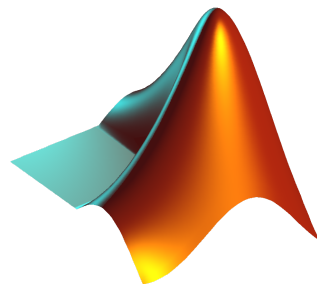




Sistemas de Control Inteligente

**Tema: Práctica Final. Diseño de
Controladores Borrosos y
Neuroborrosos para un Robot Móvil**



Grupo A2_P09

Integrantes

Ana Cortés Cercadillo
Carlos Javier Hellín Asensio

Grado de Ingeniería Informática
Curso: 2021-2022

Índice

Parte 1. Diseño manual de un control borroso de tipo MAMDANI.	3
Diseño de un control borroso capaz de recorrer el entorno sin obstáculos	3
Diseño de un control borroso capaz de recorrer el entorno con obstáculos	11
Parte 2. Diseño automático de un controlador neuroborroso de tipo SUGENO	17
Diseño de un control neuroborroso para recorrer el entorno sin obstáculos	17
Diseño de un control neuroborroso para recorrer el entorno con obstáculos	25
Competición	29

Parte 1. Diseño manual de un control borroso de tipo MAMDANI.

Diseño de un control borroso capaz de recorrer el entorno sin obstáculos

En esta parte de la práctica diseñamos un controlador de la velocidad angular y lineal del robot móvil en un entorno sin obstáculos para recorrer el mismo en el menor tiempo posible a partir de la información obtenida de los sensores de ultrasonidos del robot y la posición y orientación con respecto al entorno. El entorno que se va a usar es el de la Figura 1.

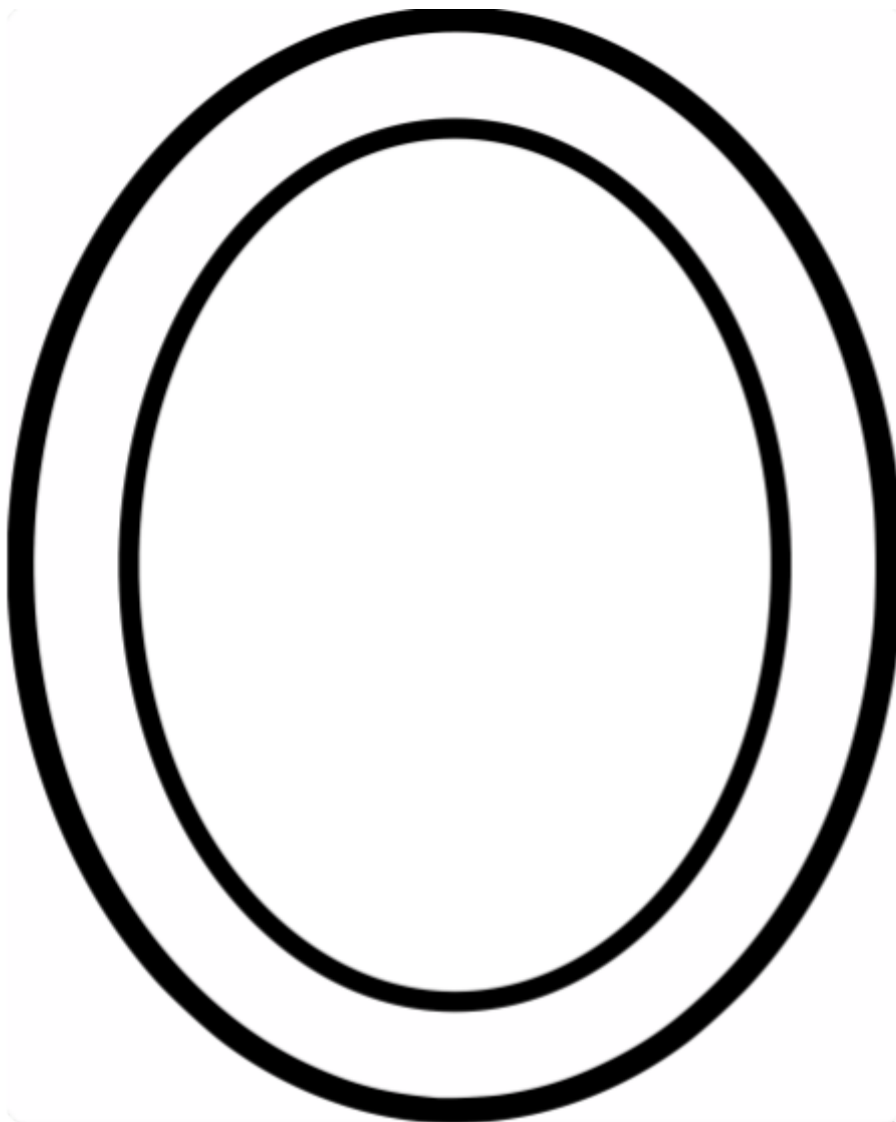


Figura 1. Circuito sin obstáculos.

Para realizar el diseño, haremos uso de la herramienta *fuzzy* de Matlab utilizada en prácticas anteriores (Figura 2).

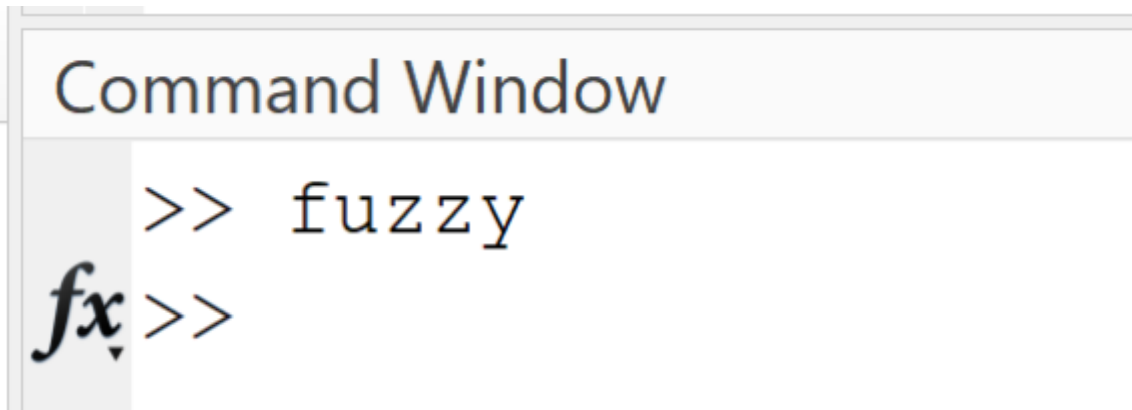


Figura 2. Ejecutando el comando fuzzy.

Al ejecutar el comando anterior en la ventana de comandos de Matlab, aparece la ventana de la Figura 3.

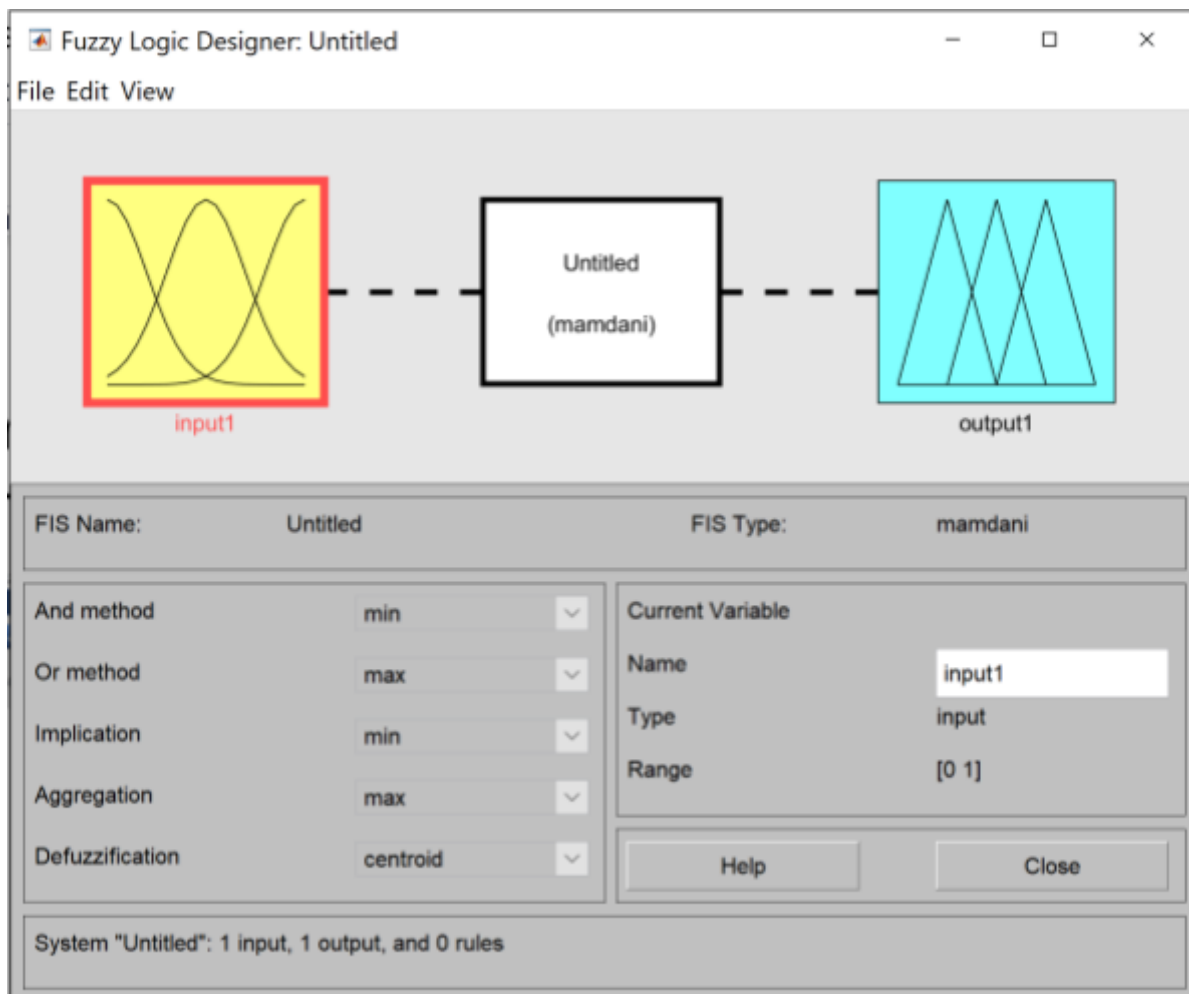


Figura 3. La interfaz de Fuzzy Logic Designer.

A continuación vamos a mostrar y explicar el controlador tipo Mamdani que hemos diseñado para la resolución de este problema. Para ello cargamos el fichero *ControlBorroso.fis* (Figura 4) que se encuentra en la carpeta SinObstaculos de la Parte 1.

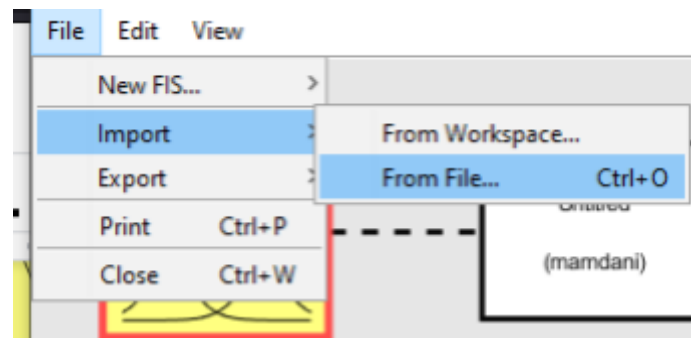


Figura 4. Importar Controlador desde un fichero.

