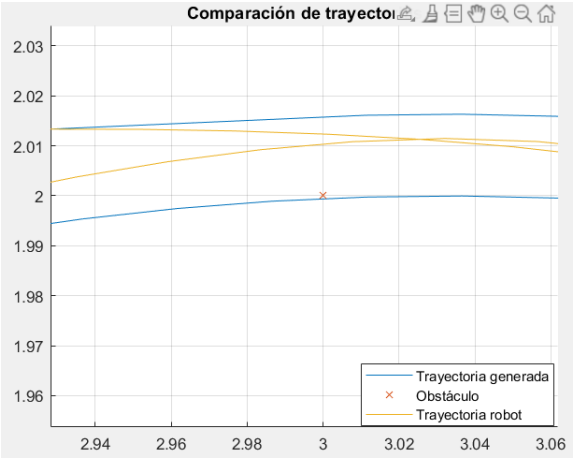


Gráfico de trayectoria realizada por el robot

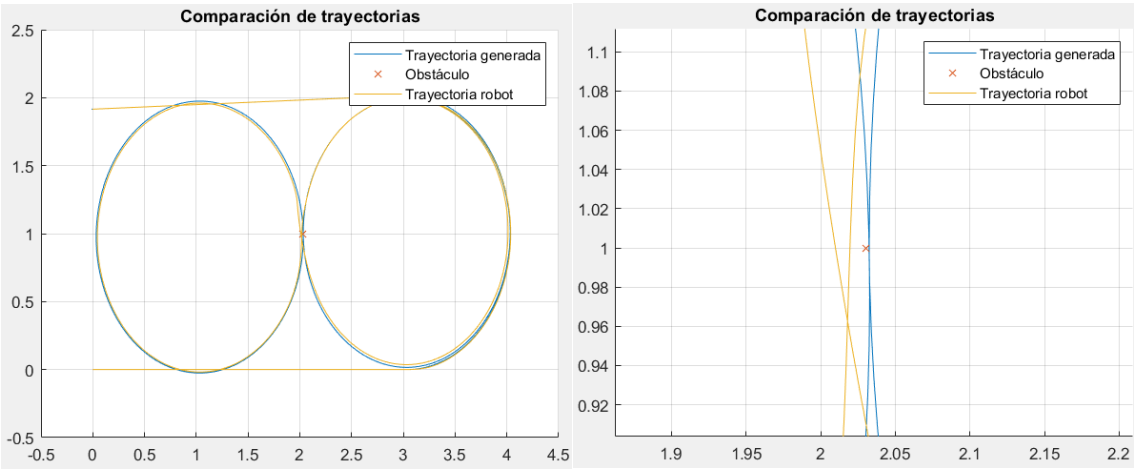
Como se puede observar a continuación, el recorrido es inmejorable y se puede ver mediante zoom que evita el obstáculo satisfactoriamente.





Probemos esta vez con algo más difícil, ¿será capaz nuestro robot de esquivar el obstáculo en el centro del recorrido? Vamos a comprobarlo.

Establecemos el obstáculo, en este caso, en el **punto (2,1).**



Como era de esperar, el robot evita el obstáculo tanto a la ida como a la vuelta, por lo que podemos decir que el resultado es muy satisfactorio.

Cabe añadir que si queremos dejar más espacio entre el recorrido del robot y el obstáculo, es tan sencillo como jugar con los valores del error de distancia ampliando el conjunto borroso que termina la distancia pequeña.