Universidad de Alcalá

Escuela Politécnica Superior

Universidad de Alcalá

Práctica Final

Controladores Borrosos y Neuroborrosos para Robot Móvil

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

ÍNDICE

CONSIDERACIONES PREVIAS

Es importante reseñar que la práctica se ha realizado empleando VMWare Workstation 15 Player y Matlab 2019b.

PARTE 1 – CONTROL BORROSO

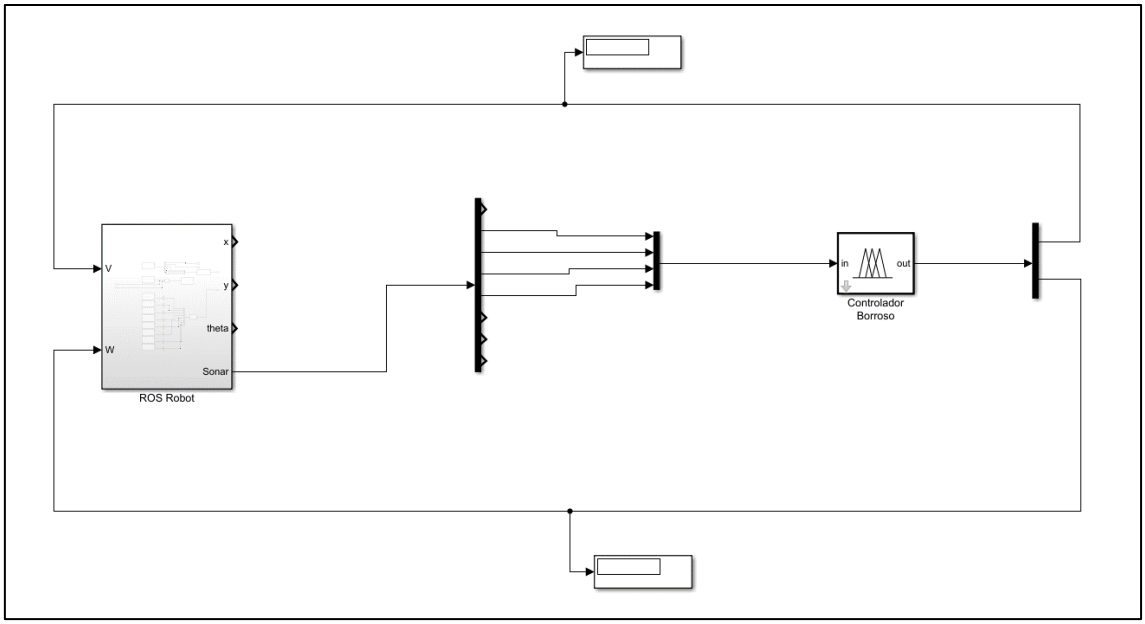
Para el desarrollo de este controlador se ha creado un modelo en Simulink sin contraparte en Matlab.

Es importante que el sondeo de los sonares del robot se haga, al menos, cada 0.2 segundos relativos a la simulación (constante TimeStep). En el archivo .slx que se entrega está configurada la simulación con el TimeStep definido como 0.1, con 100 segundos de tiempo máximo de simulación, y con una relación de tiempo de simulación 1:1. Si se reduce el "pacing" de la simulación, es importante cambiar el TimeStep también, para que se haga la lectura de los sensores correctamente y el robot pueda corregir su trayectoria.

MODELO SIMULINK

El modelo cuenta con el bloque de robot proporcionado por los profesores, el bloque con el controlador borroso para regular la trayectoria del robot, y los multiplexores/demultiplexores necesarios para hacer las conexiones.

El esquema general del modelo es el siguiente:

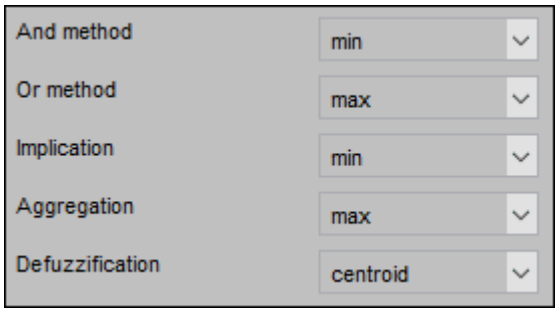


Se han añadido dos displays que nos permiten ver las variables de velocidad lineal y velocidad angular que se le pasan al robot como entradas.