Se nos abre la ventana de la Figura 5 donde podemos observar las entradas, *sensor3*, y las salidas, *V* (velocidad lineal) y *W* (velocidad angular), del controlador.

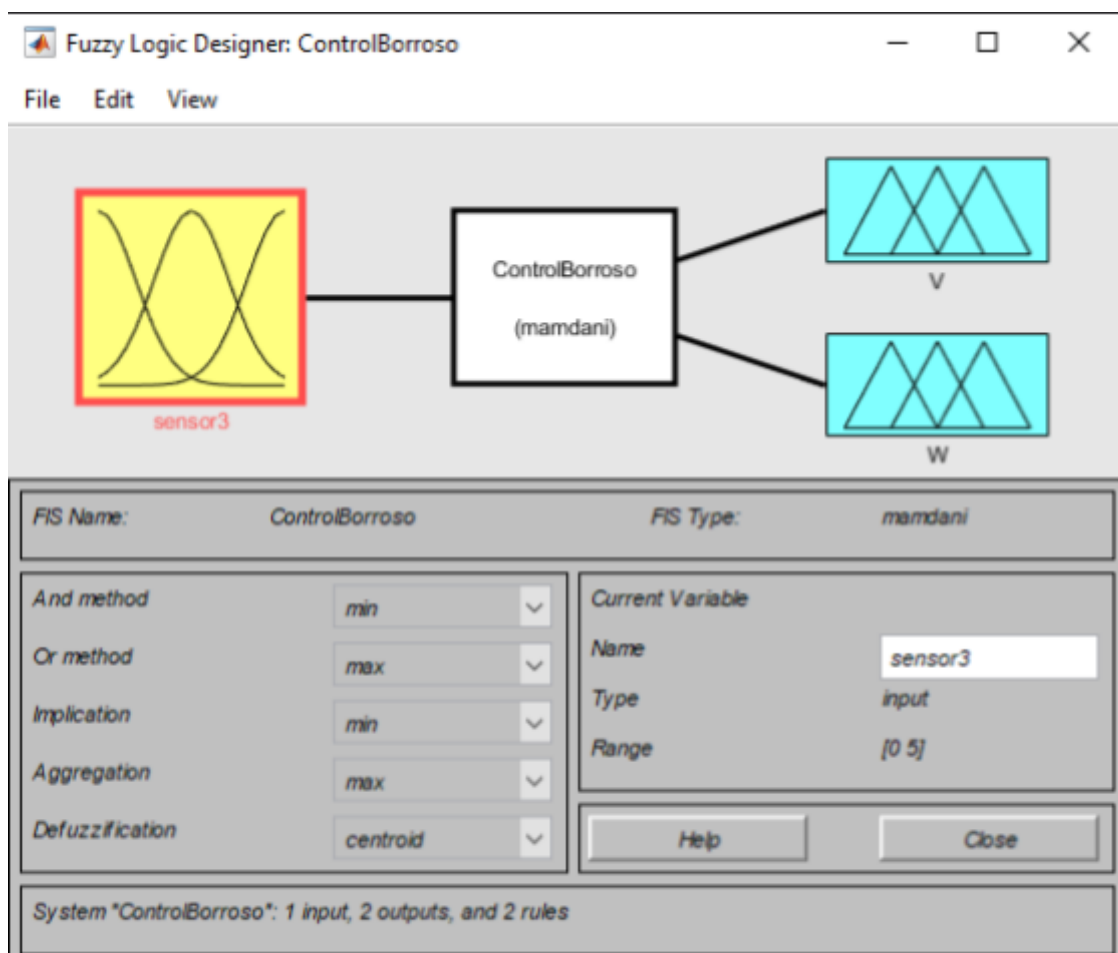


Figura 5. El Control Borroso para sin obstáculos.

Como se ha podido observar en el diseño del controlador borroso usamos el mínimo número de sensores posible, en este caso solo uno, el sensor 3, situado a un ángulo de -15° (Figura 6).

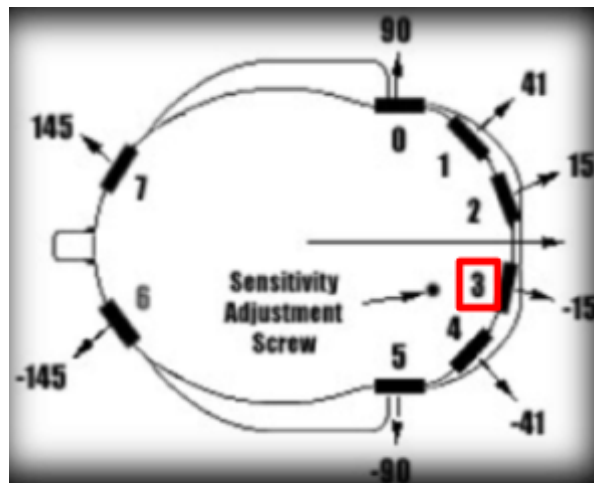
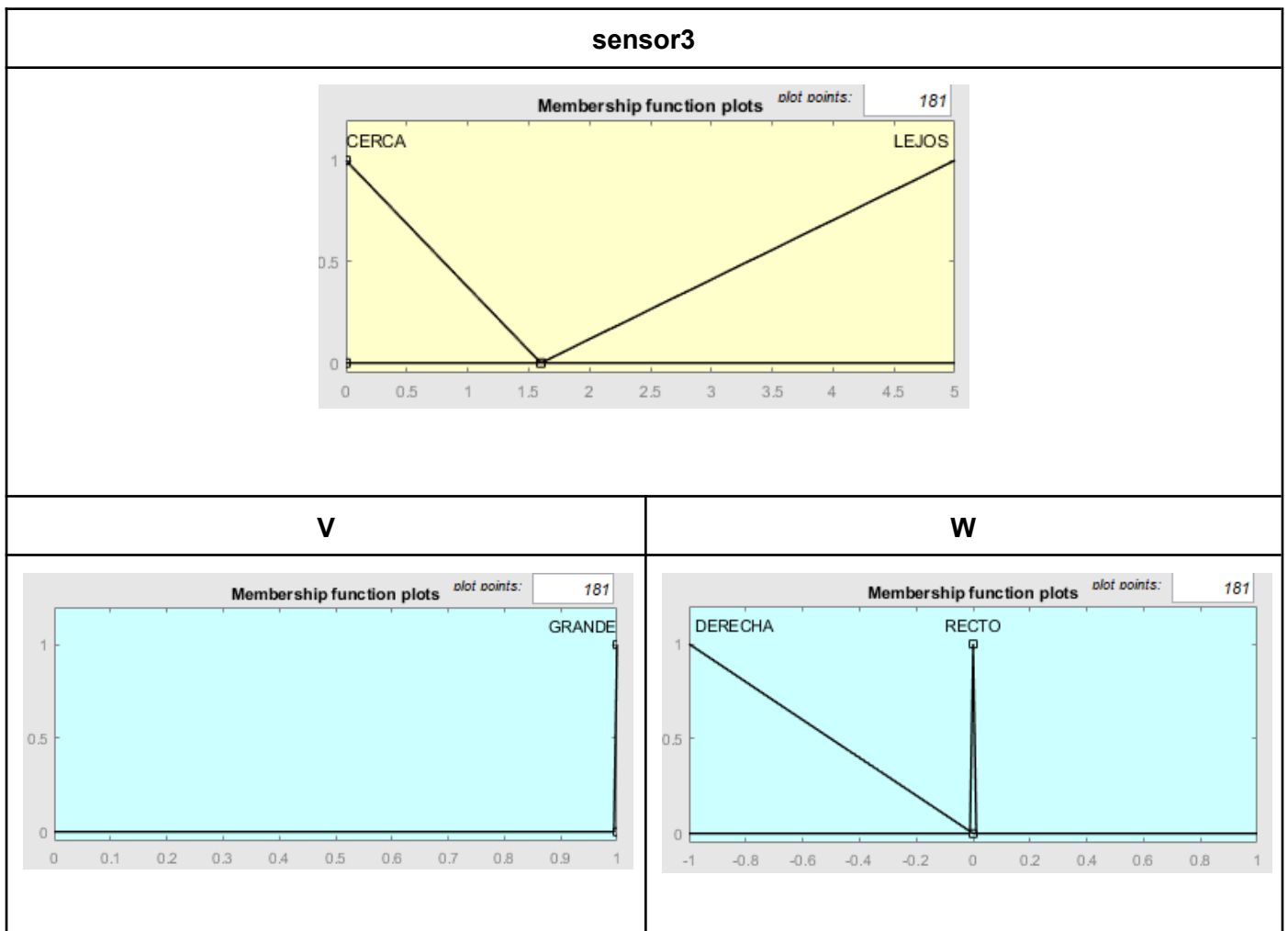


Figura 6. Robot con sus sensores.

Las gráficas de las funciones de cada variable que hemos creado son las siguientes:



Los datos que hemos introducido para diseñar cada variable se describen a continuación:

- **sensor3**: el rango de la variable es [0, 5] y se compone de dos funciones:
 - CERCA: el pico lo alcanza en el valor $x=0$ y desciende hasta $x=1.6$.
 - LEJOS: en $x=1.6$ empieza a subir hasta llegar a su máximo en $x=5$.
- **V**: el rango es [0, 1] y se compone de una sola función:
 - GRANDE: es una función delta con valor $x=1$.
- **W**: el rango es [-1, 1] y tiene dos funciones:
 - DERECHA: el pico lo tiene es $x=-1$ y va bajando hasta $x=0$.
 - RECTO: es una delta con valor $x=0$.

Posteriormente se añaden las reglas mediante el “Rule Editor” que está disponible desde el menú “Edit -> Rules...” (Figura 7). Recordamos también que otra forma de abrir la ventana de las reglas sería haciendo doble click sobre la figura que representa al controlador borroso (cuadrado blanco con bordes negros que pone “mamdani”) en la ventana de la Figura 5.

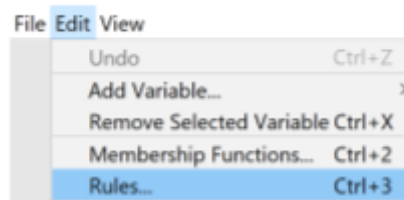


Figura 7. Editar las reglas del Controlador Borroso.

Se han definido las reglas en el controlador que se observan en la Figura 8.

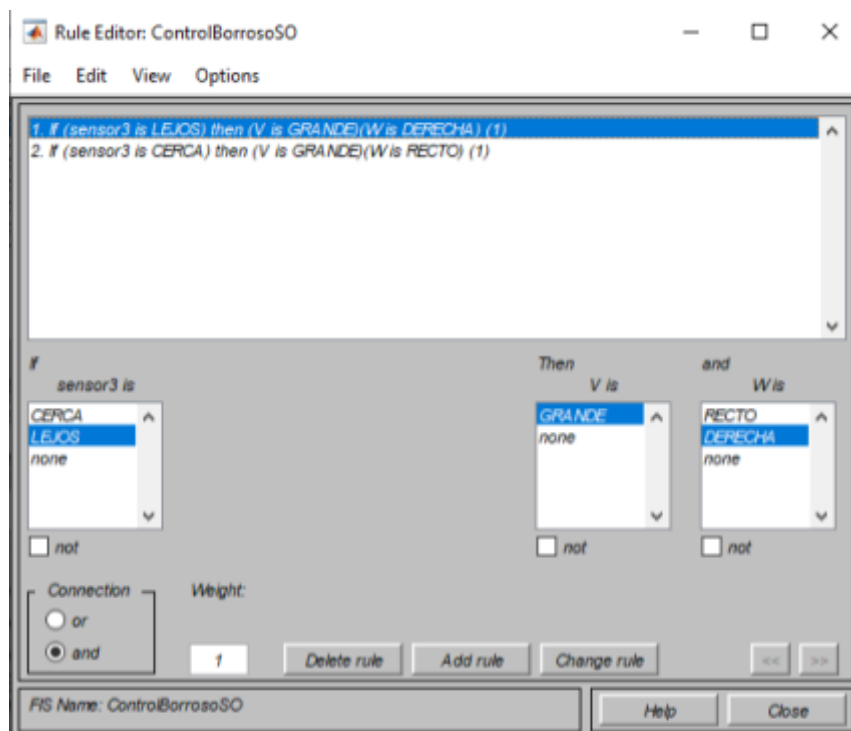


Figura 8. Las reglas definidas del controlador.

