Miniproyecto

Sistemas de Control Inteligente

Daniel Lopez Moreno 03217279Q
Luis Alejandro Cabanillas Prudencio 04236930P

Contenido

INTRODUCCIÓN

El objetivo de la práctica es el diseño del control de velocidad (lineal y angular) de un robot móvil para que éste recorra un circuito cerrado (delimitado por paredes), en el que pueden aparecer obstáculos estáticos, en el menor tiempo posible.

El robot cuenta con una serie de ***sensores de ultrasonidos*** distribuidos alrededor de este que son capaces de detectar objetos a una distancia en un rango [0.1- 5]m.

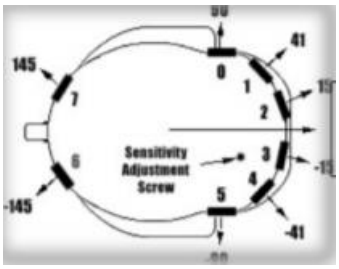

Figura 1: Distribución de los sensores

El comportamiento del robot y su interacción con el entorno se simula en la ***plataforma ROS-STDR***. Robot Operating System o ***ROS***, es una plataforma de desarrollo robótico que permite crear aplicaciones con múltiples sensores y actuadores de forma flexible. Es una colección de herramientas, bibliotecas y convenciones que tiene como objetivo simplificar la tarea de crear un comportamiento de robot complejo y robusto en una amplia variedad de plataformas robóticas.

A través de una Máquina Virtual con el sistema ROS Kinetic ya preinstalado podremos controlar nuestro robot desde Matlab.

Lo primero que hicimos fue configurar el entorno de trabajo necesario para manejar todo lo relativo al robot en el directorio ***robótica_movil_ws***.

Además, debimos añadir el espacio de trabajo al path por defecto añadiendo estas 2 líneas al final del archivo ***.bashrc***:

source ~/robotica_movil_ws/devel/setup.bash

export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:~/robotica_movil_ws/

Y también añadimos las IPs necesarias, extraídas con el comando ***ifconfig***, al final del mismo archivo, en nuestro caso serán las siguientes:

export ROS_MASTER_URI=http://192.168.1.26:11311

export ROS_IP=:192.168.1.26

Una vez realizado todo este proceso de configuración ya pudimos iniciar el simulador con el siguiente comando:

roslaunch stdr_launchers PracticaFinal.launch

A lo largo de la práctica necesitaremos cambiar el mapa que recorrerá nuestro robot entre 3 distintos que se incluyen entre los archivos del Miniproyecto: ***EntornoSinObstaculos.png, EntornoConObstaculos.png y EntornoConObstaculos2.png***. Para ello simplemente basta con modificar el archivo de descripción del mapa ***EntornoPracticaFinal.yaml*** y especificar que archivo .png deseamos utilizar al iniciar el simulador.

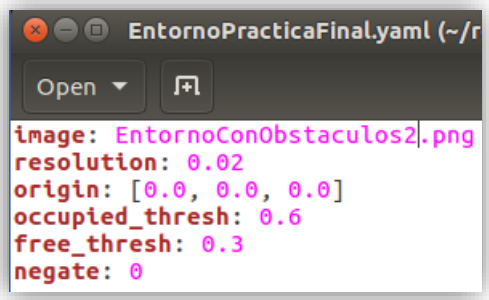

Figura 2: Archivo de descripción del mapa EntornoPracticaFinal.yaml

PARTE 1: Diseño manual de un control borroso de tipo MAMDANI.

En esta primera parte de la práctica se nos pide que nuestro ***amigobot*** recorra 3 circuitos utilizando un Controlador Borroso de ***tipo Mamdani***.

Hemos utilizado la función ***fuzzy*** para generar un ***controlador.fis*** que añadimos a nuestro bloque de Controlador en Simulink y conectamos con el ***bloque ROS Robot*** a través de bloques multiplexores y demultiplexores. En la Figura 3 podemos ver como quedaría el archivo Simulink **cont_mamdani.slx** sin ningún sensor conectado de los 8 disponibles, a lo largo de la

práctica conectaremos aquellos que consideremos necesarios para llevar a cabo una trayectoria exitosa:

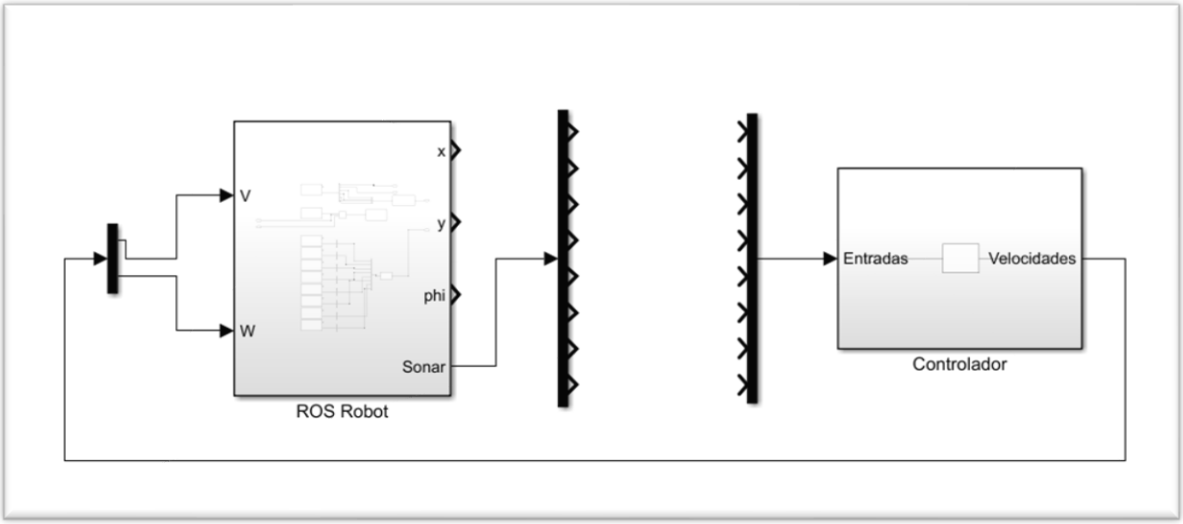

Figura 3: Esquema de Simulink cont_mamdani.slx

Por último, para terminar de configurar todo, antes de comenzar las simulaciones debimos conectar los bloques Suscriptores del bloque ROS Robot a nuestra máquina virtual configurando la dirección red:

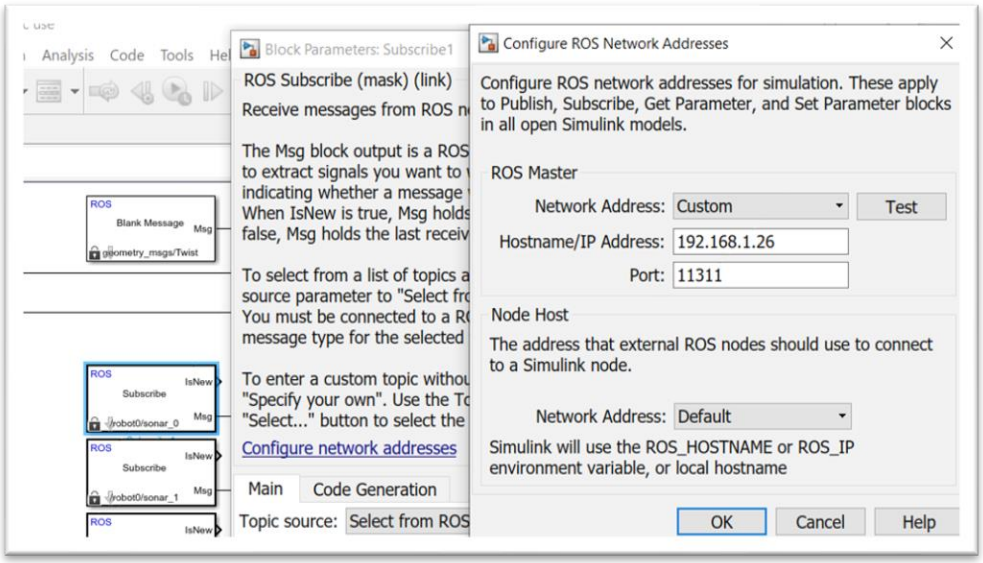

Figura 4: Configuración de red de bloques suscriptores

PARTE 1.1: Circuito Sin Obstáculos

Lo primero que tuvimos que hacer es realizar un controlador borroso de tipo Mamdani que ayude a nuestro robot a superar el circuito con el mapa sin obstáculos.

Para ello, lo primero que hicimos fue determinar cuántos sensores íbamos a utilizar para este caso. Nuestra decisión fue utilizar dos sensores (sonar_0 y sonar_2), para mantenernos cerca de la pared interior. La arquitectura sería la siguiente:

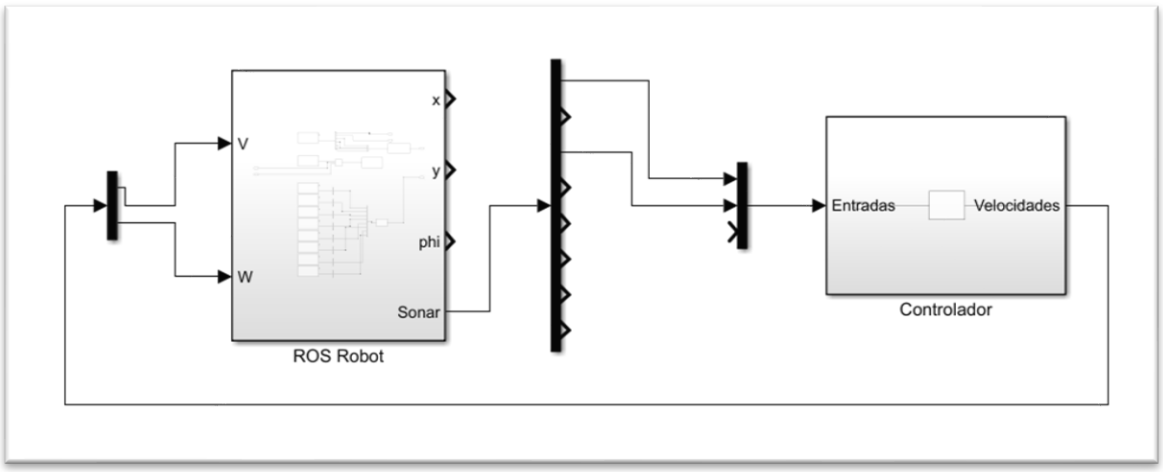

Figura 5: Arquitectura del sistema circuito sin obstáculos

Una vez tuvimos clara la arquitectura, nos pusimos con el controlador borroso. Teniendo en cuenta las medidas del circuito establecimos los mismos límites para el sonar_0 y el sonar_2:

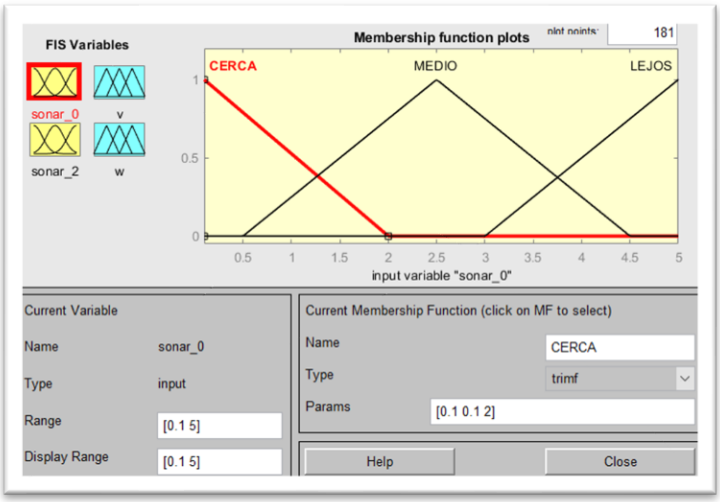

Figura 6: Límites y funciones de pertenencia controlador sin obstáculos

Por otro lado, para las velocidades lineal y angular:

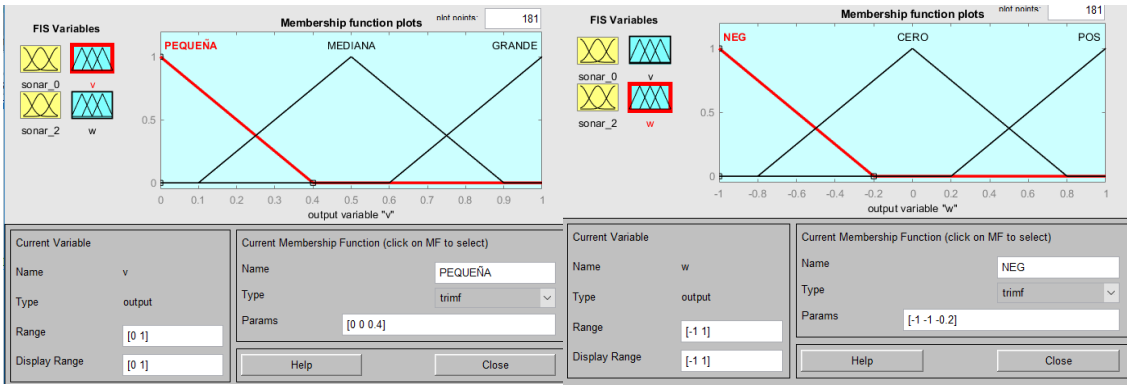

Figura 7: Límites y funciones de pertenencia W y V

Funciones de pertenencia				
Variable	**Rango**	**Nombre de función**	**Forma**	**Parámetros**
Sonar_0	[0.1, 5]	**CERCA**	Trimf	[0.1 0.1 2]
		MEDIO	Trimf	[0.5 2.5 4.5]
		LEJOS	Trimf	[3 5 5]
Sonar_2	[0.1, 5]	**CERCA**	Trimf	[0.1 0.1 2]
		MEDIO	Trimf	[0.5 2.5 4.5]
		LEJOS	Trimf	[3 5 5]
V	[0, 1]	**PEQUEÑA**	Trimf	[0 0 0.4]
		MEDIANA	Trimf	[0.1 0.5 0.9]
		GRANDE	Trimf	[0.6 1 1]
W	[-1, 1]	**NEG**	Trimf	[-1 -1 -0.2]
		CERO	Trimf	[-0.8 0 0.8]
		POS	Trimf	[0.2 1 1]

Tabla 1: Funciones de pertenencia de las variables

Y, por último, las reglas que modelarán el comportamiento del robot serán las siguientes:

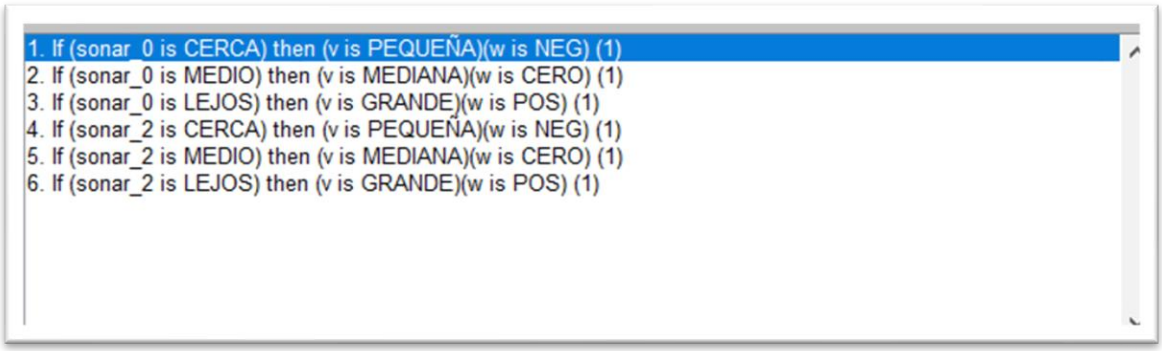

Figura 8: Reglas controlador sin obstáculos

Una vez terminamos de configurar nuestro controlador borroso, lo guardamos en el archivo **ContSinObstaculos.fis**.

Por último, cuando tuvimos todo conectado y configurado correctamente, procedimos a probar nuestro controlador con ROS ejecutando el siguiente código **Ej_Mamdani.m**, especificando en nuestro esquema Simulink que el controlador a utilizar es el recién creado **ContSinObstaculos.fis**:

```
clear all;
close all;
rosshutdown

sim("cont_mamdani.slx")
```

Figura 9: Código .m para la ejecución de nuestro controlador Mamdani.

Los resultados obtenidos fueron satisfactorios, ya que nuestro robot daba una vuelta completa al circuito sin chocarse con nada. **Estos resultados pueden observarse en el archivo de vídeo adjuntado en el entregable del proyecto.**

PARTE 1.2: Circuito con Obstáculos 1

El siguiente circuito que tuvo que superar nuestro controlador fue el "CircuitoConObstaculos1", que pudimos encontrar en la máquina virtual y tiene el siguiente aspecto:

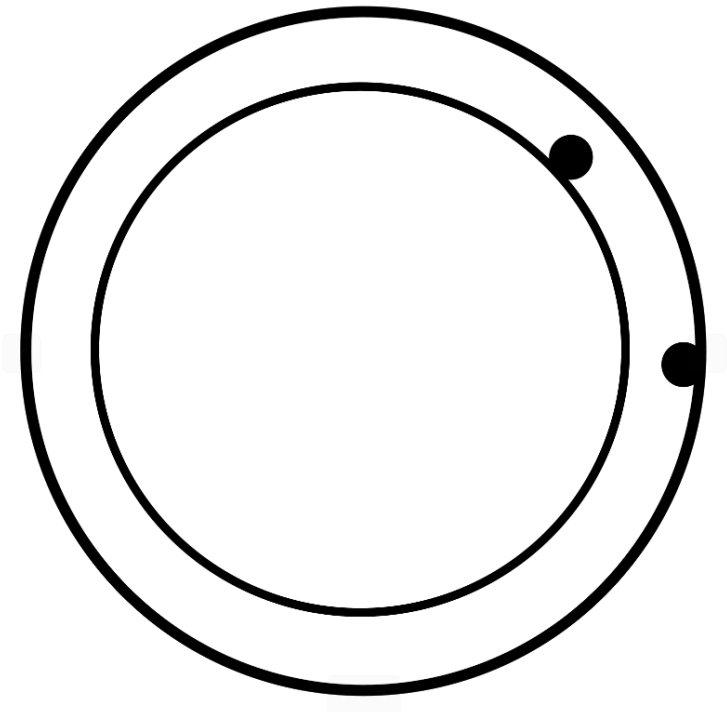

Figura 10: Circuito con obstáculos 1

Como se puede observar, ahora nuestro robot tenía que sortear dos obstáculos para dar una vuelta completa al circuito, por lo que debimos gestionarlo en el controlador. La arquitectura

cambió, utilizamos 3 sensores para controlar que el robot no se choca con ningún obstáculo (sonar_0, sonar_2 y sonar_3). Así, la nueva arquitectura quedó de la siguiente manera:

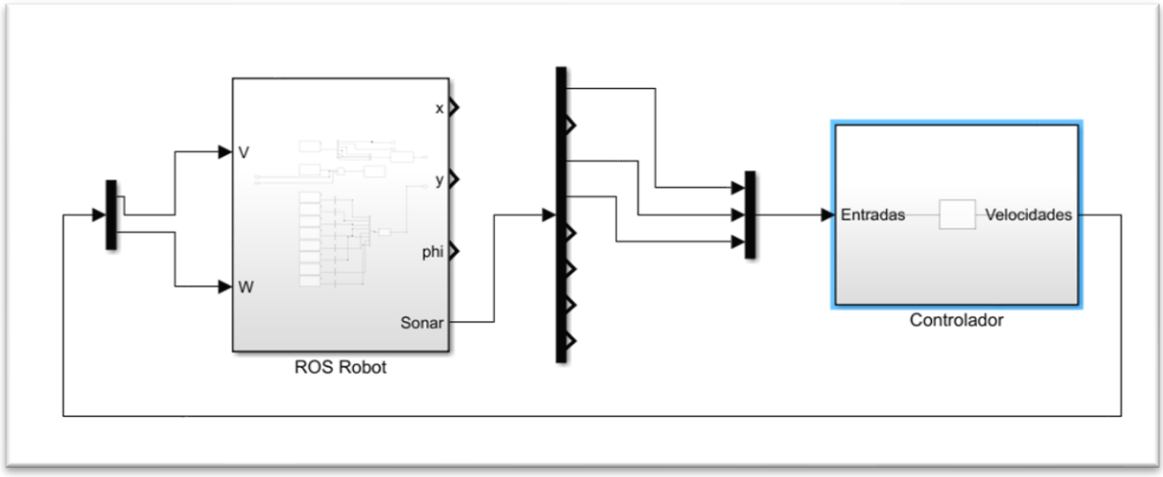

Figura 11: Arquitectura del sistema para el circuito con obstáculos 1

El siguiente paso fue modelar las funciones de pertenencia y límites de cada sonar, así como de la velocidad lineal y angular. Al introducir un nuevo sonar en el modelo, los límites de los sónares cambiaron, pero los de w y v quedan igual que en el modelo anterior. Así, los nuevos límites de los sónares fueron:

Figura 12: Funciones de pertenencia de los sónares para el circuito con obstáculos 1

Funciones de pertenencia				
Variable	**Rango**	**Nombre de función**	**Forma**	**Parámetros**
Sonar_0	[0.1, 5]	**CERCA**	Trimf	[0.1 0.1 1]
		MEDIO	Trimf	[0.5 2.5 4.5]
		LEJOS	Trimf	[3 5 5]
Sonar_2	[0.1, 5]	**CERCA**	Trimf	[0.1 0.1 2]
		MEDIO	Trimf	[0.5 2.5 4.5]
		LEJOS	Trimf	[3 5 5]
Sonar_3	[0.1, 5]	**CERCA**	Trimf	[0.1 0.1 1.5]
		MEDIO	Trimf	[0.5 2 3.5]
		LEJOS	Trimf	[2.5 5 5]
V	[0, 1]	**PEQUEÑA**	Trimf	[0 0 0.4]
		MEDIANA	Trimf	[0.1 0.5 0.9]
		GRANDE	Trimf	[0.6 1 1]
W	[-1, 1]	**NEG**	Trimf	[-1 -1 -0.2]
		CERO	Trimf	[-0.8 0 0.8]
		POS	Trimf	[0.2 1 1]