Como se puede apreciar por el multiplexor, únicamente se han hecho uso de los sensores 1, 2, 3, y 4 del robot para la corrección de su trayectoria. El resto de las salidas del bloque del robot no se utilizan.

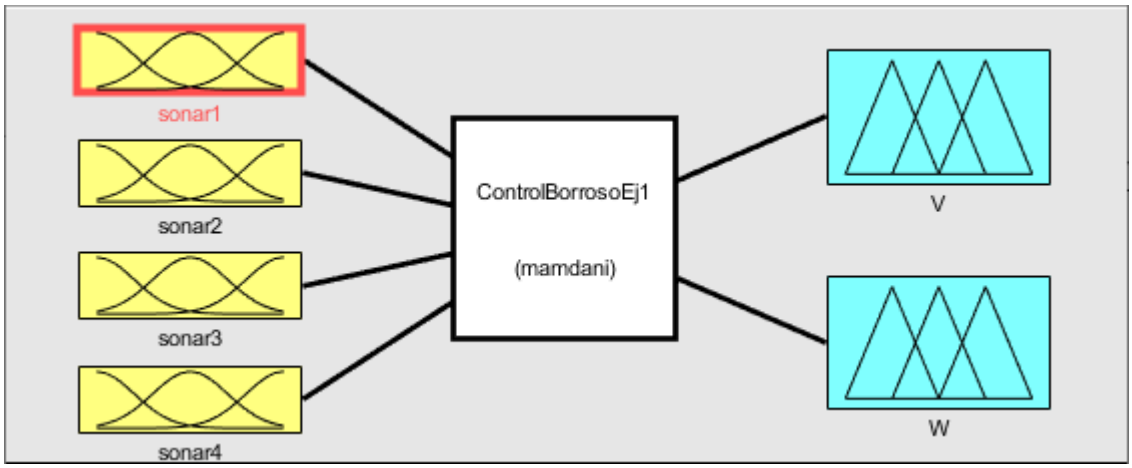
CONFIGURACIÓN DEL CONTROLADOR

En cuanto al controlador, se usa la siguiente configuración en el proceso de borrosificación, desborrosificación, y las normas a la hora de hacer inferencia:

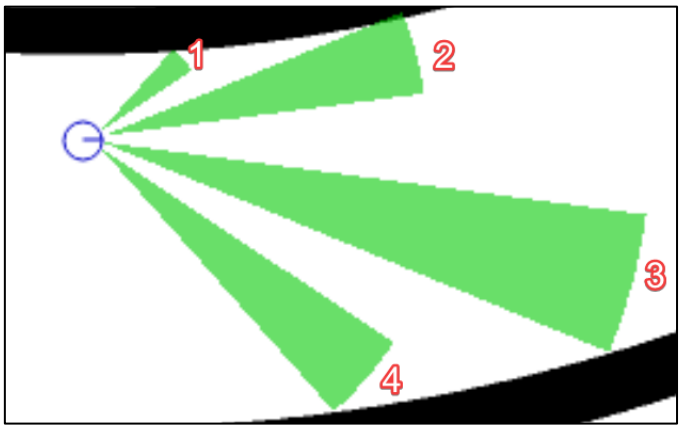


Se ha utilizado esta configuración ya que es la que viene por defecto y se obtuvieron buenos resultados en las prácticas anteriores.