Hemos usado dos reglas para controlar al robot:

- La regla número 1 quiere decir que cuando el sensor 3 del robot capte la pared lejos que vaya a una velocidad grande (1 m/s) y que gire a la derecha.
- La regla número 2 implica que cuando el sensor 3 del robot esté cerca de la pared, se corrige la trayectoria yendo recto y manteniendo una velocidad grande.

El siguiente paso es modificar el esquema *test_controler_generic.slx* de la Figura 9.

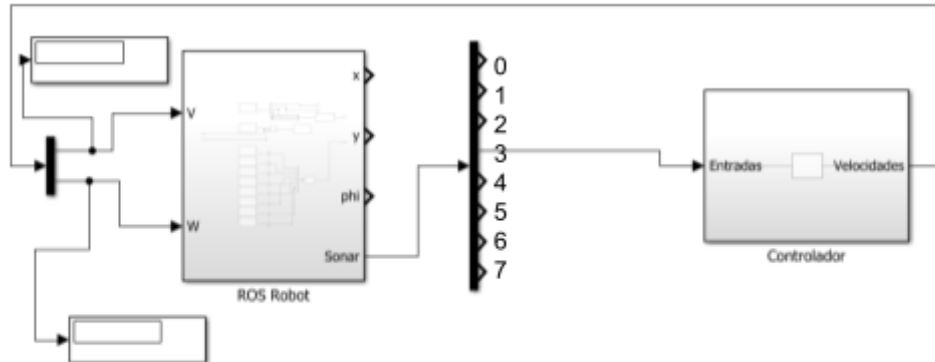


Figura 9. Esquema final con el Controlador Borroso.

En la entrada del controlador se hace la conexión con la salida del sensor 3. También hemos añadido dos displays para ir controlando los valores de V y W según se va ejecutando el controlador.

Hay que recordar añadir el nombre del fichero *ControlBorroso.fis* dentro del Controlador (Figura 9) en el bloque Fuzzy Logic Controller (Figura 10).

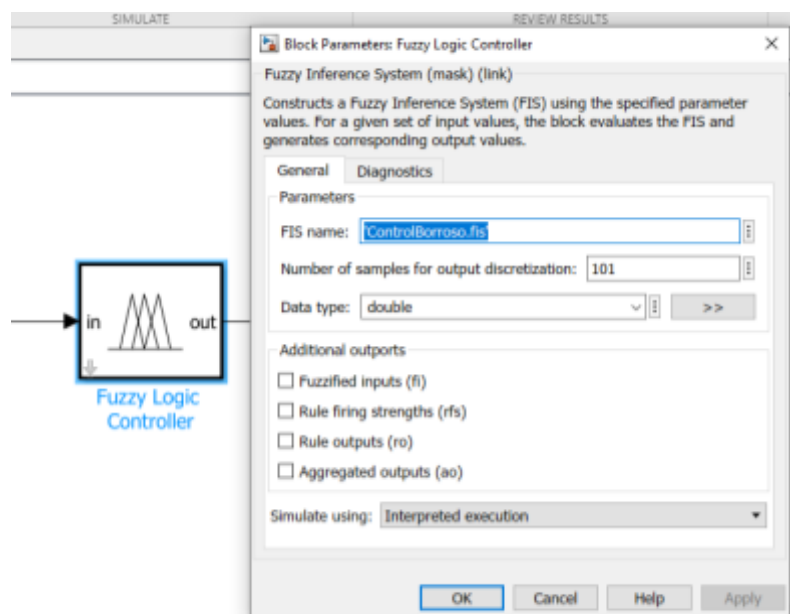


Figura 10. Cambiar el fichero FIS del Controlador Borroso.

No hay que olvidar también comprobar que la dirección IP es la de la máquina virtual donde va a ejecutarse el recorrido del robot. Esta dirección se debe introducir en el bloque *ROS Robot* de la Figura 9 dentro de cualquier bloque *Subscribe* que pertenezca al *robot0*. Allí hay que pulsar la opción *Configure network addresses* (Figura 11). Se abre una nueva ventana donde podemos cambiar la dirección IP en el apartado *Hostname/IP Address* (Figura 12). Se puede consultar la dirección IP de la máquina virtual escribiendo en su Terminal el siguiente comando: `ifconfig`.

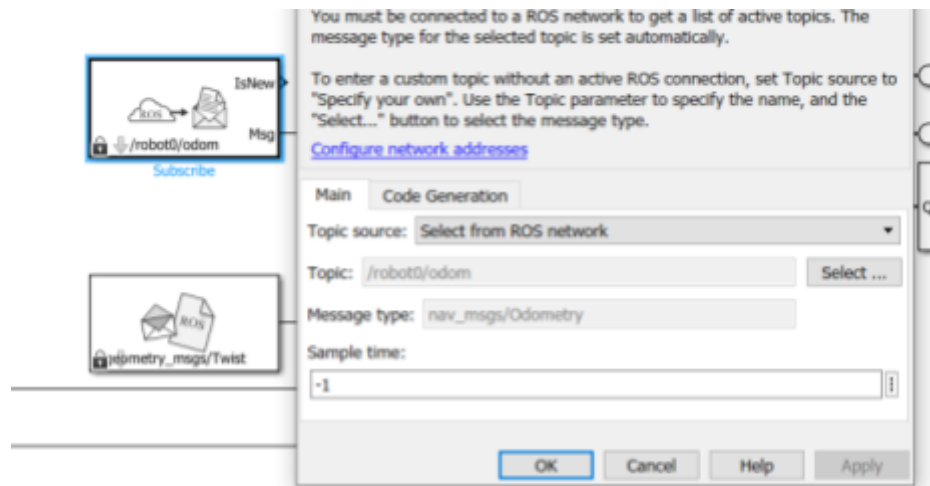


Figura 11. Configurar la dirección de red.

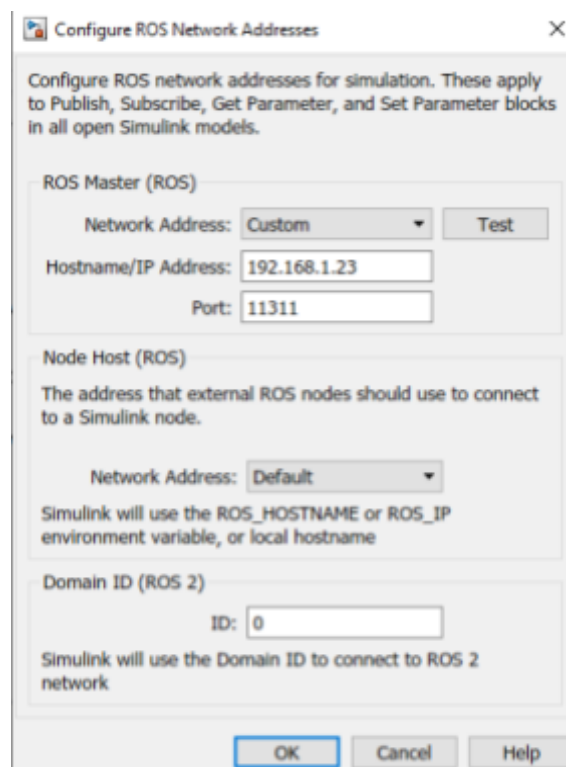


Figura 12. Cambiar la IP.

Otro paso a tener en cuenta antes de probar la ejecución del controlador es, dentro de la máquina virtual, comprobar el contenido de los ficheros *EntornoPracticaFinal.yaml* y *PracticaFinal.launch*. En el primero debe de aparecer la imagen del mapa que se vaya a usar en este caso *OvaloSinObstaculos.png* (Figura 1) como se muestra en la Figura 13.

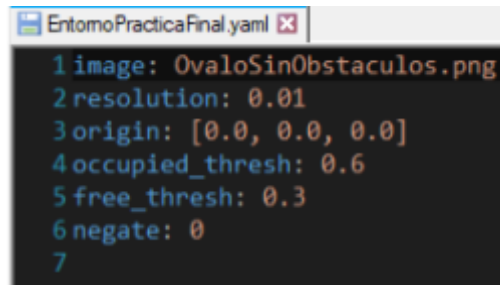


Figura 13. Fichero de configuración del entorno.

En el segundo fichero no hay que realizar ninguna modificación, solamente comprobar los números de la posición inicial del robot, en este caso *10 1.5 3.14* (Figura 14).

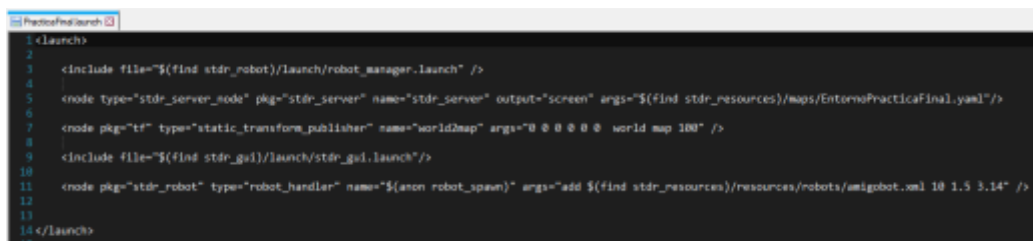


Figura 14. El fichero .launch.

Una vez diseñado el controlador y configurados los parámetros necesarios para un correcto funcionamiento, mostramos cómo el robot realiza el recorrido en el siguiente video:

- <https://www.youtube.com/watch?v=Wp3PXAoKAvQ>

Diseño de un control borroso capaz de recorrer el entorno con obstáculos

En esta parte de la práctica diseñamos un controlador de la velocidad angular y lineal del robot móvil en un entorno con obstáculos para recorrer el mismo en el menor tiempo posible a partir de la información obtenida de los sensores de ultrasonidos del robot y la posición y orientación con respecto al entorno. El entorno que se va a usar es el de la Figura 15.