Tabla 2: Funciones de pertenencia de las variables

Y, por último, las reglas que modelarán el comportamiento del robot serán las siguientes:

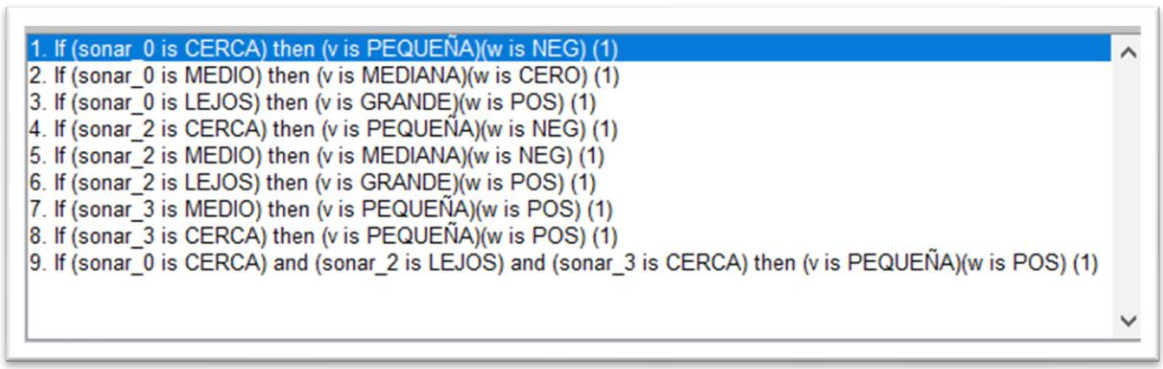

Figura 13: Reglas para el circuito con obstáculos 1

Una vez configurado completamente nuestro nuevo controlador borroso, lo guardamos en el archivo **ContConObstaculos.fis**.

Por último, cuando tuvimos todo conectado y configurado correctamente, procedimos a probar nuestro controlador con ROS. Para ello utilizamos exactamente el mismo código **Ej_Mamdani.m** mencionado en el apartado anterior y expuesto en la Figura 9. Además, fue necesario especificar en nuestro esquema Simulink que el controlador a utilizar esta vez es **ContConObstaculos.fis**.

Los resultados obtenidos fueron satisfactorios, ya que nuestro robot daba una vuelta completa al circuito sin chocarse con nada y evitando los obstáculos. **Estos resultados pueden observarse en el archivo de vídeo adjuntado en el entregable del proyecto.**

PARTE 1.3: Circuito con Obstáculos 2

El último circuito que tuvo que superar nuestro controlador es el “CircuitoConObstaculos2”, que pudimos encontrar en la máquina virtual y tiene el siguiente aspecto:

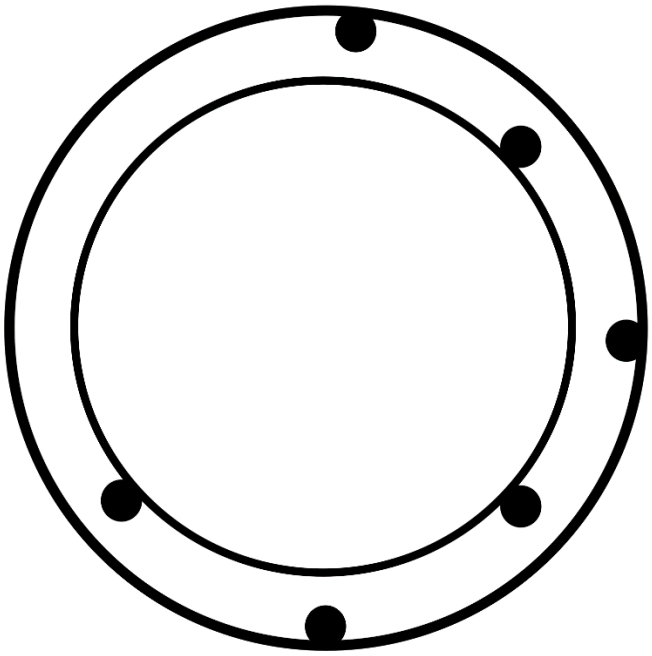

Figura 14: Circuito con obstáculos 2

En este caso nuestro amigobot debía recorrer un circuito que cuenta con **6 obstáculos** distribuidos entre las paredes interior y exterior de este. Para realizar este circuito no ha sido necesario utilizar las lecturas de ningún sonar extra, por lo que las entradas de nuestro controlador seguirán siendo sonar_0, sonar_2 y sonar_3 como se puede apreciar en la Figura 15:

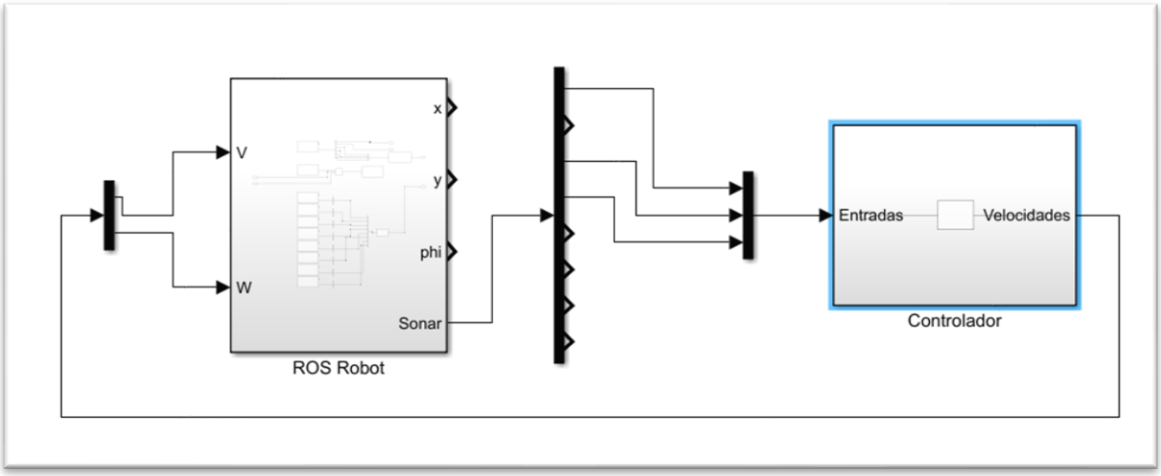

Figura 15: Arquitectura del sistema para el circuito con obstáculos 2

Sin embargo, aunque los sónares utilizados son los mismos que en el apartado anterior, si que fue necesario modificar algunas funciones de pertenencia de los sónares para evitar choques con las paredes. Las funciones de pertenencia de las salidas V y W se mantienen exactamente igual. En la Figura 16 podemos ver las nuevas funciones de pertenencia de los sónares:

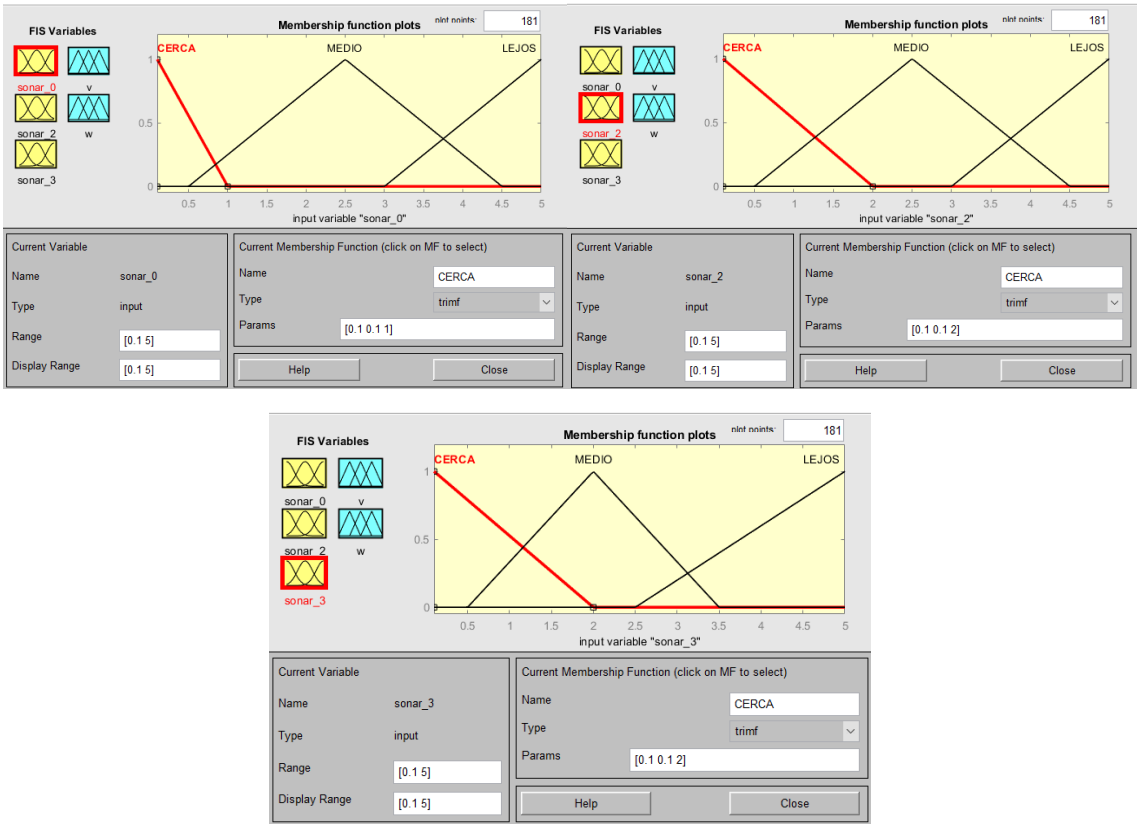

Figura 16: Funciones de pertenencia de los sónares para el circuito con obstáculos 2

Funciones de pertenencia				
Variable	**Rango**	**Nombre de función**	**Forma**	**Parámetros**
Sonar_0	[0.1, 5]	**CERCA**	Trimf	[0.1 0.1 1]
		MEDIO	Trimf	[0.5 2.5 4.5]
		LEJOS	Trimf	[3 5 5]
Sonar_2	[0.1, 5]	**CERCA**	Trimf	[0.1 0.1 2]
		MEDIO	Trimf	[0.5 2.5 4.5]
		LEJOS	Trimf	[3 5 5]
Sonar_3	[0.1, 5]	**CERCA**	Trimf	[0.1 0.1 2]
		MEDIO	Trimf	[0.5 2 3.5]
		LEJOS	Trimf	[2.5 5 5]
V	[0, 1]	**PEQUEÑA**	Trimf	[0 0 0.4]
		MEDIANA	Trimf	[0.1 0.5 0.9]
		GRANDE	Trimf	[0.6 1 1]
W	[-1, 1]	**NEG**	Trimf	[-1 -1 -0.2]
		CERO	Trimf	[-0.8 0 0.8]
		POS	Trimf	[0.2 1 1]