Como podemos observar en el modelo de Simulink, el controlador borroso cuenta con 4 entradas, una por cada sensor. El controlador tiene 2 salidas: la velocidad lineal, y la velocidad angular que el robot debe tomar para seguir con la trayectoria de forma correcta. Este es el esquema del controlador:



Haciendo que sólo los sensores que se usan del robot sean visibles, podemos identificarlos más fácilmente, lo cual nos ayuda a la hora de crear las reglas y estudiar el comportamiento del robot. En nuestro caso, los sensores utilizados son los siguientes:



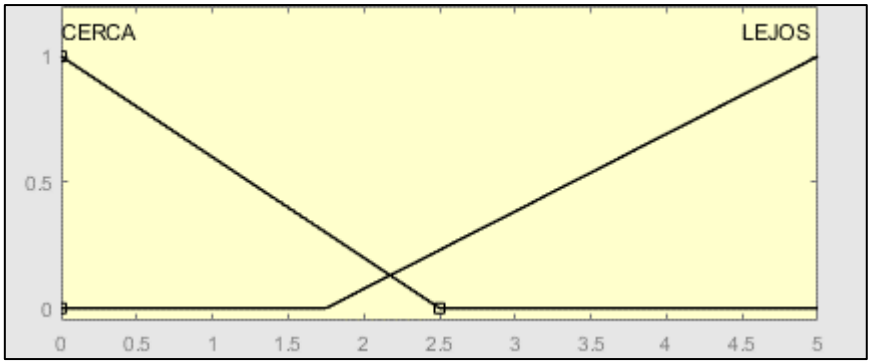
Como podemos apreciar, la disposición de los sensores del robot es simétrica. Esto es, los sensores 1 y 4, así como los sensores 2 y 3, se abren con el mismo ángulo, pero con signo contrario. Esta propiedad ha tenido mucha importancia a la hora de definir los conjuntos borrosos de cada uno de estos inputs, así como a la hora de generar las reglas de inferencia borrosa.

De cara a la definición de los rangos, se tiene lo siguiente:

- **Inputs (sonar1, sonar2, sonar3, sonar4)**: rango comprendido en [0, 5]. Se ha escogido 5 ya que es el rango máximo que pueden leer los sonares del robot.
- **Output 1 (V)**: rango comprendido en [0, 5]. Tras realizar varias pruebas, se llego a la conclusión que 5 era una velocidad máxima aceptable para el desarrollo del controlador. Puesto que el controlador está pensado para que la velocidad máxima sea 5 y, sobre todo, para este circuito en concreto, si se incrementa la velocidad, es muy probable que el robot falle.
- **Output 2 (W)**: se ha establecido que el rango de giro del robot se mueva en el intervalo [-pi, pi], siguiendo los valores escogidos en prácticas anteriores.

En cuanto al tratamiento de cada uno de estos inputs, se han utilizado funciones trimf para definir los conjuntos borrosos en todos los casos.

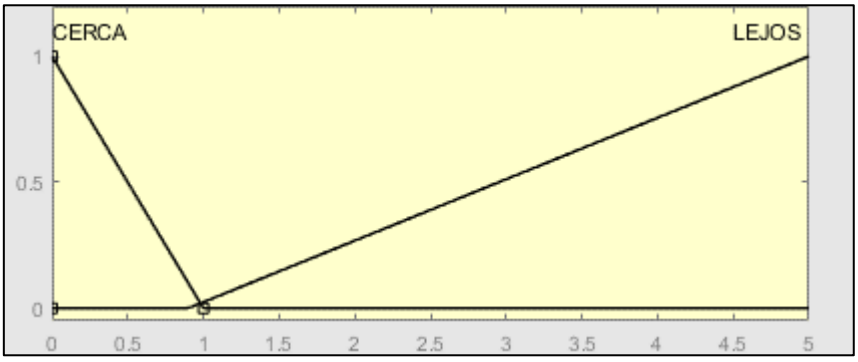
Debido a la propiedad de simetría, los sensores 1 y 4 cuentan con la misma configuración para sus conjuntos borrosos. Esta configuración es la siguiente:



Los parámetros de estos conjuntos son:

- **CERCA:** [0 0 2.5]
- **LEJOS:** [1.75 5 5]

La configuración de los sensores 2 y 3 es la siguiente:

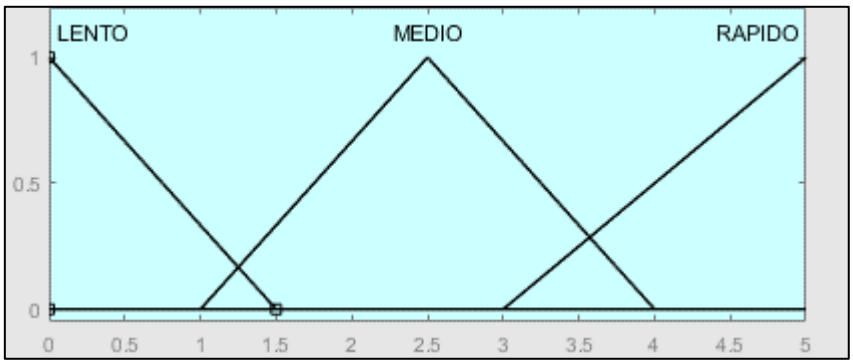


Los parámetros de estos conjuntos son:

- **CERCA:** [0 0 1]
- **LEJOS:** [0.9 5 5]

Para los outputs, tenemos los siguientes conjuntos:

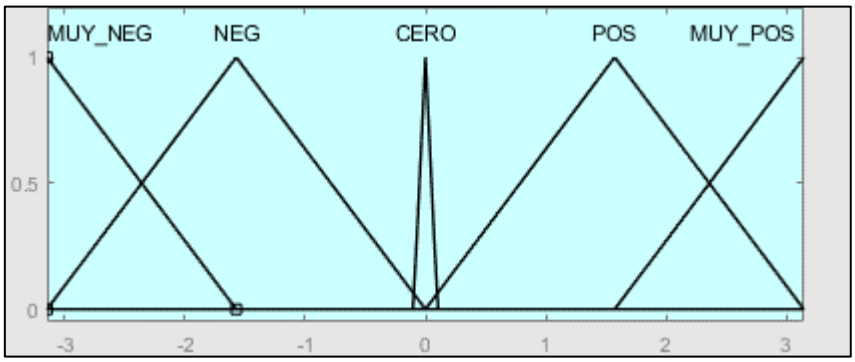
La velocidad lineal está definida por los siguientes conjuntos:



Los parámetros de estos conjuntos son:

- **LENTO:** [0 0 1.5]
- **MEDIO:** [1 2.5 4]
- **RAPIDO:** [3 5 5]

En cuanto a la velocidad angular:



Los parámetros de estos conjuntos son:

- **MUY_NEG:** [-3.142 -3.142 -1.571]
- **NEG:** [-3.142 -1.571 0]
- **CERO:** [-0.1 0 0.1]
- **POS:** [0 1.571 3.142]
- **MUY_POS:** [1.571 3.142 3.142]

Los parámetros para estos conjuntos se han obtenido de forma experimental tras muchas pruebas. En un primer momento se tenían conjuntos intermedios que representaban distancias medias, pero acabaron por ser desestimados ya que no encajaban bien a la hora de definir reglas de inferencia.

REGLAS DE INFERENCIA

En cuanto a las reglas de inferencia borrosa, se han definido un total de 9 reglas. Estas reglas son las siguientes:

1. If (sonar1 is CERCA) and (sonar3 is LEJOS) then (W is NEG) (1)
2. If (sonar2 is LEJOS) and (sonar4 is CERCA) then (W is POS) (1)
3. If (sonar2 is CERCA) and (sonar3 is not CERCA) then (W is MUY_NEG) (1)
4. If (sonar2 is not CERCA) and (sonar3 is CERCA) then (W is MUY_POS) (1)
5. If (sonar1 is LEJOS) and (sonar2 is LEJOS) and (sonar3 is LEJOS) and (sonar4 is LEJOS) then (W is CERO) (1)
6. If (sonar2 is LEJOS) and (sonar3 is LEJOS) then (V is RAPIDO) (1)
7. If (sonar2 is LEJOS) and (sonar3 is not LEJOS) then (V is MEDIO) (1)
8. If (sonar2 is not LEJOS) and (sonar3 is LEJOS) then (V is MEDIO) (1)
9. If (sonar2 is CERCA) and (sonar3 is CERCA) then (V is LENTO) (1)