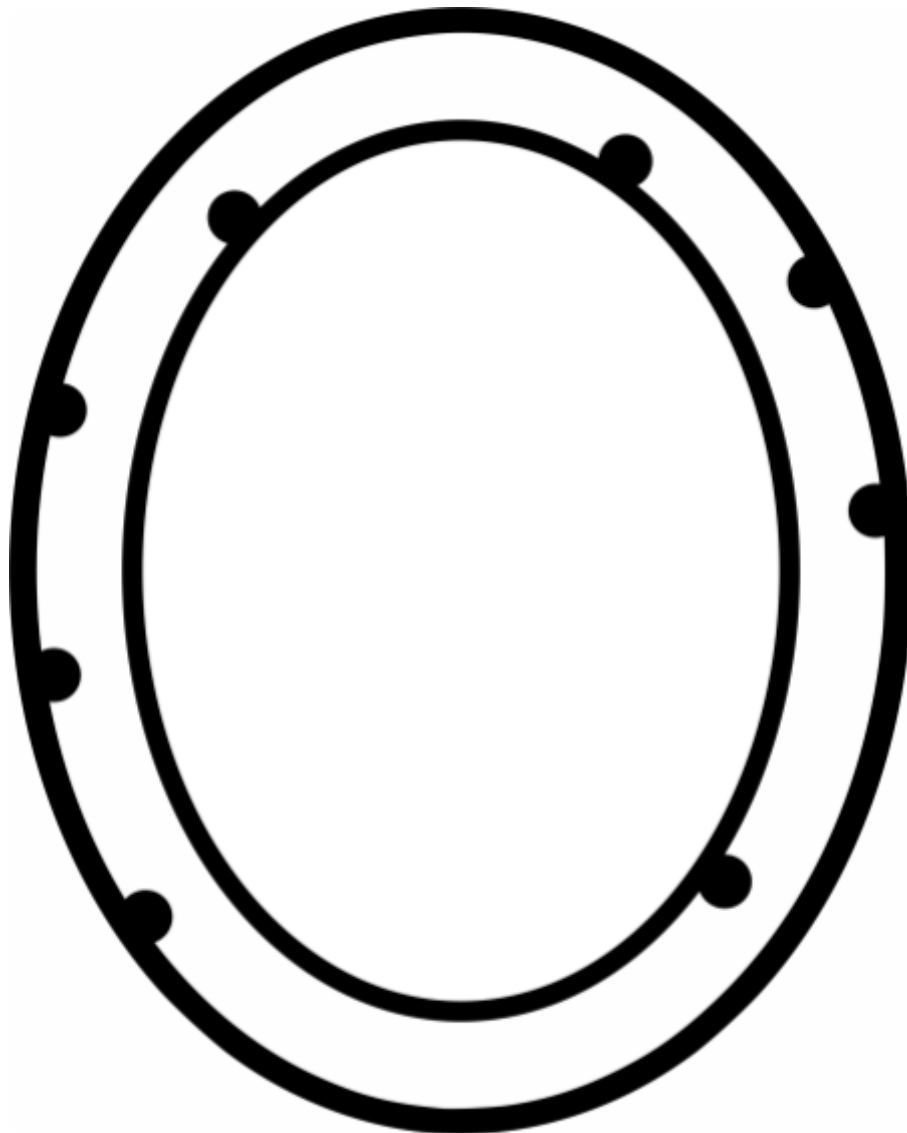


Figura 15. Circuito con obstáculos (mapa definitivo)

Ahora, como se ha hecho en el apartado sin obstáculos, vamos a mostrar y explicar el controlador tipo Mamdani que hemos diseñado para la resolución de este problema. Para ello cargamos el fichero *ControlBorroso.fis* que se encuentra en la carpeta ConObstaculos de la Parte 1 de igual forma que antes. Se nos abre la ventana de la Figura 16 donde podemos observar las entradas, *sensor1* y *sensor4*, y las salidas, *V* (velocidad lineal) y *W* (velocidad angular), del controlador.

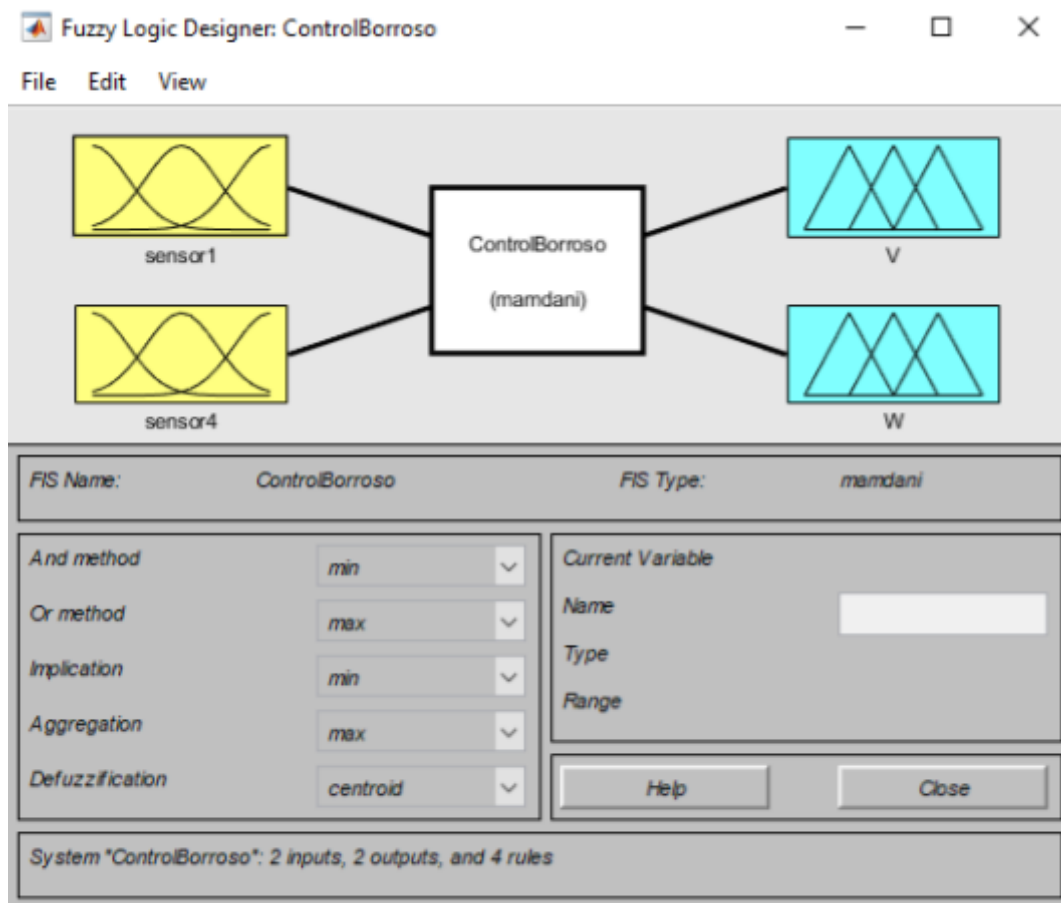


Figura 16. El Control Borroso para con obstaculos.

En el diseño del controlador borroso usamos el mínimo número de sensores posible que en este caso, usamos dos, el sensor 1 y el sensor 4, situado a un ángulo de 41° y -41° respectivamente (Figura 6).

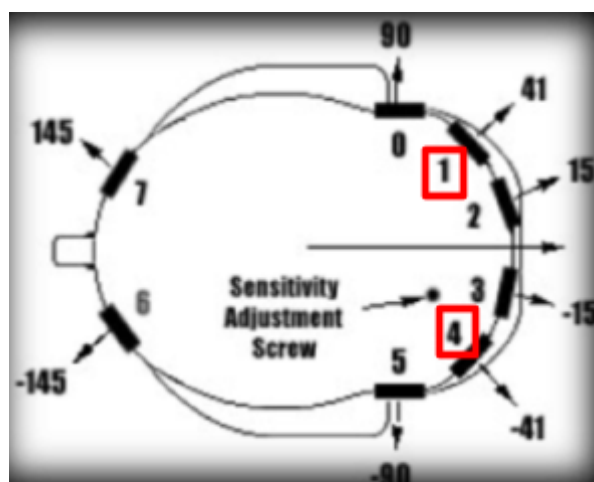
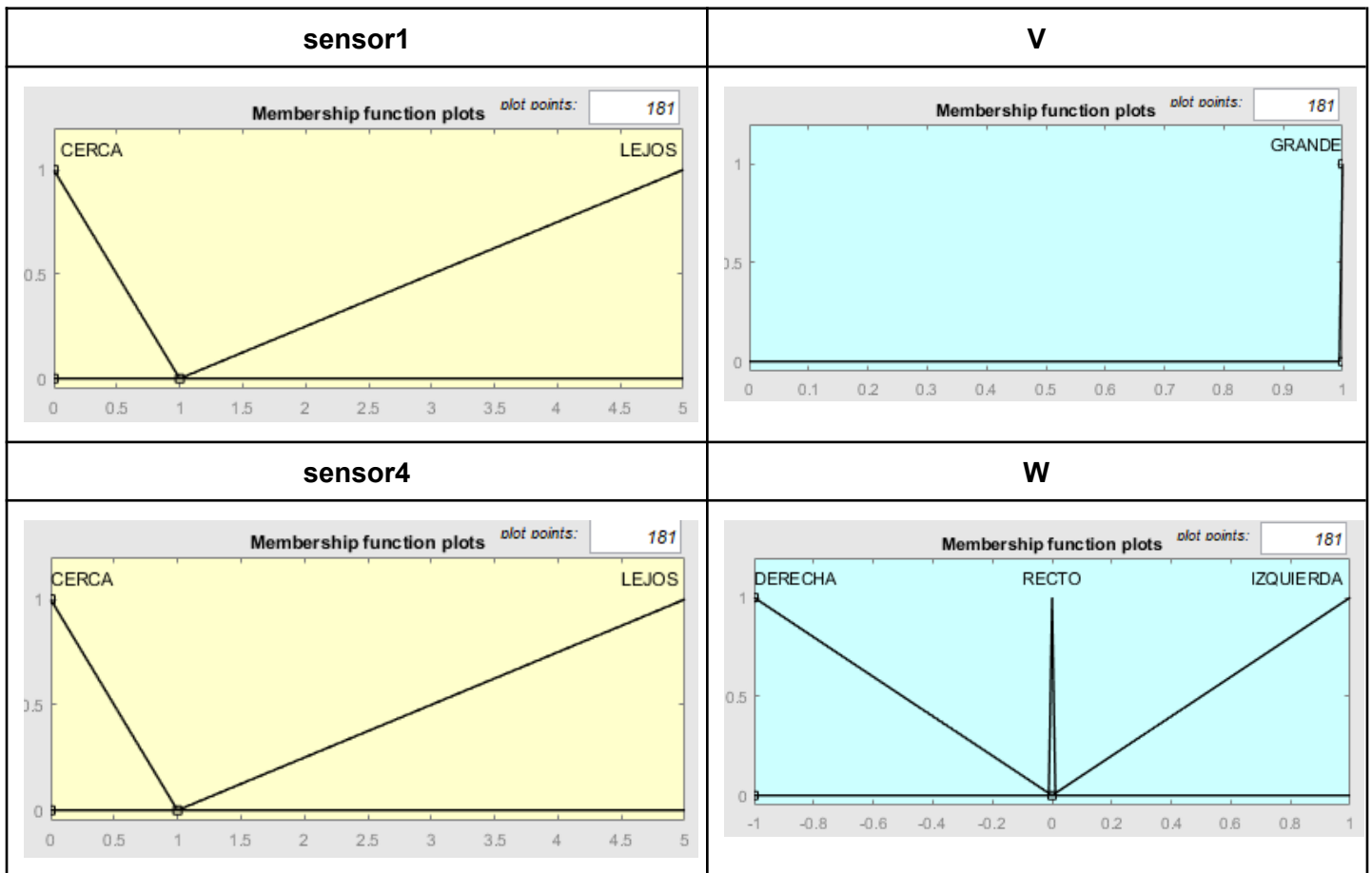


Figura 17. Robot con sus sensores.