Tabla 3: Funciones de pertenencia de las variables

Y, por último, las reglas que modelarán el comportamiento del robot serán las siguientes:

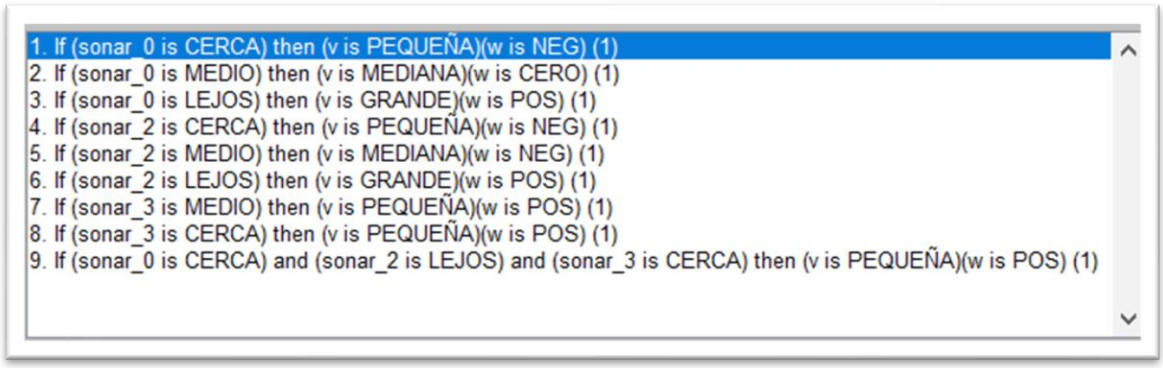

Figura 16: Reglas para el circuito con obstáculos 1

Una vez terminamos de configurar este último controlador borroso, lo guardamos en el archivo **ContConObstaculos2.fis**.

Por último, cuando tuvimos todo conectado y configurado correctamente, procedimos a probar nuestro controlador con ROS. De nuevo, el código utilizado para ejecutar la simulación es **Ej_Mamdani.m** y especificaremos en Simulink que el controlador a utilizar es **ContConObstaculos2.fis**.

Los resultados obtenidos fueron satisfactorios, ya que nuestro robot daba una vuelta completa al circuito sin chocarse con nada y evitando los obstáculos. **Estos resultados pueden observarse en el archivo de vídeo adjuntado en el entregable del proyecto.**

PARTE 1.4: Conclusiones Mamdani

Durante todo el proceso de desarrollo de Mamdani hemos generado un total de 3 controladores borrosos:

- **ContSinObstaculos:** Utilizado en el apartado 1.1 para realizar el circuito sin obstáculos.
- **ContConObstaculos:** Utilizado en el apartado 1.2 para realizar el circuito con obstáculos 1.
- **ContConObstaculos2:** Utilizado en el apartado 1.3 para realizar el circuito con obstáculos 2.

A diferencia de los otros 2 primeros controladores desarrollados, este último **si se puede utilizar en todos los circuitos**. Si intentásemos utilizar el **ContSinObs** en un circuito con obstáculos nuestro robot acabaría chocando. Y si intentásemos utilizar el **ContConObs1** en este último **CircuitoConObstaculos2** también acabaríamos colisionando nuestro amigobot.

Controlador borroso	Circuito Sin Obstáculos	Circuito Con Obstáculos	Circuito Con Obstáculos 2
ContSinObstaculos	**V**	**X**	**X**
ContConObstaculos	**V**	**V**	**X**
ContConObstaculos2	**V**	**V**	**V**

Tabla 4: Circuitos que es capaz de recorrer cada controlador.

Por lo tanto, el controlador definitivo que debemos utilizar es **ContConObs2**, ya que será capaz de realizar los 3 circuitos sin ningún tipo de problema.

PARTE 2: Diseño automático de un controlador neuroborroso de tipo SUGENO.

Al igual que en la primera parte del Miniproyecto, en esta segunda se nos pide que nuestro **amigobot** recorra los 3 mismos circuitos anteriores, pero esta vez debemos desarrollar **controladores neuroborrosos de tipo Sugeno** en lugar de Mamdani.

Para ello, utilizamos la función ***anfisedit***, en lugar de la función fuzzy utilizada anteriormente, para generar un **controlador.fis** que añadimos a nuestro bloque Controlador en Simulink. Sin embargo, con la función anfisedit surge un problema, y es que solamente es capaz de generar un **controlador con una única salida**.

Por lo tanto, como necesitamos gestionar 2 salidas, V y W, debemos generar **2 controladores neuroborrosos** distintos (**ControladorVelocidadLineal y ControladorVelocidadAngular**) y conectarlos al bloque ROS Robot.

En la Figura 17 podemos ver como quedaría nuestro archivo Simulink **cont_sugeno.slx** sin ningún sensor conectado de los 8 disponibles, a lo largo de la práctica conectaremos aquellos que consideremos necesarios para una trayectoria exitosa:

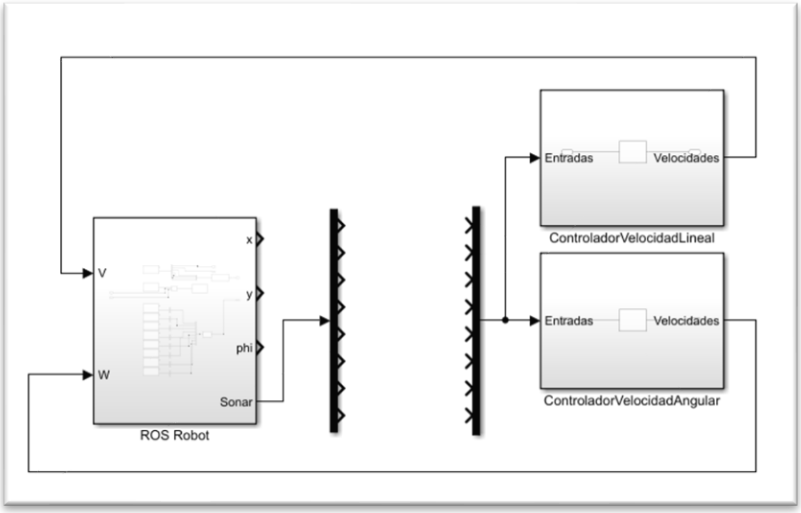

Figura 17: Esquema de Simulink cont_sugeno.slx.

En cuanto a la configuración de red de los bloques subscriber de ROS Robot, al igual que en la Parte 1, debimos añadir la IP de la máquina virtual ROS.

PARTE 2.1: Circuito Sin Obstáculos

Para empezar, al igual que en la Parte 1, comenzamos a recorrer el Circuito Sin Obstáculos con nuestro amigobot.

Para generar nuestro controlador de Sugeno, primero de todo recorrimos el circuito de forma manual. Para ello empleamos el código ya entregado por los profesores con los demás archivos de la práctica **ControlManualRobot.m**, simplemente modificando las IPs del código por las nuestras propias para poder inicializar ROS correctamente.

En la variable **ROS_MASTER_IP** especificamos la IP de nuestra máquina virtual con ROS, y en **ROS_IP** especificamos la IP de Windows:

```
12 -    ROS_MASTER_IP = '192.168.1.26'
13 -    ROS_IP = '192.168.1.139'
14
15 -    rosinit(['http://',ROS_MASTER_IP,':11311'],'NodeHost',ROS_IP)
```

Figura 18: Fragmento de código ControlManualRobot.m de inicialización de IPs

Después de recorrer el circuito manualmente, ControlManualRobot.m se encarga de guardar los datos recogidos por los 8 sensores, posición y la velocidad angular y lineal en la variable **datos_entrenamiento**.

A continuación, utilizando el código que creamos llamado **Entrenamiento.m** pudimos cargar en las variables **train_angular y train_linear** los datos de los **sensores 0, 2 y 3** (columnas 1, 3 y 4 de training) y las **velocidades angular y lineal** (columnas 12 y 13).

```
1 -    load ("datos_entrenamiento.mat")
2 -    train_angular = training(:,[1,3,4,12])
3 -    indices_angular = round(linspace(1,size(training,1),1500))
4 -    train_angular = train_angular(indices_angular,:)
5 -    train_angular(isinf(train_angular)) = 5.0
6 -    train_angular = double(train_angular)
7
8 -    train_linear = training(:,[1,3,4,13])
9 -    indices_linear = round(linspace(1,size(training,1),1500))
10 -   train_linear = train_linear(indices_linear,:)
11 -   train_linear(isinf(train_linear)) = 5.0
12 -   train_linear = double(train_linear)
```

Figura 19: Código Entrenamiento.m

Una vez cargamos los datos en las variables, pudimos comenzar a entrenar nuestros controladores neuroborrosos para que intenten imitar el recorrido manual.

ANFISEDIT Circuito Sin Obstáculos:

Datos de entrenamiento

Utilizando **anfisedit** especificamos qué variable deseamos cargar del workspace para entrenar. En la Figura 20 podemos ver las lecturas de datos de las variables train_angular y train_linear:

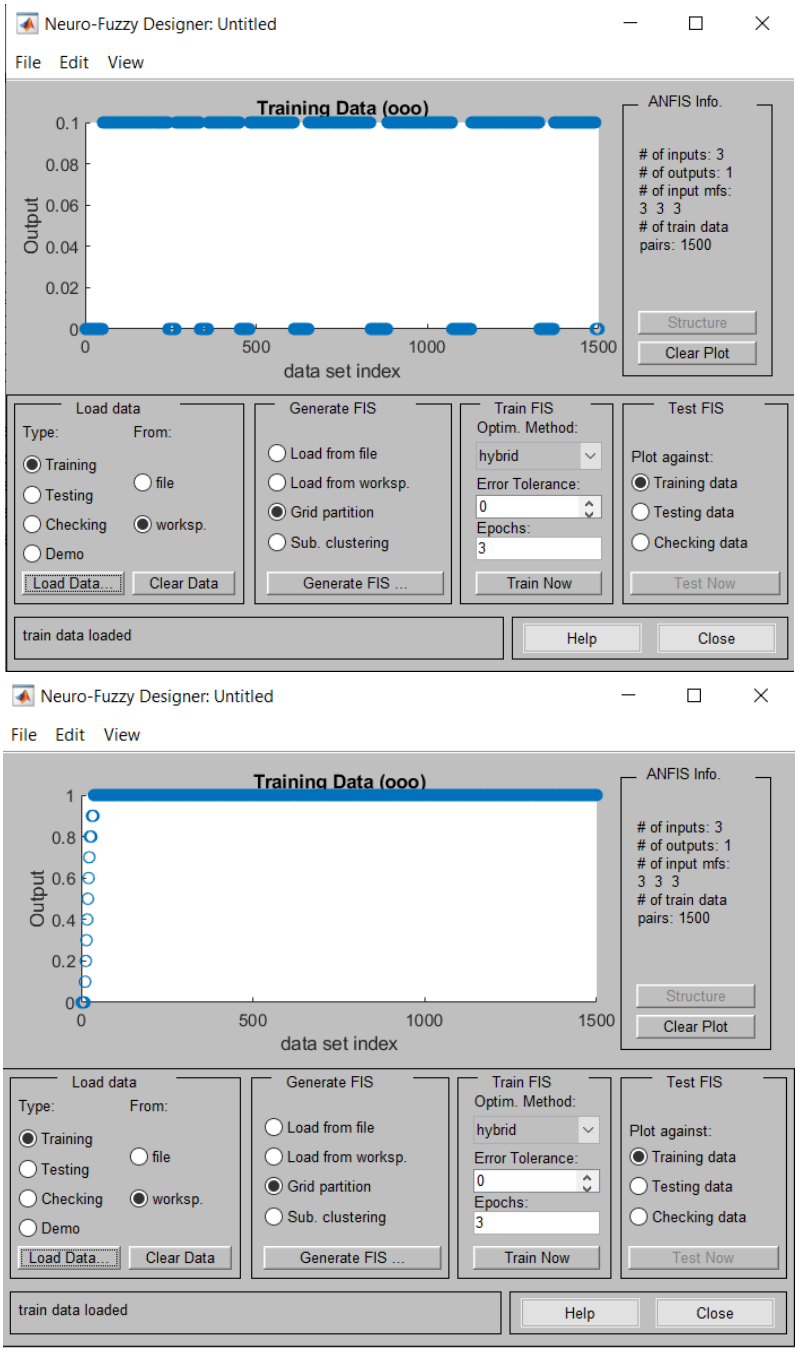

Figura 20: Datos de entrenamiento de velocidad angular y lineal

Los datos de **Velocidad Angular varían entre 0 y 0.1** ya que al recorrer el circuito sin obstáculos simplemente debíamos desviarnos hacia la izquierda (w = 0.1) cuando nos alejásemos de la pared interior.

En el caso de la **Velocidad Lineal se mantiene siempre en valor V=1** ya que no es necesario reducirla para evitar ningún obstáculo

Error en el entrenamiento

Lo siguiente que hicimos fue hacer click en Generate FIS, utilizando el **método Grid Partition** y especificando que deseábamos **3 funciones de pertenencia** por cada una de las entradas. Y posteriormente entrenamos nuestro FIS poniendo un límite de **100 Epochs**.

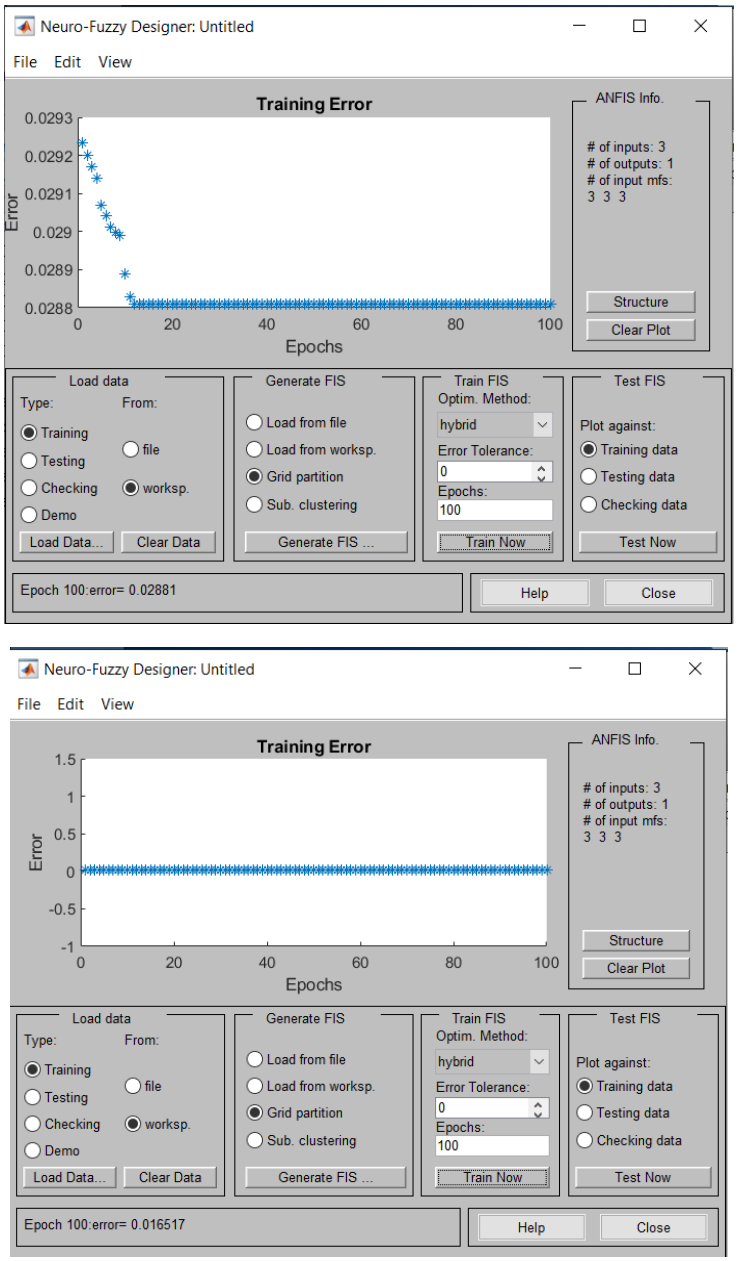

Figura 21: Resultado del entrenamiento de velocidad angular y lineal

El error en la Velocidad Angular comenzó alrededor de 0.0292 y bajó levemente hasta 0.0288, valores bastante bajos por lo que podemos comprobar que el entrenamiento está siendo efectivo, ya que **los valores de salida son casi iguales al target**.

En el caso de la Velocidad Lineal, **el error se mantiene constante en 0.016517**.

Test y comparativa

Por último, hicimos click en **Test Now** para comparar los datos de entrenamiento con la salida esperada de nuestro controlador:

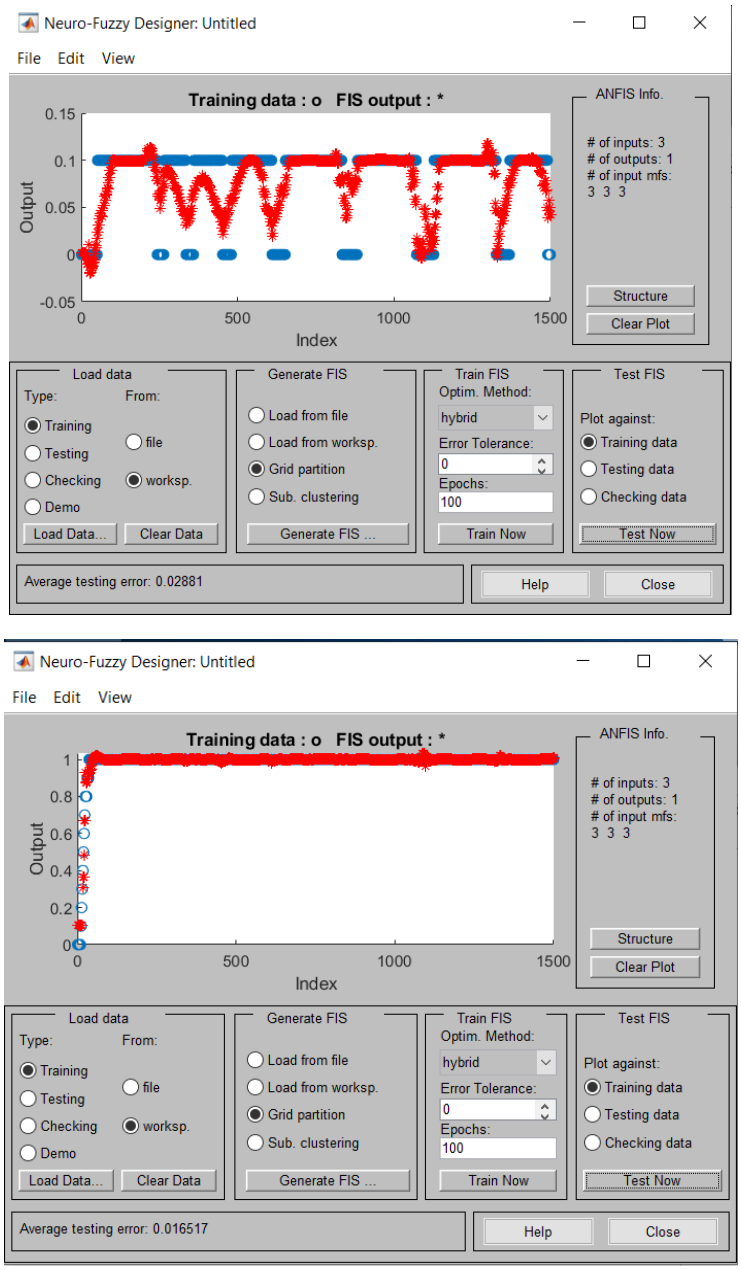

Figura 22: Gráfica de comparación de velocidad angular y lineal

Al comparar los datos de entrenamiento de la Velocidad Angular con la salida del controlador podemos ver como **cuando la velocidad es 0.1 coinciden**, pero en los leves momentos que se reduce a 0 se puede apreciar un pico descendente pero no encaja perfectamente debido a que es un tiempo muy breve y es un salto directo sin escalas.

En el caso de la Velocidad Lineal, **los datos de entrenamiento coinciden perfectamente con las salidas del controlador**.

Una vez finalizado todo el proceso solo queda exportar nuestros controladores en los archivos **AngSinObs.fis** y **LinSinObs.fis** y especificar en nuestro archivo Simulink **cont_sugeno.slx** que

estos son los controladores que deseamos utilizar en los bloques **ControladorVelocidadAngular** y **ControladorVelocidadLineal**.

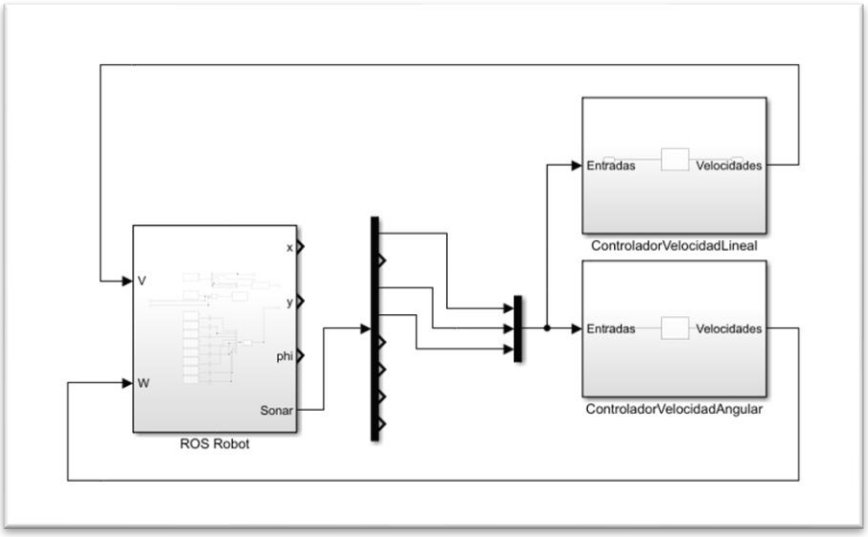

Figura 23: Arquitectura del sistema cont_sugeno.slx

Para comprobar que nuestros controladores logran recorrer el circuito sin problema utilizaremos el siguiente código Ej_Sugeno.m:

```
1 -    clear all;
2 -    close all;
3 -    rosshutdown
4
5 -    sim("cont sugeno.slx")
```

Figura 24: Código de Ej_Sugeno.m

Los resultados obtenidos fueron satisfactorios, ya que nuestro robot daba una vuelta completa al circuito sin chocarse con nada. **Estos resultados pueden observarse en el archivo de vídeo adjuntado en el entregable del proyecto.**

PARTE 2.2: Circuito con Obstáculos 1

Ya realizado el circuito sin obstáculos, procedimos a recorrer el **Circuto Con Obstáculos 1**. Al igual que en el apartado anterior, utilizamos el código ControlManualRobot.m para recorrer el circuito.