Con la siguiente explicación para cada regla:

1. Si el sonar1 (externo izquierda) está cerca y el sonar3 (interno derecha) no está cerca, es decir, no hay peligro de choque inmediato, hay un giro constante con W negativo.
2. Contraparte a la regla 1, exactamente lo mismo, pero en sentido contrario. Usa los sensores sonar4 (externo derecha) y sonar2 (interno izquierda).
3. En el caso de que el sonar 2 detecte un choque inminente (obstáculo en la mayoría de los casos), la regla 2 deja de aplicarse y hace que esta regla tenga mucho más peso. En el caso de que esta regla se satisfaga, se produce un giro fuerte con W negativo.
4. Contraparte a la regla 3, mismo funcionamiento, pero en sentido contrario. Para este tipo de giro, necesita que el sensor 3 no detecte un choque inminente también, porque si no, al cumplirse las reglas 3 y 4 a la vez, no intentaría girar y se estrellaría contra cualquier obstáculo que ambos sensores detectasen como próximo.
5. Si todos los sonares no registran obstáculos cerca, el robot puede avanzar en línea recta.
6. Si los sonares internos no detectan ningún obstáculo cerca, la velocidad puede incrementarse al máximo.
7. En el caso de que sólo uno de los sensores internos registre que no hay obstáculos cerca, la velocidad se puede aumentar a un punto intermedio.
8. Contraparte a la regla 7.
9. En el caso de que los dos sensores internos detecten un obstáculo cerca, se reduce la velocidad para facilitar el giro y evitar el impacto.

Con esta configuración, se ha conseguido que el robot haga una trayectoria bastante suave en la cual intenta avanzar por el centro del carril, solo moviéndose del centro cuando encuentra obstáculos.

Cabe destacar que el robot está pensado específicamente para este circuito y con estas condiciones de obstáculos. Si se saca de estas condiciones, es muy probable que el robot no funcione correctamente.

Se han hecho bastantes pruebas, la más exhaustiva siendo que el robot recorriese este circuito con el número máximo de obstáculos durante 300 segundos satisfactoriamente sin chocarse. Se ha decidido grabar un video mostrándolo que se puede ver aquí:

- Sin obstáculos: https://www.youtube.com/watch?v=1rVMNv2kESk
- Con obstáculos: https://www.youtube.com/watch?v=-xLAxGIwTD0

PARTE 2 – CONTROL NEUROBORROSO

Para el diseño del controlador neuroborroso se debe, en primer lugar, generar datos para poder entrenar la red neuronal. Estos datos se recogen del script de control manual proporcionado, que almacena cada 0.1 segundos los datos de los sensores, posiciones, velocidad angular y lineal.

Se han recorrido los circuitos sin y con obstáculos a mano unas 4-5 veces, lo suficiente para generar más de 1500 filas, y se ha guardado la matriz de datos en archivos distintos para cada circuito.

Una vez recogidos, se ha empleado la herramienta AnfisEdit para generar y entrenar un controlador borroso para la velocidad angular y otro para la lineal, ya que la herramienta utilizada no permite generar controladores con más de una salida.

Tras esto, se cargan los controladores en el archivo simulink, se ajustan las líneas de entradas y salidas, y se puede simular el comportamiento del robot sobre la máquina virtual.

ENTRENAMIENTO SIN OBSTÁCULOS

Para el circuito sin obstáculos se ha generado el siguiente script para cargar los datos de entrenamiento.