Las gráficas de las funciones de cada variable que hemos creado son las siguientes:



Los datos que hemos introducido para diseñar cada variable se describen a continuación:

- **sensor1**: el rango de la variable es $[0, 5]$ y se compone de dos funciones:
 - CERCA: el pico lo alcanza en el valor $x=0$ y desciende hasta $x=1$.
 - LEJOS: en $x=1$ empieza a subir hasta llegar a su máximo en $x=5$.
- **sensor4**: el rango de la variable es $[0, 5]$ y se compone de dos funciones:
 - CERCA: el pico lo alcanza en el valor $x=0$ y desciende hasta $x=1$.
 - LEJOS: en $x=1$ empieza a subir hasta llegar a su máximo en $x=5$.
- **V**: el rango es $[0, 1]$ y se compone de una sola función:
 - GRANDE: es una función delta con valor $x=1$.
- **W**: el rango es $[-1, 1]$ y tiene dos funciones:
 - DERECHA: el pico lo tiene es $x=-1$ y va bajando hasta $x=0$.
 - RECTO: es una delta con valor $x=0$.

Las reglas que se han definido en el controlador se observan en la Figura 18.

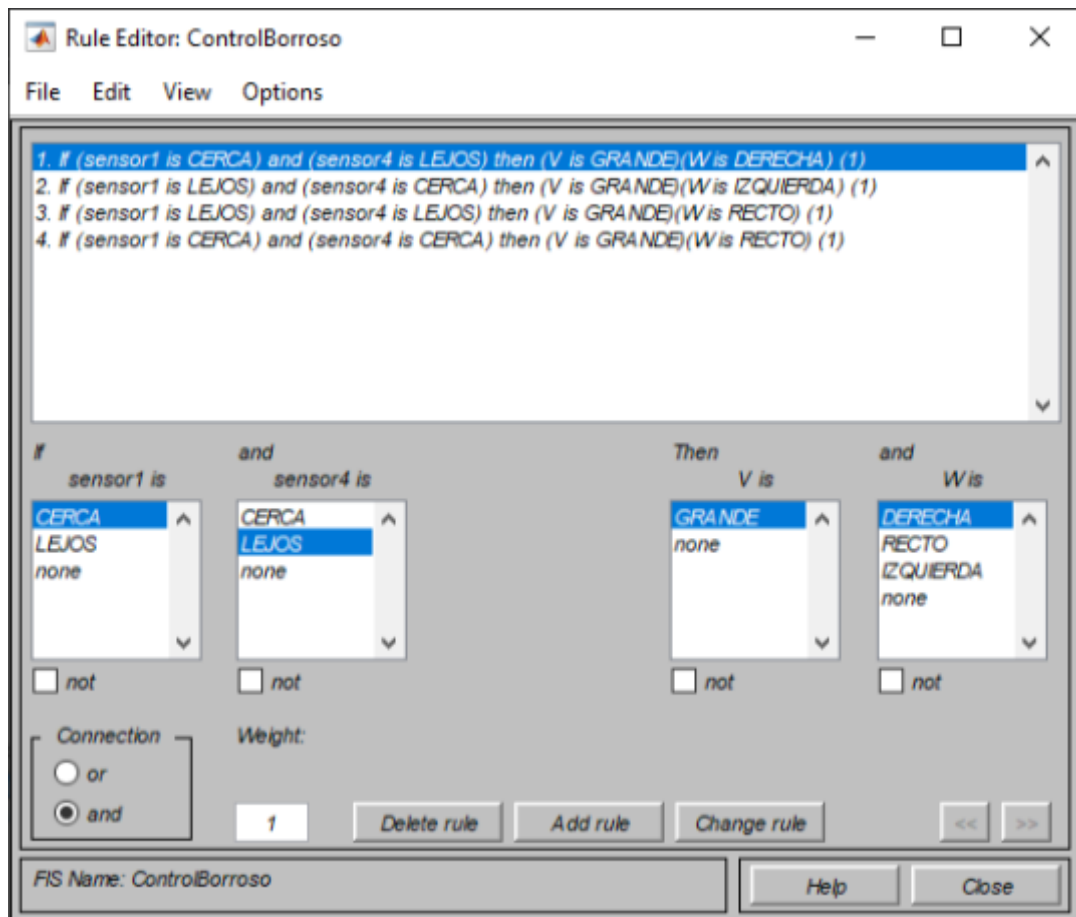


Figura 18. Las reglas definidas del controlador.

Hemos usado cuatro reglas para controlar al robot:

- La regla número 1 quiere decir que cuando el sensor 1 del robot capte la pared o un obstáculo cerca y el sensor 4 capte algo lejos que vaya a una velocidad grande (1 m/s) y que gire a la derecha.
- La regla número 2 implica que cuando el sensor 1 del robot esté lejos de un obstáculo y el sensor 4 cerca, se corrige la trayectoria girando a la izquierda y manteniendo una velocidad grande.
- La regla número 3 ayuda a que si el sensor 1 y el sensor 4 se encuentran lejos de un obstáculo, pueda mantener una velocidad grande e ir recto.
- La regla número 4 quiere decir que cuando el sensor 1 y el sensor 4 del robot capten la pared o un obstáculo cerca que vaya a una velocidad grande y que continúe recto.

El siguiente paso es modificar el esquema *test_controler_generic.slx* de la Figura 19.

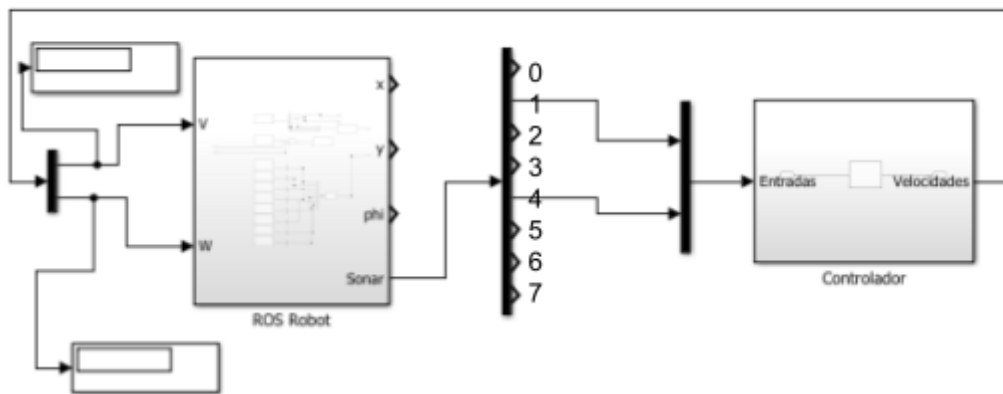


Figura 19. Esquema final con el Control Borroso.

En la entrada del controlador se hace la conexión con las salidas de los sensores 1 y 4. Los displays se siguen manteniendo. Al igual que antes se añade el nombre del fichero `ControlBorroso.fis` dentro del Controlador (Figura 19) en el bloque Fuzzy Logic Controller (Figura 10) y se comprueba la IP de la máquina virtual (Figura 12).

Finalmente comprobamos el contenido de los ficheros `EntornoPracticaFinal.yaml` y `PracticaFinal.launch`. En el primero debe de aparecer la imagen del mapa que vamos a usar `SCI_2122_mapa_definitivo.png` (Figura 15) como se muestra en la Figura 20.

```
EntornoPracticaFinal.yaml
1 image: SCI_2122_mapa_definitivo.png
2 resolution: 0.01
3 origin: [0.0, 0.0, 0.0]
4 occupied_thresh: 0.6
5 free_thresh: 0.3
6 negate: 0
7
```

Figura 20. Cambio de mapa en el fichero `EntornoPracticasFinal.yaml`

En el segundo fichero hay que modificar los números de la posición inicial del robot, en este caso son 7 2.1 2.5 (Figura 21).

```
args="add $(find stdr_resources)/resources/robots/amigobot.xml 7 2.1 2.5" />
```

Figura 21. Cambio de posición del robot en el fichero `PracticaFinal.launch`

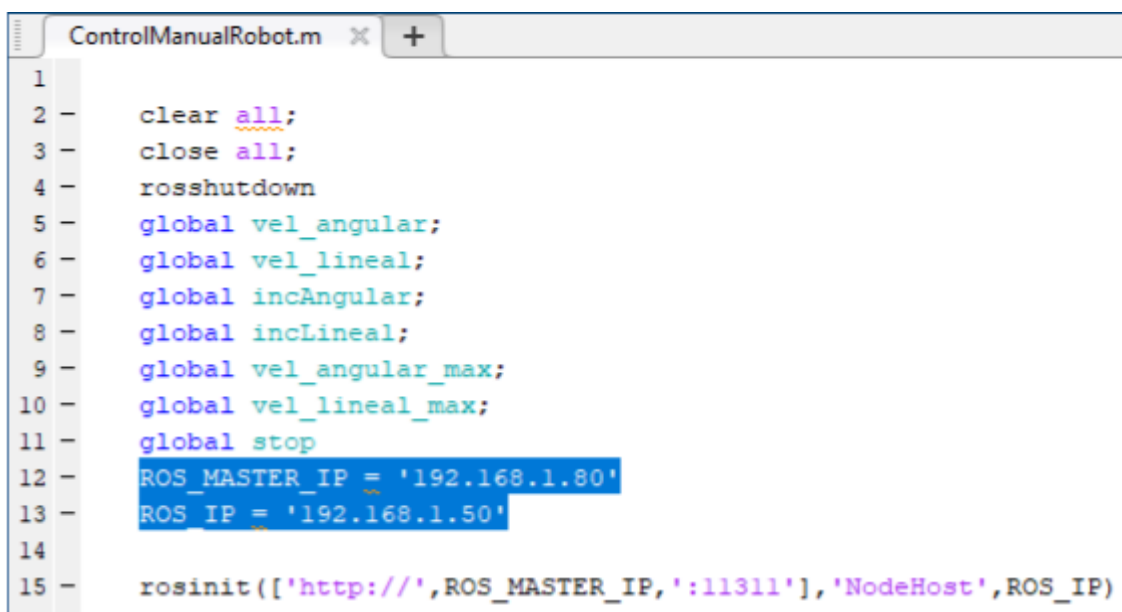
Una vez diseñado el controlador y configurados los parámetros necesarios para un correcto funcionamiento, mostramos cómo el robot realiza el recorrido en el siguiente video:

- <https://www.youtube.com/watch?v=5lhJNkv2mWg>

Parte 2. Diseño automático de un controlador neuroborroso de tipo SUGENO

Diseño de un control neuroborroso para recorrer el entorno sin obstáculos

Para la realización de esta parte se dispone de una interfaz a través del teclado (script de Matlab *ControlManualRobot.m* incluido en los archivos de la práctica) con la que podemos controlar manualmente el robot para que éste recorra el circuito y en el que se registrará la velocidad angular, lineal, posición, orientación y lecturas de los sensores. Para la ejecución del control manual se deberán configurar correctamente las direcciones IP de la máquina virtual (*ROS_MASTER_IP*) y la IP de la máquina (*ROS_IP*) en la primera línea de código de *ControlManualRobot.m* (Figura 22).



```
1
2 - clear all;
3 - close all;
4 - rosshutdown
5 - global vel_angular;
6 - global vel_lineal;
7 - global incAngular;
8 - global incLineal;
9 - global vel_angular_max;
10 - global vel_lineal_max;
11 - global stop
12 - ROS_MASTER_IP = '192.168.1.80'
13 - ROS_IP = '192.168.1.50'
14
15 - rosinit(['http://',ROS_MASTER_IP,':11311'],'NodeHost',ROS_IP)
```

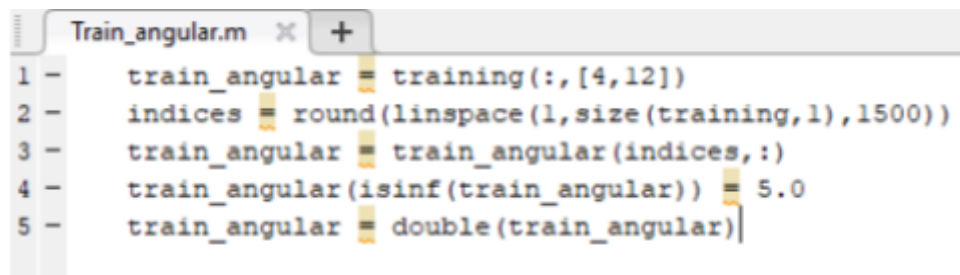
Figura 22. Configuración de las direcciones IP.