La única diferencia esa vez a la hora de recorrerlo fue que debíamos tener en cuenta los obstáculos, por lo que debíamos **desviar la trayectoria** de nuestro robot y **reducir la velocidad** lineal al acercarnos a ellos.

Sin embargo, fue esencial tener en cuenta que, aunque nosotros fuésemos capaces de ver el obstáculo en todo momento desde arriba, **no debíamos comenzar a desviar el robot hasta que los sensores de este no lo hubieran detectado** o nos encontraríamos con datos de entrenamiento erróneos.

De nuevo, al igual que en el apartado anterior, utilizamos el código **Entrenamiento.m** para cargar en las variables **train_angular y train_linear** los datos de los **sensores 0,2 y 3** y las **velocidad angular y lineal**.

Una vez cargamos los datos en las variables, pudimos comenzar a entrenar nuestros controladores neuroborrosos para que intenten imitar el recorrido manual de este Circuito Con Obstáculos 1.

ANFISEDIT Circuito Con Obstáculos:

Datos de entrenamiento

Al igual que en el apartado anterior, utilizamos la función **anfisedit** para generar los controladores y cargar los datos de entrenamiento desde el workspace:

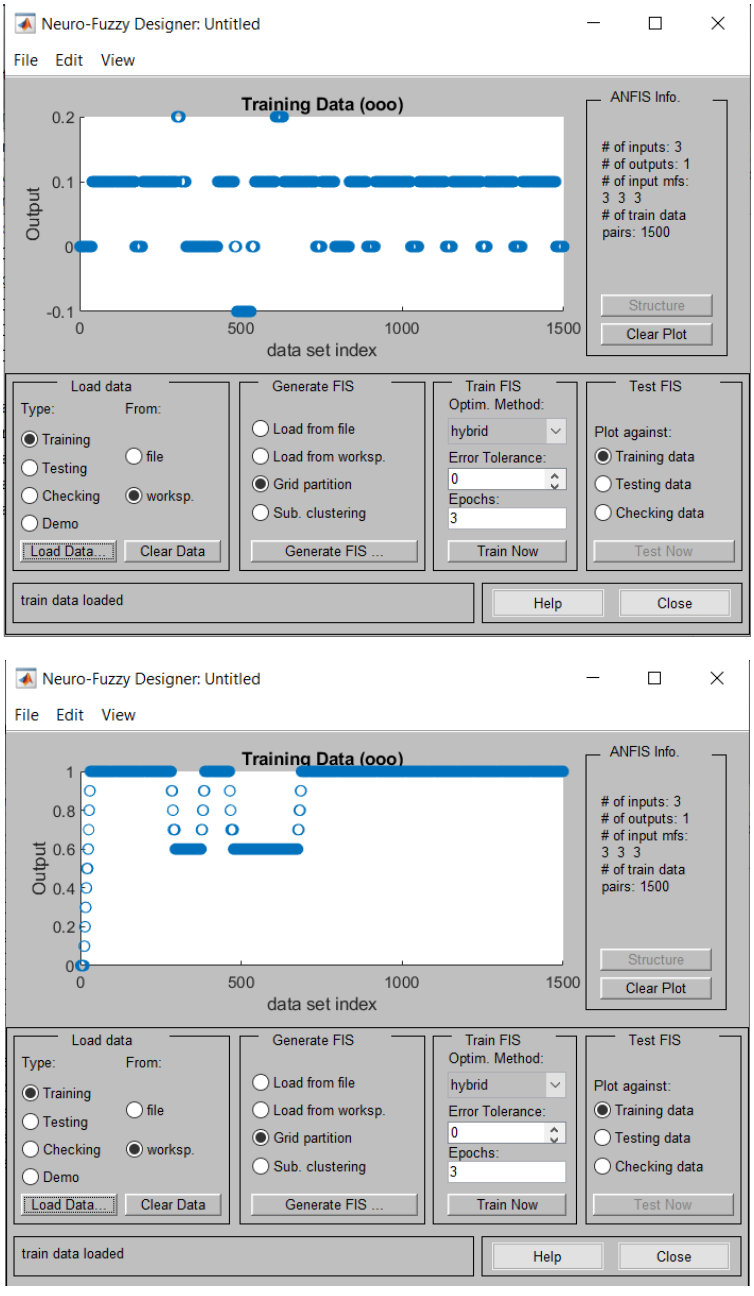

Figura 25: Datos de entrenamiento de velocidad angular y lineal

Los datos de **Velocidad Angular varían entre -0.1 y 0.1** para desplazarse hacia izquierda y derecha evitando obstáculos, excepto en casos excepcionales que llega a ser 0.2 cuando al evitar un obstáculo el robot se aproxima en exceso a la pared exterior.

En el caso de la **Velocidad Lineal** esta vez el valor no es constantemente 1. Ahora la velocidad lineal tendrá valor 0.6 o 1. Su valor es 1 cuando el robot pueda moverse libremente y será 0.6 cuando detecte un obstáculo y deba reducir velocidad y comenzar a desviarse.

Error en el entrenamiento

De nuevo, hicimos click en Generate FIS, utilizando el **método Grid Partition** y especificando que deseábamos **3 funciones de pertenencia** por cada entrada. Y posteriormente entrenamos nuestro FIS con un límite de **100 Epochs.**

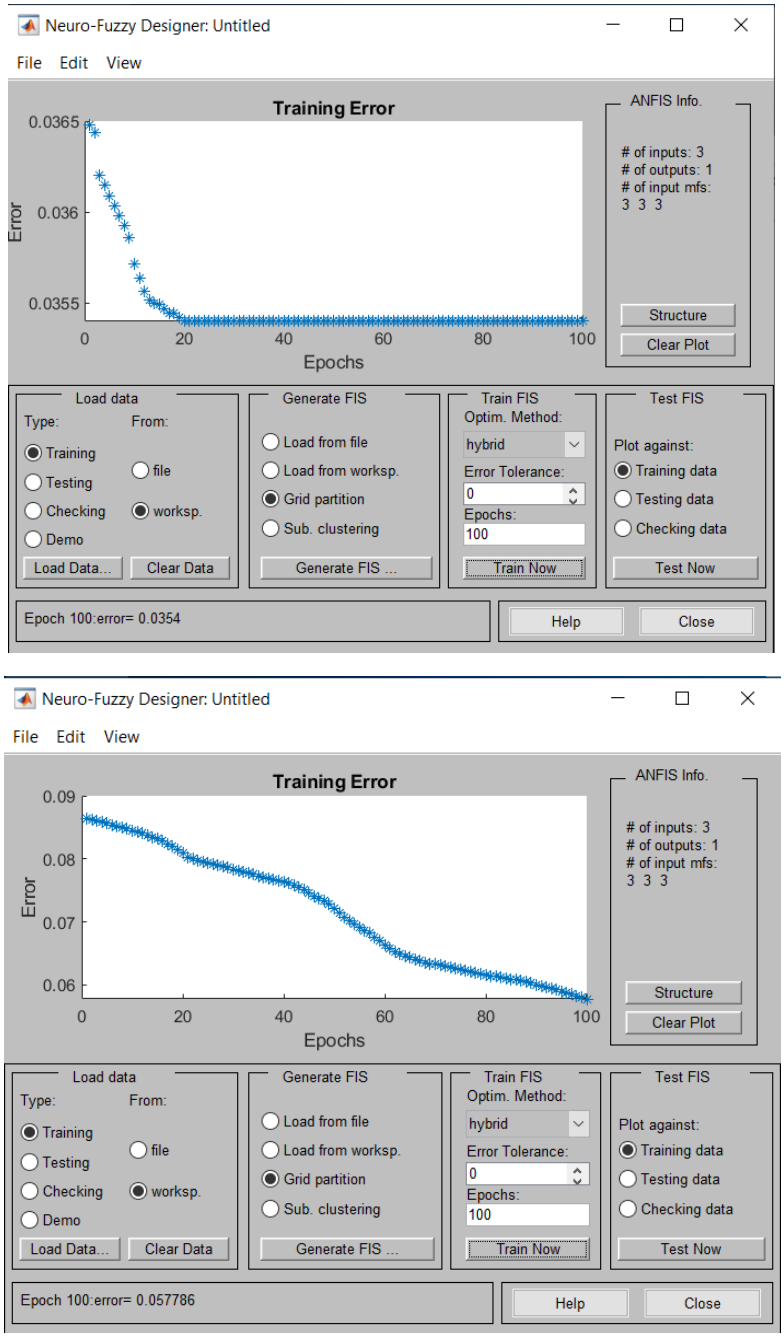

Figura 26: Resultado del entrenamiento de velocidad angular y lineal

Esta vez el error en la Velocidad Angular es levemente mayor al del circuito sin obstáculos, sin embargo, **sigue siendo un bastante bajo rondando el valor 0.0355**.

En cuanto al **error en la Velocidad Lineal** ahora si podemos notar que **no es constante**, ya que esta vez la velocidad lineal no siempre tiene valor 1. Aun así, **el error sigue siendo bastante reducido comenzando en 0.85 y disminuyendo hasta 0.57**.