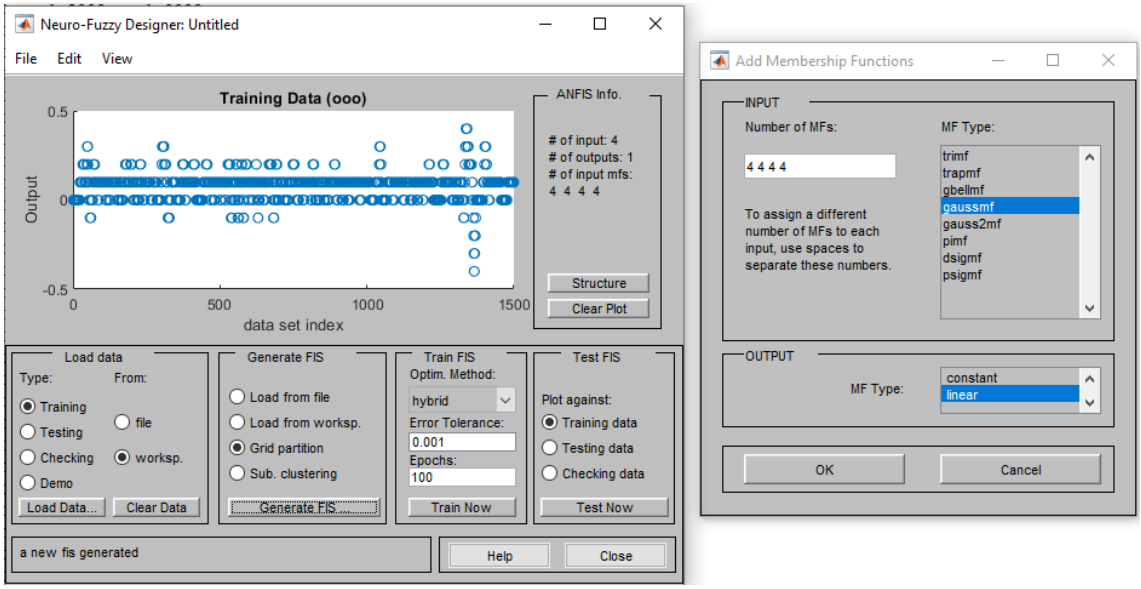
Para el entrenamiento del controlador angular, se emplean los sónares 0, 1, 4, y 5, es decir, los laterales, y los frontales exteriores.

Para el entrenamiento del controlador lineal, se emplean los sónares 1, 2, 3 y 4, es decir, todos los frontales.

```
clear all;
close all;

% Entrenamiento sin obstáculos
load datos_entrenamiento_sin_obs training

% Entrenamiento angular
train_angular = training(:,[1,6,7,8,12])
indices_angular = round(linspace(1,size(training,1),1500))
train_angular = train_angular(indices_angular,:)
train_angular(isinf(train_angular)) = 5.0
train_angular = double(train_angular)

% Entrenamiento lineal:
train_lineal = training(:,[2,3,4,5,13])
indices_lineal = round(linspace(1,size(training,1),1500))
train_lineal = train_lineal(indices_lineal,:)
train_lineal(isinf(train_lineal)) = 5.0
train_lineal = double(train_lineal)
```