Una vez iniciado el simulador STDR, se ejecuta el script *ControlManualRobot.m* desde la consola de Matlab. Para el control manual de las velocidades del robot se utilizan las teclas de dirección DERECHA e IZQUIERDA, para incrementar y decrementar respectivamente en 0.1 rad/s la velocidad angular del robot y las teclas de dirección ARRIBA y ABAJO para incrementar y decrementar respectivamente en 0.1 m/s la velocidad lineal del robot. Una vez terminado el entrenamiento, mediante la pulsación de la barra espaciadora, se da por finalizado el control manual y se guarda el contenido de la variable *training* en un fichero llamado *datos_entrenamiento.mat* debido a la siguiente línea de código: `save datos_entrenamiento training`.

En ese fichero se guarda la información de las filas de la matriz *training* que se han grabado cada 0.1 segundos durante la ejecución. En la siguiente línea de código es donde se va creando dicha matriz: `training = [training; [s0, s1, s2, s3, s4, s5, s6,`

s7, x, y, theta, vel_angular, vel_lineal]]. De esta forma se recogen los valores de los diferentes sonares (del 0 al 7), de la posición del robot (x e y), su orientación (theta), la velocidad angular (vel_angular) y velocidad lineal (vel_lineal).

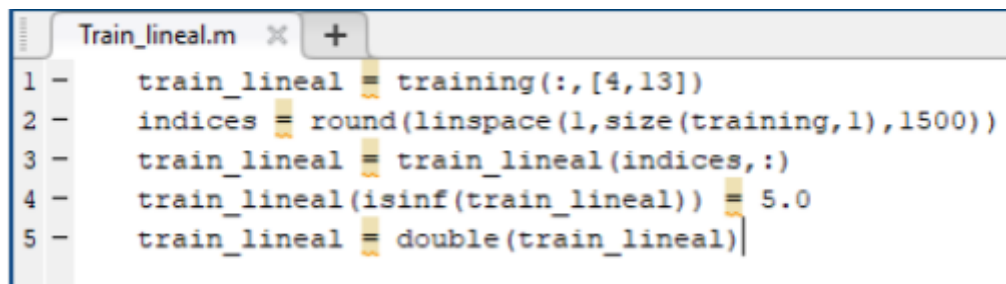
Vamos a usar la variable *training* para crear los controladores de velocidad lineal y angular del robot mediante la herramienta *Anfisedit*. En este caso creamos dos controladores independientes para la velocidad lineal y angular del robot. También se recomienda reducir el número de filas de la matriz para que contenga un máximo aproximado de 1500 filas. Necesitamos obtener solamente la información que vamos a usar después para diseñar cada controlador, por lo tanto, usamos los siguientes scripts para obtener las columnas que nos interesan:

The image shows a MATLAB script editor window titled 'Train_angular.m'. It contains five lines of code: 1. train_angular = training(:, [4, 12]); 2. indices = round(linspace(1, size(training, 1), 1500)); 3. train_angular = train_angular(indices, :); 4. train_angular(isinf(train_angular)) = 5.0; 5. train_angular = double(train_angular);

```
1 - train_angular = training(:, [4, 12])
2 - indices = round(linspace(1, size(training, 1), 1500))
3 - train_angular = train_angular(indices, :)
4 - train_angular(isinf(train_angular)) = 5.0
5 - train_angular = double(train_angular)
```

Figura 23. Script para obtener la variable train_angular.

En el caso de la velocidad angular, el script Train_angular.m (Figura 23), solo nos quedamos con las columnas 4 y 12 de la matriz *training* (sensor 3 y velocidad angular). También reducimos el número de filas a 1500 y sustituimos los valores infinitos por el valor máximo permitido (5.0 m).

The image shows a MATLAB script editor window titled 'Train_lineal.m'. It contains five lines of code: 1. train_lineal = training(:, [4, 13]); 2. indices = round(linspace(1, size(training, 1), 1500)); 3. train_lineal = train_lineal(indices, :); 4. train_lineal(isinf(train_lineal)) = 5.0; 5. train_lineal = double(train_lineal);

```
1 - train_lineal = training(:, [4, 13])
2 - indices = round(linspace(1, size(training, 1), 1500))
3 - train_lineal = train_lineal(indices, :)
4 - train_lineal(isinf(train_lineal)) = 5.0
5 - train_lineal = double(train_lineal)
```

Figura 24. Script para obtener la variable train_lineal.

En el caso de la velocidad lineal, el script Train_lineal.m (Figura 24), nos quedamos con las columnas 4 y 13 de la matriz *training* (sensor 3 y velocidad lineal). De la misma forma, reducimos el número de filas a 1500 y sustituimos los valores infinitos por el valor máximo permitido (5.0 m).

Estas variables se guardan en el Workspace de Matlab (Figura 25). Se puede consultar los datos que hemos usado en la carpeta SinObstaculos dentro de Parte 2, la variable *training* en el fichero *training.mat*, la variable *train_angular* en *train_angular.mat* y *train_lineal* en *train_lineal.mat*.

Workspace	
Name	Value
train_angular	1433x2 double
train_lineal	1500x2 double
training	2291x13 single

Figura 25. Workspace con todas las variables de entrenamiento.

El proceso de generación de los controladores neuroborrosos pasa por las siguientes etapas:

1. Ejecutar la herramienta Anfisedit. Se escribe el comando *anfisedit* en la ventana de comandos de Matlab (Figura 26). Se abrirá la ventana de Neuro-Fuzzy Designer de la Figura 27.

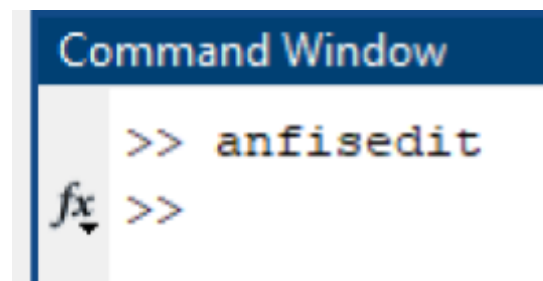


Figura 26. Ejecutando el comando anfisedit.

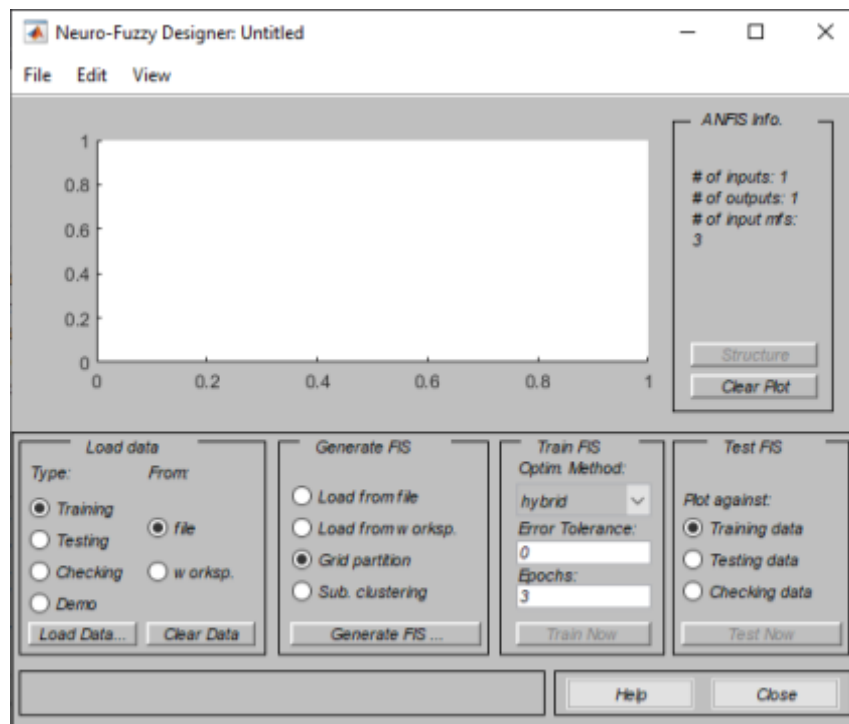


Figura 27. La interfaz de Neuro-Fuzzy Designer.

2. Cargar los datos de entrenamiento/training. En el apartado *Load data* de la interfaz se selecciona la opción *Training* de *Type* y la opción *worksp.* de *From* para cargar los datos desde el workspace. Se pulsa *Load Data...* y se escribe el nombre de la variable con la información, en este caso, *train_angular* (Figura 28). El resultado es la Figura 29.

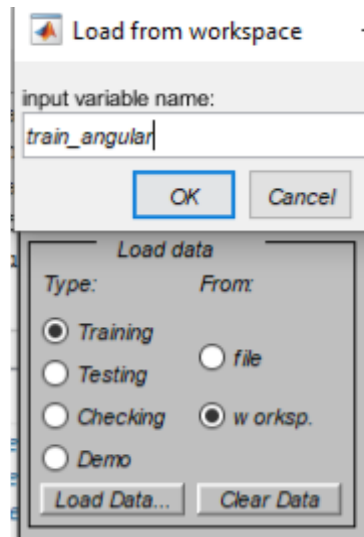


Figura 28. Cargar datos desde una variable del workspace

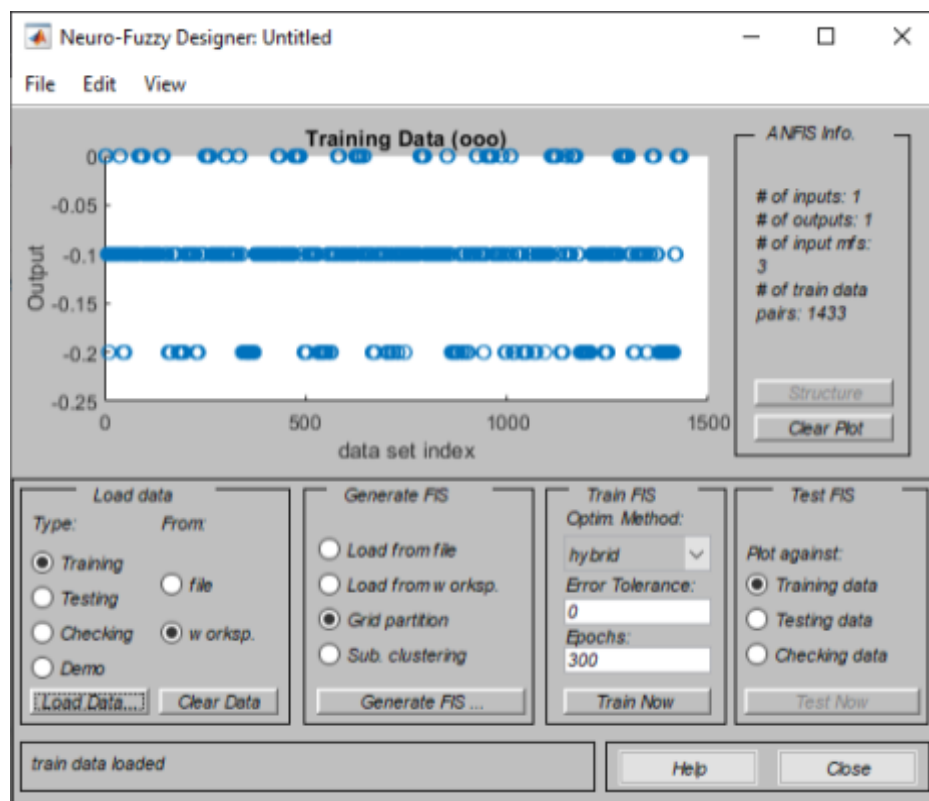


Figura 29. Datos ya cargados de la variable *train_angular*.