Test y comparativa

Por último, hicimos click en **Test Now** para comparar los datos de entrenamiento con la salida esperada de nuestro controlador:

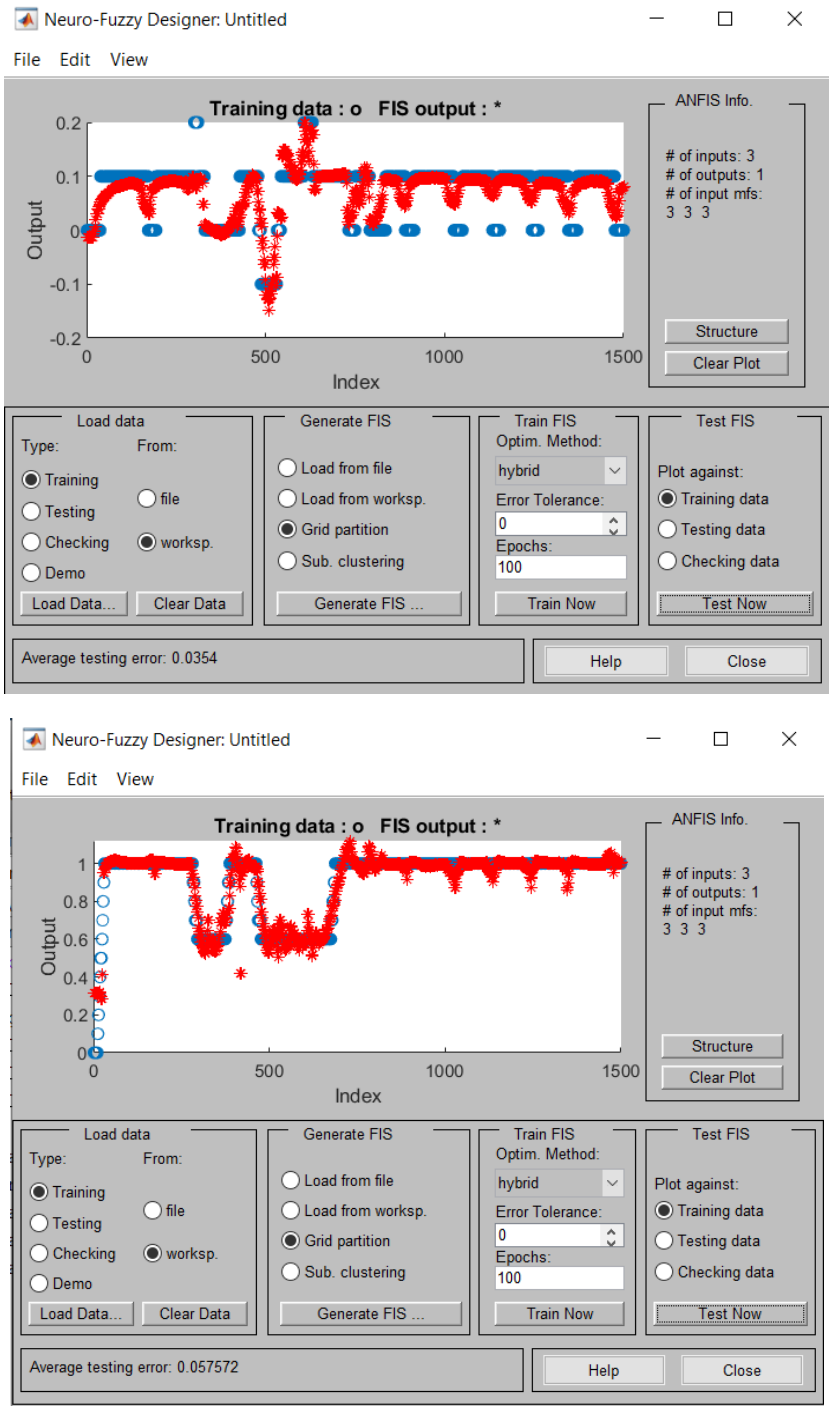

Figura 27: Gráfica de comparación de velocidad angular y lineal

Por último, esta vez podemos comprobar que los datos de entrenamiento de la Velocidad Angular no siempre coinciden con la salida del controlador, **pero se aproximan bastante cuando el valor es 0.1**, y al igual que en el circuito con obstáculos, **notamos picos en los breves momentos que la velocidad cambia a -0.1, 0 ó 0.2**.

En cuanto a la Velocidad Lineal, la comparativa no es tan exacta como antes, pero **sigue siendo casi perfecta** con un error muy reducido tanto cuando la velocidad es 1 como cuando es 0.6.

Una vez finalizado todo el proceso solo queda exportar nuestros controladores en los archivos **AngConObs.fis** y **LinConObs.fis** y especificar en nuestro archivo Simulink **cont_sugeno.slx** que estos son los controladores que deseamos utilizar en los bloques **ControladorVelocidadAngular** y **ControladorVelocidadLineal**.

La arquitectura del archivo Simulink cont_sugeno.slx se mantuvo igual a la del apartado anterior ya que utilizamos los mismos 3 sensores para la entrada de los controladores.

Y por último, para comprobar que nuestros nuevos controladores lograban recorrer el circuito con obstáculos con éxito, utilizamos de nuevo el código Ej_Sugeno.m.

Los resultados obtenidos fueron satisfactorios, ya que nuestro robot daba una vuelta completa al circuito sin chocarse con nada y evitando los obstáculos. **Estos resultados pueden observarse en el archivo de vídeo adjuntado en el entregable del proyecto**.

PARTE 2.3: Circuito con Obstáculos 2

Y para acabar, tras recorrer CircuitoSinObstaculos y CircuitoConObstaculos1 era la hora de recorrer el **Circuito Con Obstáculos 2.** Al igual que en los anteriores apartados utilizamos de nuevo el código ControlManuelRobot.m.

Esa vez el comportamiento en cuanto a las velocidades angular y lineal era parecido al que debíamos tener recorriendo el Circuito Con Obstáculos 1, **reduciendo la velocidad lineal** y **desviando la trayectoria** cuando nos acercásemos a un obstáculo.

La única diferencia es que en este caso el circuito contaba con **6 obstáculos** en lugar de solo 2, distribuidos entre las paredes interior y exterior de este.

Al igual que en el apartado anterior, fue esencial tener en cuenta cuando los sensores detectaban realmente el obstáculo y no comenzar a esquivarlo hasta entonces para evitar datos de entrenamiento equivocados.

De nuevo, utilizamos el código **Entrenamiento.m** para cargar en las variables **train_angular y train_linear** los datos de los **sensores 0,2 y 3** y las **velocidad angular y lineal**.

Una vez cargamos los datos en las variables, pudimos comenzar a entrenar nuestros controladores neuroborrosos para que intenten imitar el recorrido manual de este Circuito Con Obstáculos 2.

ANFISEDIT Circuito Con Obstáculos 2:

Datos de entrenamiento

Como siempre, utilizamos la función **anfisedit** para generar los controladores y cargar los datos de entrenamiento desde el workspace:

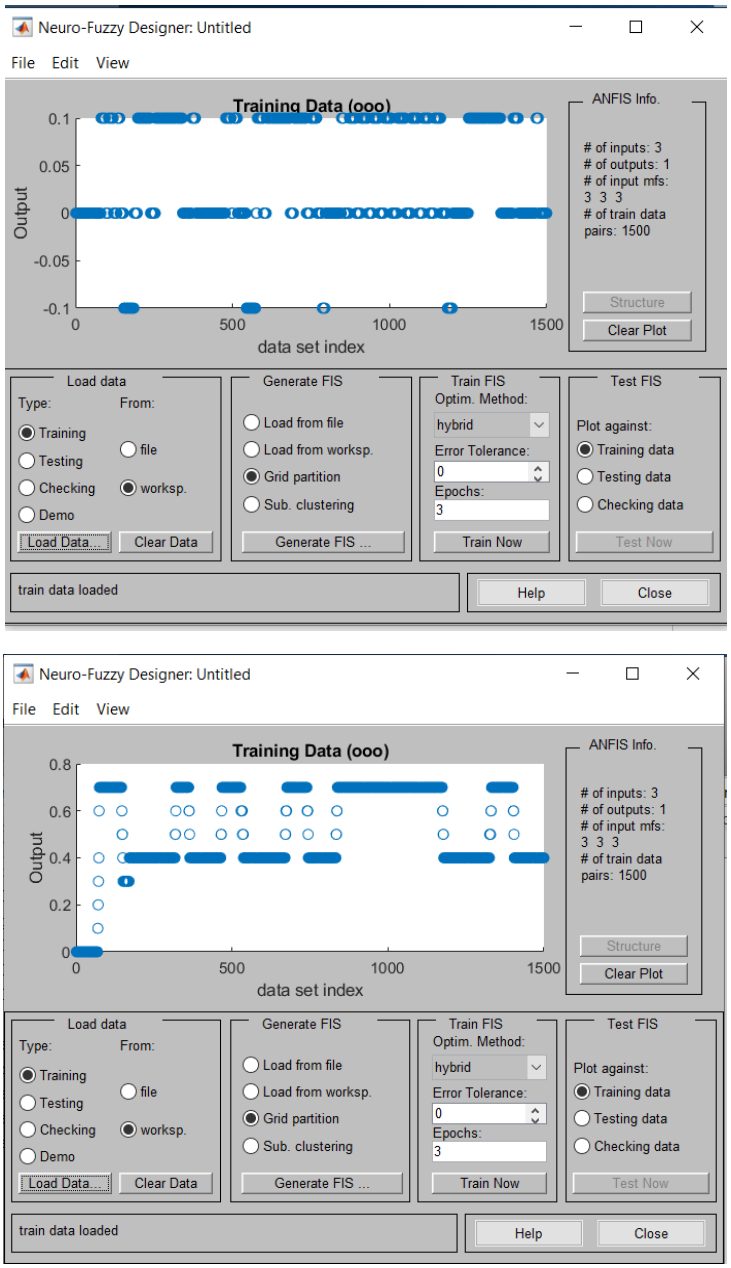

Figura 28: Datos de entrenamiento de velocidad angular y lineal

Para este último circuito los datos de **Velocidad Angular varían en el rango [-0.1, 0.1]**, aunque al igual que en ocasiones anteriores, la mayor parte del tiempo su valor es igual a 0 o 0.1, por lo que el robot avanzaría en línea recta y giraría hacia la izquierda.

En cuanto a la Velocidad Lineal encontramos unos datos algo distintos a los casos anteriores, ya que esta vez la **velocidad máxima es 0.7 y la velocidad mínima es 0.4**. Esto se debe a que este circuito supone una mayor dificultad para recorrerlo sin colisiones, por lo que acabamos utilizando velocidad 0.7 cuando el robot tenía vía libre de obstáculos, y 0.4 cuando detectase uno y debiera comenzar a desviarse.