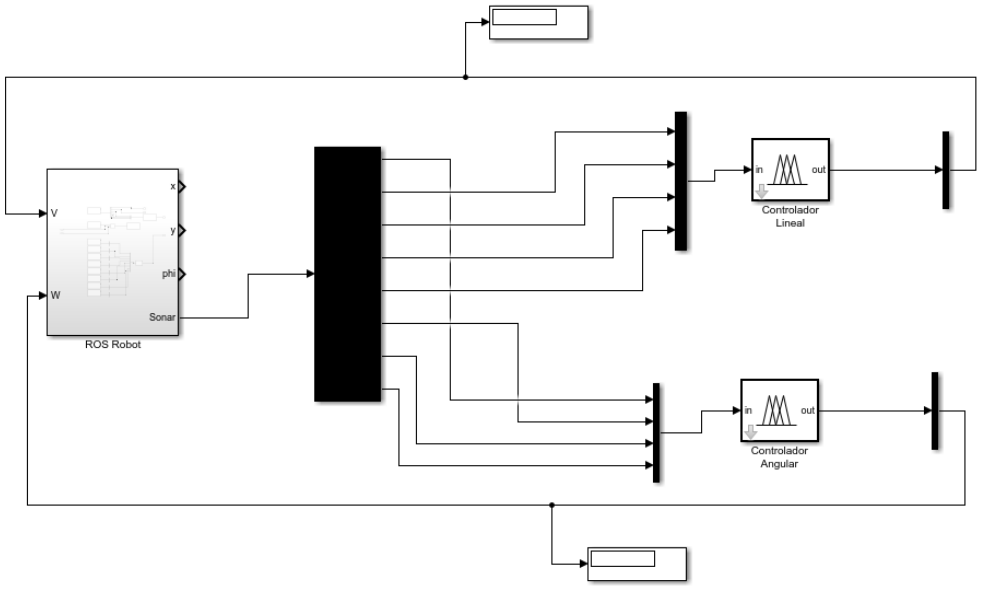
Tras la carga de datos, se lanza la herramienta AnfisEdit desde la consola de comandos, se cargan los datos de entrenamiento lineal y angular desde el entorno de trabajo, y se pueden ajustar los parámetros de entrenamiento de la red.

Tras estudiar, a base de prueba y error, qué ajustes generaban la menor tasa de error, se dejó entrenando durante una noche la velocidad angular y lineal con la siguiente configuración:



Se observa que, a mayor cantidad de conjuntos de pertenencia de las funciones, mayor precisión se tiene y, también, mayor memoria y tiempo entrenando se necesita. Con esta configuración, conseguimos reducir la cota de error por debajo del 0'03.

Se exportan los controladores a archivos *fis* para cargarlos desde simulink, quedando el siguiente diagrama:



Tras configurar la conexión, observamos que el robot consigue completar el circuito. Hemos decidido grabar un vídeo demostrándolo, que se puede ver aquí:

https://www.youtube.com/watch?v=VvFG90GlHE8

ENTRENAMIENTO CON OBSTÁCULOS

Para el circuito con muchos obstáculos, se ha generado el siguiente script.

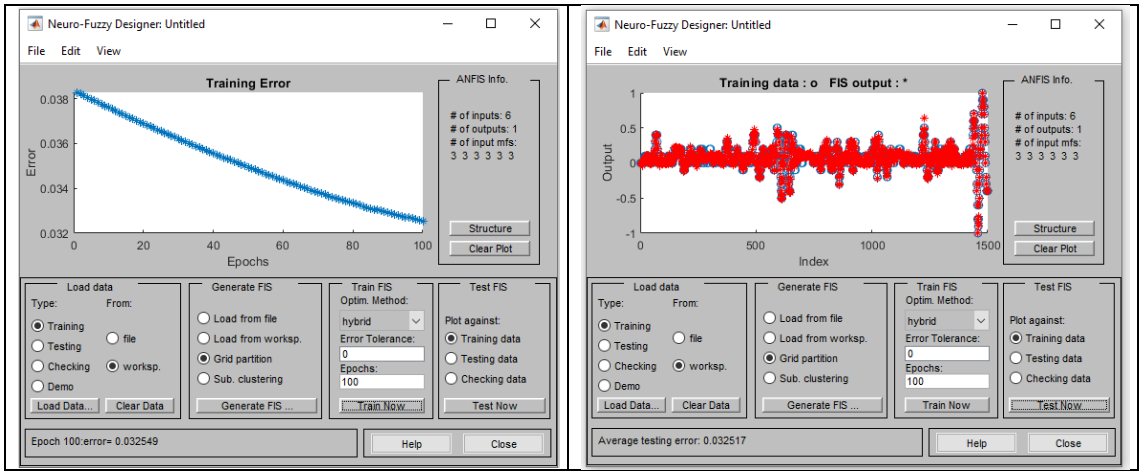
Para el entrenamiento de la velocidad angular, se emplean los sensores 0, 1, 2, 3, 4 y 5, es decir, todos los sensores frontales y los laterales.