3. Generar el FIS. En el apartado *Generate FIS* se pulsa en el botón *Genarate FIS...*, seleccionando el método *Grid partition* (Figura 29). Al presionar este botón, aparece una ventana que permite configurar el número y tipo de funciones miembro de entrada y tipo de funciones miembro de salida, tal y como se muestra en la Figura 30. Nosotros hemos elegido en la entrada (*INPUT*) un número de funciones (*Number of MFs*) de 3 y el tipo de función (*MF Type*) *trimf*. En la salida (*OUTPUT*) el tipo de función (*MF Type*) será constante (*constant*).

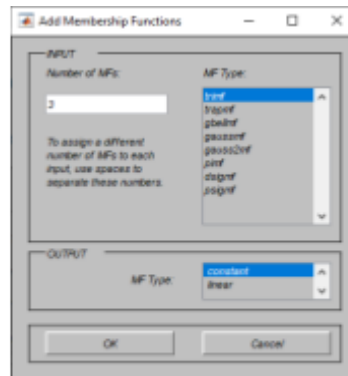


Figura 30. Configuración para generar el FIS.

4. Entrenar el sistema. En el apartado *Train FIS* hemos elegido como *Optim. Method* la opción *hybrid*, la tolerancia de error (*Error Tolerance*) la mantenemos en 0 y en el número de *Epochs* ponemos 300. Decidimos este número de Epochs porque es cuando el entrenamiento no disminuye su error y se mantiene constante. Una vez configurados todos los parámetros pulsamos *Train now* y se espera hasta que termine de entrenar. Cuando llega al epoch número 300 el entrenamiento ha finalizado con un error de 0.053111 y se muestra la ventana de la Figura 31.

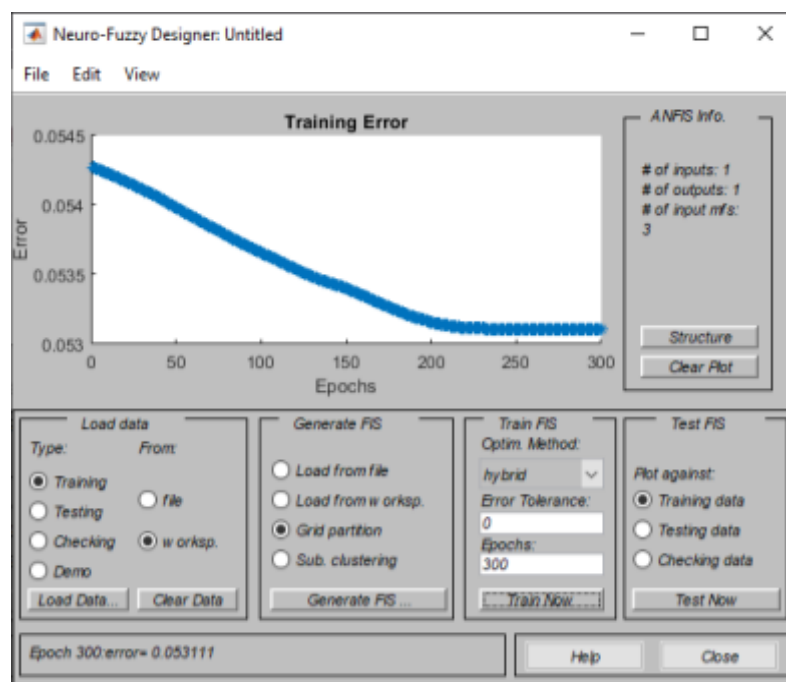


Figura 31. Entrenamiento terminado.

- Comprobar el controlador generado. En el apartado *Test FIS* se comprueba que *Plot against* tiene la opción *Training data* seleccionada y se pulsa el botón *Test Now*. A continuación se cambia la interfaz a la de la Figura 32 lo que significa que ha terminado su comprobación.

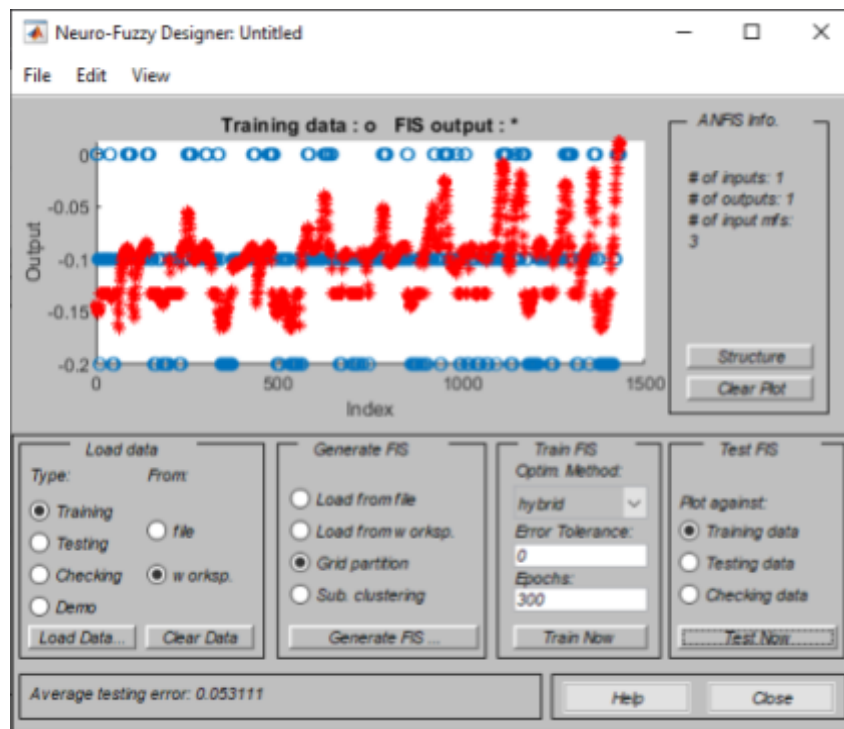


Figura 32. Comprobación terminada.

- Guardar el controlador. Una vez diseñado el controlador, es necesario guardarlo (File/Export/To File) como en la Figura 33, en nuestro caso con el nombre *ControlNBW.fis*.

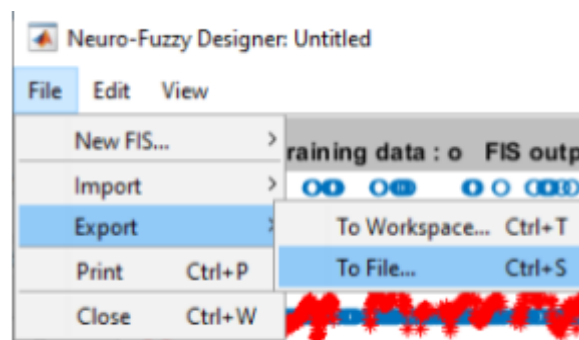


Figura 33. Exportar el controlador.

Para el controlador de la velocidad lineal se siguen los mismos pasos:

- Ejecutar la herramienta Anfisedit.
- Cargar los datos de entrenamiento/training. Seleccionando las mismas opciones que anteriormente aunque en este caso el nombre de la variable es *train_lineal*.

3. Generar el FIS. De la misma forma que para el controlador de la velocidad angular.
4. Entrenar el sistema. En el número de *Epochs* ahora pondremos 150. Cuando el entrenamiento ha finalizado se obtiene un error de 0.1641.
5. Comprobar el controlador generado. El resultado final del entrenamiento es el de la Figura 34.
6. Guardar el controlador. El nombre del fichero será *ControlNBV.fis*.

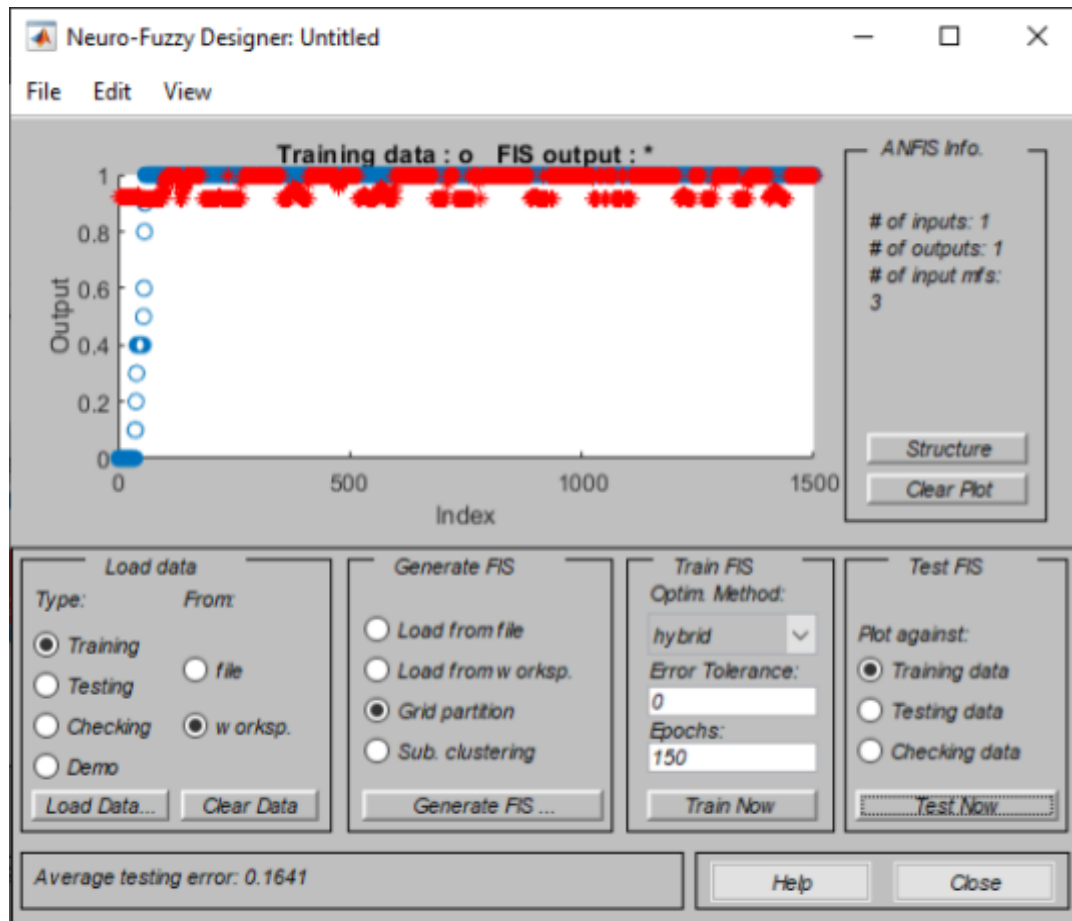


Figura 34. Resultado final del entrenamiento lineal.

Para terminar, modificamos el esquema *test_controler_generic.slx* de la Figura 35. Ahora tenemos dos controladores uno para cada variable (V y W). En la entrada del controlador de velocidad lineal (ControladorV) se hace la conexión con la salida del sensor 3, al igual que la entrada del controlador de velocidad angular (ControladorW). Los displays se siguen manteniendo. Al igual que anteriormente se añade el nombre del fichero *ControlNBV.fis* dentro del ControladorV en el bloque Fuzzy Logic Controller y el fichero *ControlNBW* dentro del ControladorW en el bloque Fuzzy Logic Controller. Finalmente se comprueba la IP de la máquina virtual (Figura 12).