Error en el entrenamiento

Nuevamente, hicimos click en Generate FIS, utilizando el **método Grid Partition** y especificando que deseábamos **3 funciones de pertenencia** por cada entrada. Y posteriormente entrenamos nuestro FIS con un límite de **100 Epochs.**

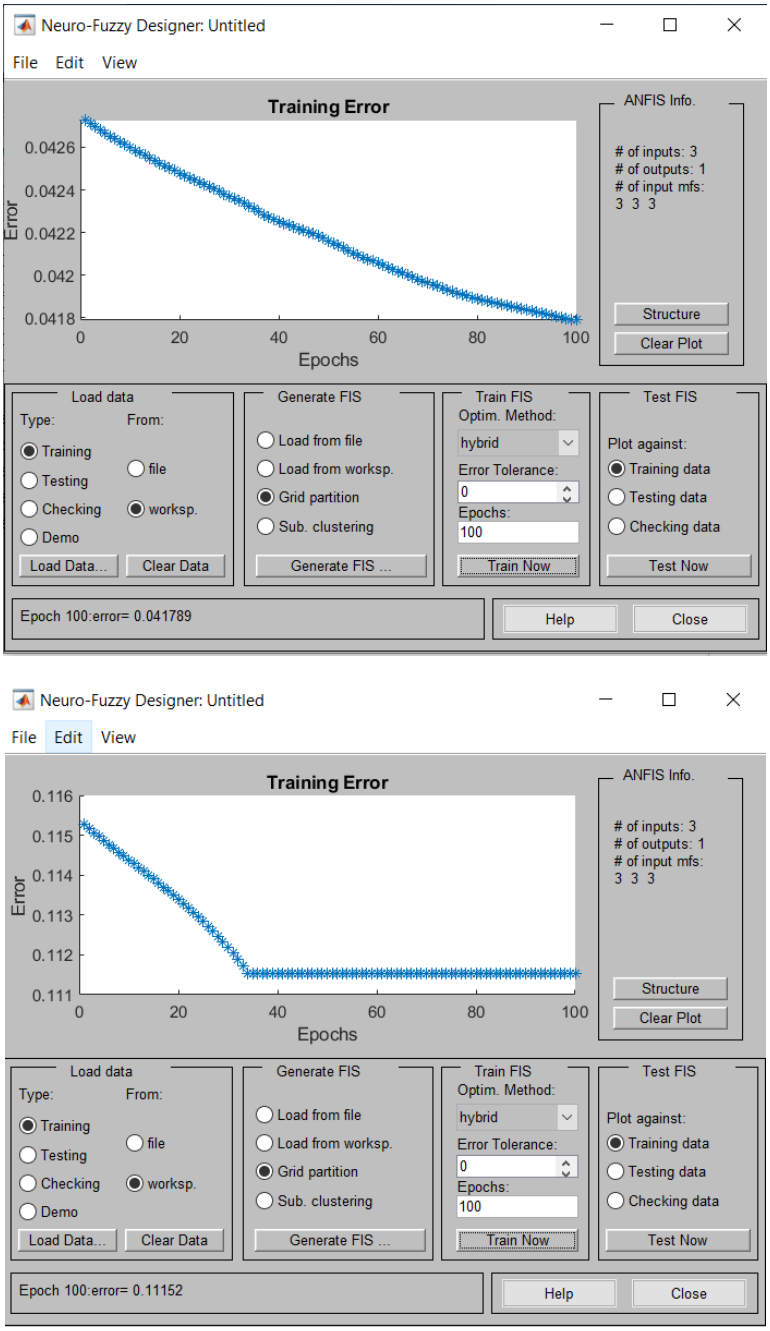

Figura 29: Resultado del entrenamiento de velocidad angular y lineal

Para este último circuito, **el error en la Velocidad Angular es de nuevo ligeramente mayor** al de ocasiones anteriores, aunque sigue siendo bastante reducido **rondando valores de 0.042**.

En cuanto a la **Velocidad Lineal si encontramos una gran diferencia** ya que esta vez **el error es bastante más elevado rondando valores de 0.12**. Este error pese a ser bastante reducido es mucho mayor al de los anteriores circuitos, lo que provocará que en la comparativa **no coincidan perfectamente las salidas con los target**.

Test y comparativa

Por último, hicimos click en **Test Now** para comparar los datos de entrenamiento con la salida esperada de nuestro controlador:

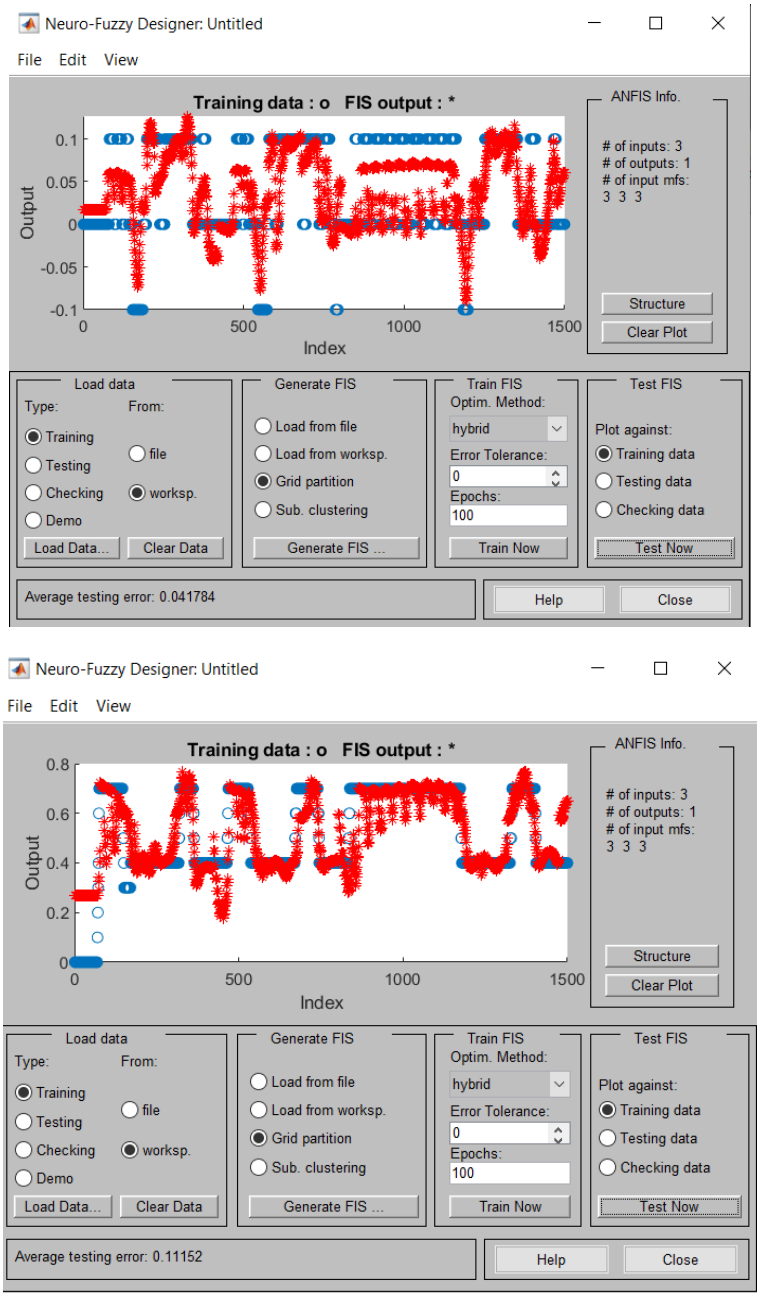

Figura 30: Gráfica de comparación de velocidad angular y lineal

Esta vez la comparativa ya no es perfecta al contrario que en apartados anteriores. **La salida del controlador en cuanto a Velocidad Angular no encaja con los datos target de entrenamiento**, sin embargo, si se puede apreciar que, aunque no los imite a la perfección, el controlador **si es capaz de detectar cuando existen picos y cambios en la velocidad angular** e imitar esos cambios.

En cuanto a la Velocidad Lineal la comparativa tampoco es tan exacta como en los anteriores circuitos, esto se debe a que el error como ya hemos mencionado es ligeramente mayor. Sin

embargo, **el controlador logra detectar cuando hay cambios en la velocidad e imitarlos casi a la perfección**.

Una vez finalizado todo el proceso solo queda exportar nuestros controladores en los archivos **AngConObs2.fis** y **LinConObs2.fis** y especificar en nuestro archivo Simulink **cont_sugeno.slx** que estos son los controladores que deseamos utilizar en los bloques **ControladorVelocidadAngular** y **ControladorVelocidadLineal**.

La arquitectura del archivo Simulink cont_sugeno.slx se mantuvo igual a la de los apartados anteriores ya que utilizamos los mismos 3 sensores para la entrada de los controladores.

Y por último, para comprobar que nuestros nuevos controladores lograban recorrer el circuito con obstáculos con éxito, utilizamos de nuevo el código Ej_Sugeno.m.

Los resultados obtenidos fueron satisfactorios, ya que nuestro robot daba una vuelta completa al circuito sin chocarse con nada y evitando los obstáculos. **Estos resultados pueden observarse en el archivo de vídeo adjuntado en el entregable del proyecto.**

PARTE 2.4: Conclusiones Sugeno

Al igual que en Mamdani, durante el desarrollo de Sugeno generamos un total de **6 controladores neuroborrosos** (3 para velocidad angular y 3 para velocidad lineal):

- **AngSinObs y LinSinObs:** Utilizados en el apartado 2.1 para controlar la velocidad angular y lineal y recorrer el Circuito Sin Obstáculos
- **AngConObs y LinConObs:** Utilizados en el apartado 2.2 para controlar la velocidad angular y lineal y recorrer el Circuito Con Obstáculos 1.
- **AngConObs2 y LinConObs2:** Utilizados en el apartado 2.3 para controlar la velocidad angular y lineal y recorrer el Circuito Con Obstáculos 2.

Del mismo modo que ocurría en Mamdani, **solamente la última pareja de controladores generados es capaz de recorrer los 3 circuitos sin problema**. Si intentásemos utilizar **AngSinObs y LinSinObs** en un circuito con obstáculos nuestro robot acabaría chocando.Del mismo modo, si utilizásemos **AngConObs y LinConObs** en el Circuito Con Obstáculos2 también acabaríamos colisionando nuestro robot.

Controlador neuroborroso	Circuito Sin Obstáculos	Circuito Con Obstáculos	Circuito Con Obstáculos 2
AngSinObs y LinSinObs	**V**	**X**	**X**
AngConObs y LinConObs	**V**	**V**	**X**
AngConObs2 y LinConObs2	**V**	**V**	**V**

Tabla 5: Circuitos que es capaz de recorrer cada controlador.

Por lo tanto, la pareja de controladores definitiva que debemos utilizar son **AngConObs2 y LinConObs2**, ya que serán capaces de realizar los 3 circuitos sin ningún tipo de problema.

PARTE 3: Conclusiones generales

Tras desarrollar completamente todos y cada uno de los aspectos de la práctica hemos llegado a una serie de conclusiones.

Utilizando un controlador borroso de tipo Mamdani hemos logrado dirigir un robot durante 3 circuitos sorteando diferentes obstáculos y evitando chocar con las paredes. **Simplemente modificando las funciones de pertenencia y algunas reglas de nuestro controlador** hemos sido capaces de adaptarnos a cada uno de los circuitos, por lo que hemos aprendido que con ligeros cambios **podemos lograr que nuestro amigobot se adapte a cualquier circuito**.

También hemos logrado generar 3 pares de controladores de tipo Sugeno para recorrer los 3 circuitos. En cuanto a estos controladores, el desarrollo ha sido distinto, ya que no era necesario definir ninguna regla ni función de pertenencia ya que la función anfisedit nos lo hacía de manera automática.

La conclusión que podemos sacar del tipo Sugeno es que **hemos logrado generar controladores neuroborrosos a partir de nuestro propio control manual**, entrenándolos con datos de velocidad angular y lineal y reduciendo el error.

Mientras que el desarrollo de **Mamdani era más cuestión de estudiar las funciones de pertenencia y diseñar las reglas óptimas**, el desarrollo de **Sugeno se ha basado más en realizar prueba y error**, ya que ha sido necesario realizar un gran número de intentos hasta lograr unos datos de entrenamiento que generasen un controlador funcional.