Para el entrenamiento de la velocidad lineal, se utilizan los sónares 1, 2, 3 y 4, es decir, únicamente los 4 frontales.

```
clear all;
close all;

% Entrenamiento con obstáculos
load datos_entrenamiento_con_obs training

% Entrenamiento angular
train_angular = training(:,[1,2,3,4,5,6,12])
indices_angular = round(linspace(1,size(training,1),1500))
train_angular = train_angular(indices_angular,:)
train_angular(isinf(train_angular)) = 5.0
train_angular = double(train_angular)

% Entrenamiento lineal:
train_lineal = training(:,[2,3,4,5,13])
indices_lineal = round(linspace(1,size(training,1),1500))
train_lineal = train_lineal(indices_lineal,:)
train_lineal(isinf(train_lineal)) = 5.0
train_lineal = double(train_lineal)
```

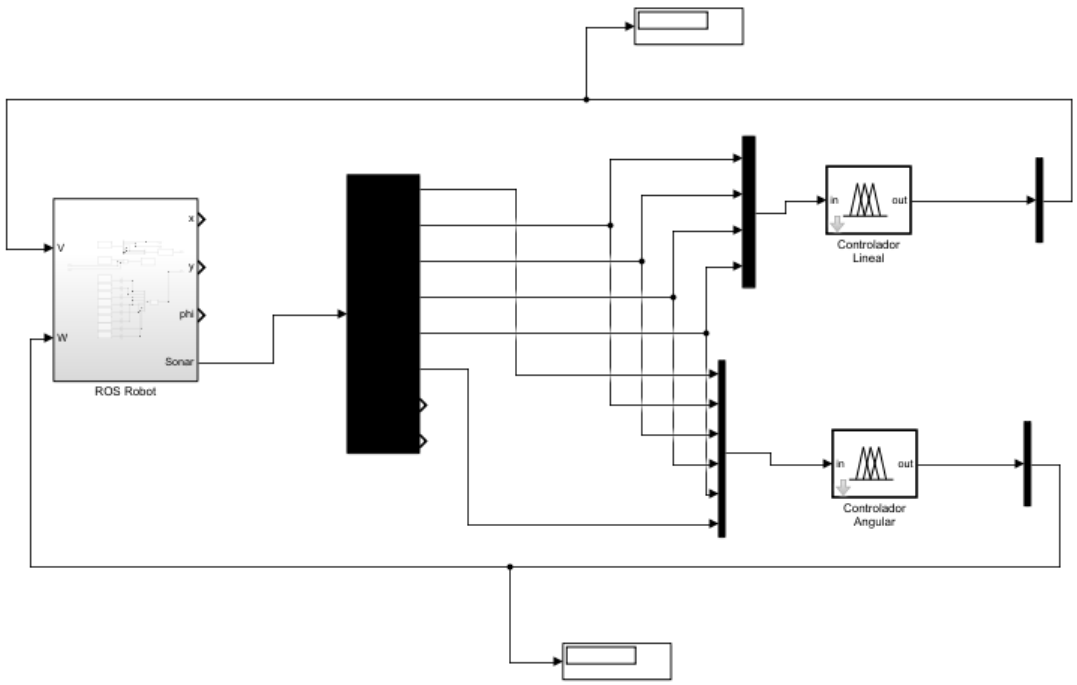
Para el entrenamiento angular, se tiene la siguiente configuración y entrenamiento, y funciones de pertenencia de tipo trimf y constante.

Observamos que la red neuronal entrena y reduce el error conforme avanza el tiempo, y se ajusta a la salida de los datos de entrenamiento:



La configuración del entrenamiento para la velocidad lineal es similar a la del entrenamiento en el circuito sin obstáculos.

En el archivo simulink, modificamos las líneas para que las entradas y salidas sean las adecuadas respecto controlador angular, quedando el siguiente diagrama:



Se ejecuta la simulación y verificamos que el robot completa el circuito. Una vez más, se ha decidido grabar un vídeo que se puede consultar en el siguiente enlace:

https://www.youtube.com/watch?v=GztpFHZHc0w