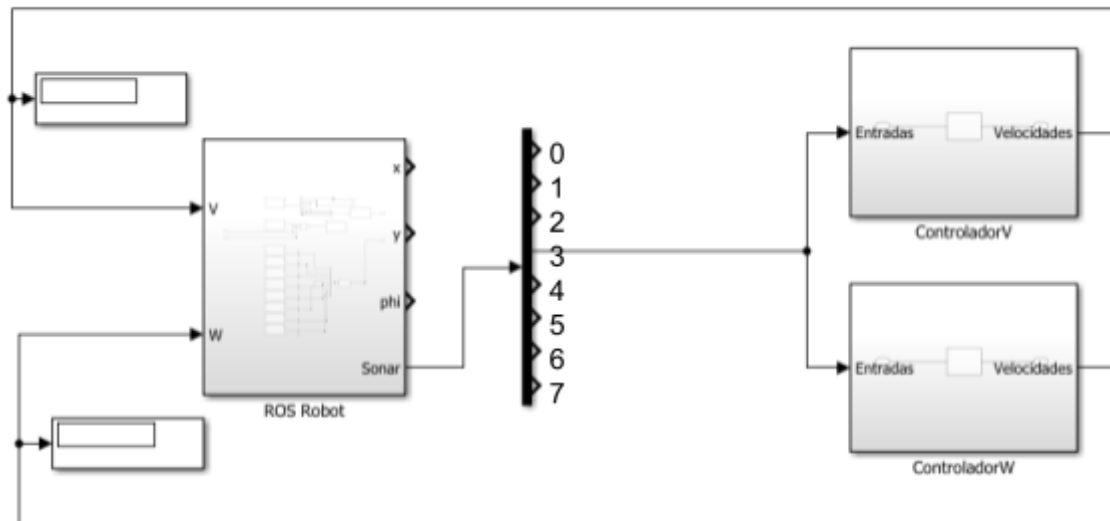


Figura 35. Esquema final con los dos Controladores Neuroborroso.

Hay que recordar comprobar el contenido de los ficheros *EntornoPracticaFinal.yaml* y *PracticaFinal.launch* dentro de la máquina virtual. Ambos ficheros deben de ser iguales que en el apartado “Diseño de un control borroso capaz de recorrer el entorno sin obstáculos” de la Parte 1 de esta práctica (Figura 13 y 14).

Una vez diseñados los controladores y configurados los parámetros necesarios para un correcto funcionamiento, mostramos cómo el robot realiza el recorrido en el siguiente video:

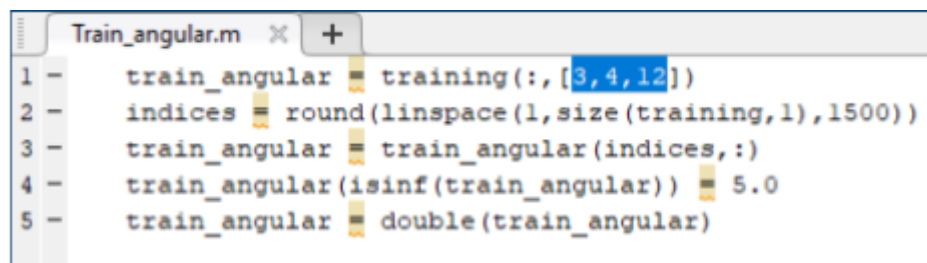
- <https://www.youtube.com/watch?v=L02WO8gTH9U>

Diseño de un control neuroborroso para recorrer el entorno con obstáculos

Para la realización de esta parte se usa la misma interfaz a través del teclado (script de Matlab *ControlManualRobot.m*) que en la anterior parte para que éste recorra el circuito evitando obstáculos y en el que se registrará la velocidad angular, lineal, posición, orientación y lecturas de los sensores. Hay que recordar comprobar que las direcciones IP de la máquina virtual (*ROS_MASTER_IP*) y la IP de la máquina (*ROS_IP*) son las correctas (Figura 22).

Una vez iniciado el simulador STDR, se ejecuta el script *ControlManualRobot.m* desde la consola de Matlab y, de la misma manera, se obtienen los datos de los sensores con la variable *training*.

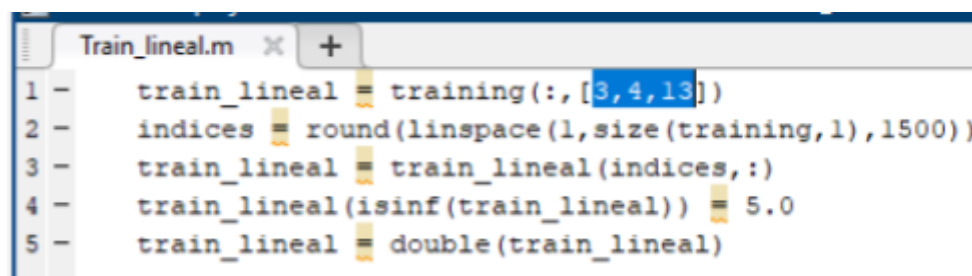
Necesitamos obtener solamente la información que vamos a usar después para diseñar cada controlador, por lo tanto, usamos los siguientes scripts para obtener las columnas que nos interesan:



```
1 - train_angular = training(:, [3,4,12])
2 - indices = round(linspace(1,size(training,1),1500))
3 - train_angular = train_angular(indices,:)
4 - train_angular(isinf(train_angular)) = 5.0
5 - train_angular = double(train_angular)
```

Figura 36. Cambios en el script para obtener la variable *train_angular*.

En el caso de la velocidad angular, el script *Train_angular.m* (Figura 36), solo nos quedamos con las columnas 3, 4 y 12 de la matriz *training* (sensor 2, sensor 3 y velocidad angular). También reducimos el número de filas a 1500 y sustituimos los valores infinitos por el valor máximo permitido (5.0 m).



```
1 - train_lineal = training(:, [3,4,13])
2 - indices = round(linspace(1,size(training,1),1500))
3 - train_lineal = train_lineal(indices,:)
4 - train_lineal(isinf(train_lineal)) = 5.0
5 - train_lineal = double(train_lineal)
```

Figura 37. Cambios en el script para obtener la variable *train_lineal*.

En el caso de la velocidad lineal, el script *Train_lineal.m* (Figura 37), nos quedamos con las columnas 3, 4 y 13 de la matriz *training* (sensor 2, sensor 3 y velocidad lineal). De la misma forma, reducimos el número de filas a 1500 y sustituimos los valores infinitos por el valor máximo permitido (5.0 m).

Estas variables se guardan en el Workspace de Matlab (Figura 25). Se puede consultar los datos que hemos usado en la carpeta ConObstaculos dentro de Parte 2, la variable *training* en el fichero *training.mat*, la variable *train_angular* en *train_angular.mat* y *train_lineal* en *train_lineal.mat*.

Para el controlador de la velocidad angular se siguen los mismos pasos que en el apartado sin obstáculos:

1. Ejecutar la herramienta Anfisedit.
2. Cargar los datos de entrenamiento/training. Seleccionando las mismas opciones que anteriormente. El nombre de la variable es *train_angular*.
3. Generar el FIS. De la misma forma que para los controladores anteriores.
4. Entrenar el sistema. El número de *Epochs* son 220. Cuando el entrenamiento ha finalizado se obtiene un error de 0.11704.
5. Comprobar el controlador generado. El resultado final del entrenamiento es el de la Figura 38.
6. Guardar el controlador. El nombre del fichero será *ControlNBW.fis*.

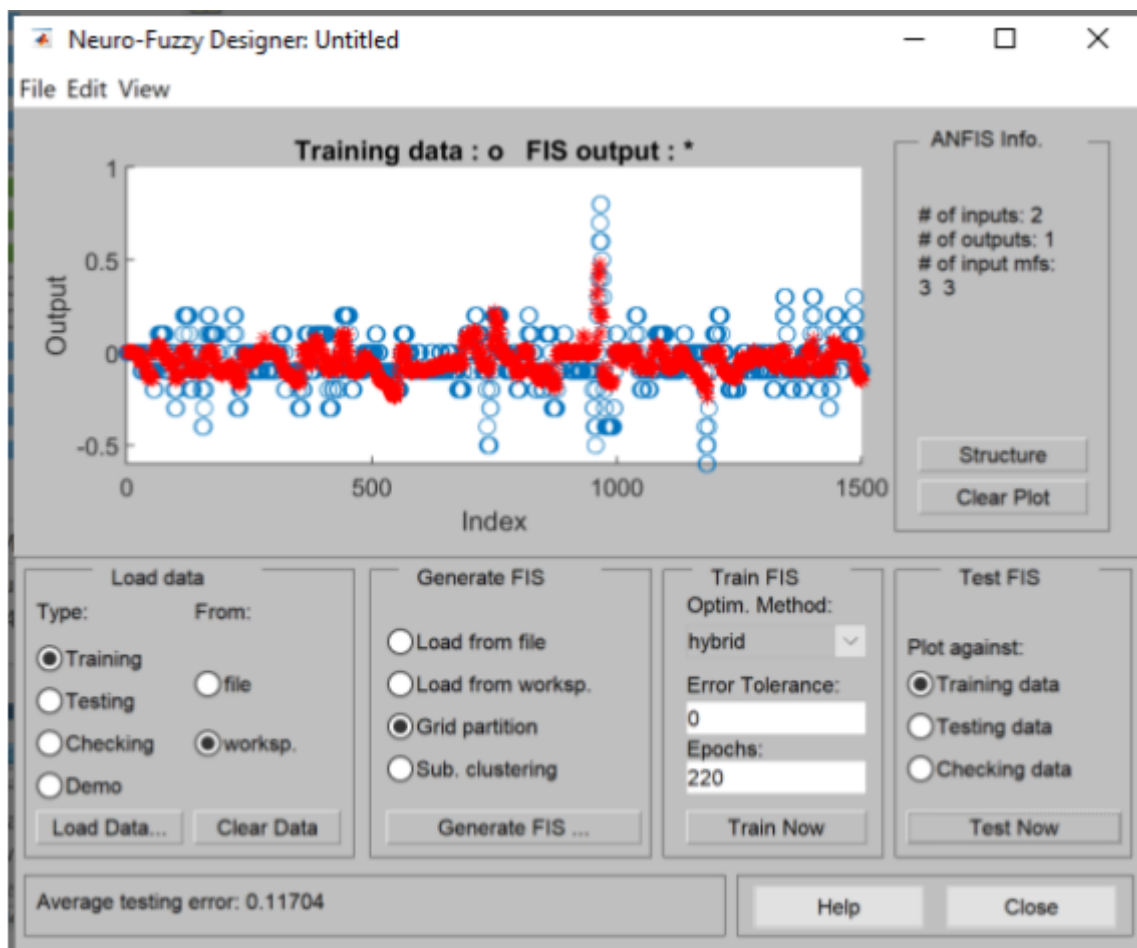


Figura 38. Resultado final del entrenamiento angular.

Para el controlador de la velocidad lineal se realizan los mismos pasos que en los casos anteriores:

1. Ejecutar la herramienta Anfisedit.
2. Cargar los datos de entrenamiento/training. Seleccionando las mismas opciones que anteriormente. El nombre de la variable es *train_lineal*.
3. Generar el FIS. De la misma forma que para los controladores anteriores.
4. Entrenar el sistema. El número de *Epochs* son 50. Cuando el entrenamiento ha finalizado se obtiene un error de 0.067457.
5. Comprobar el controlador generado. El resultado final del entrenamiento es el de la Figura 39.
6. Guardar el controlador. El nombre del fichero será *ControlNBV.fis*.

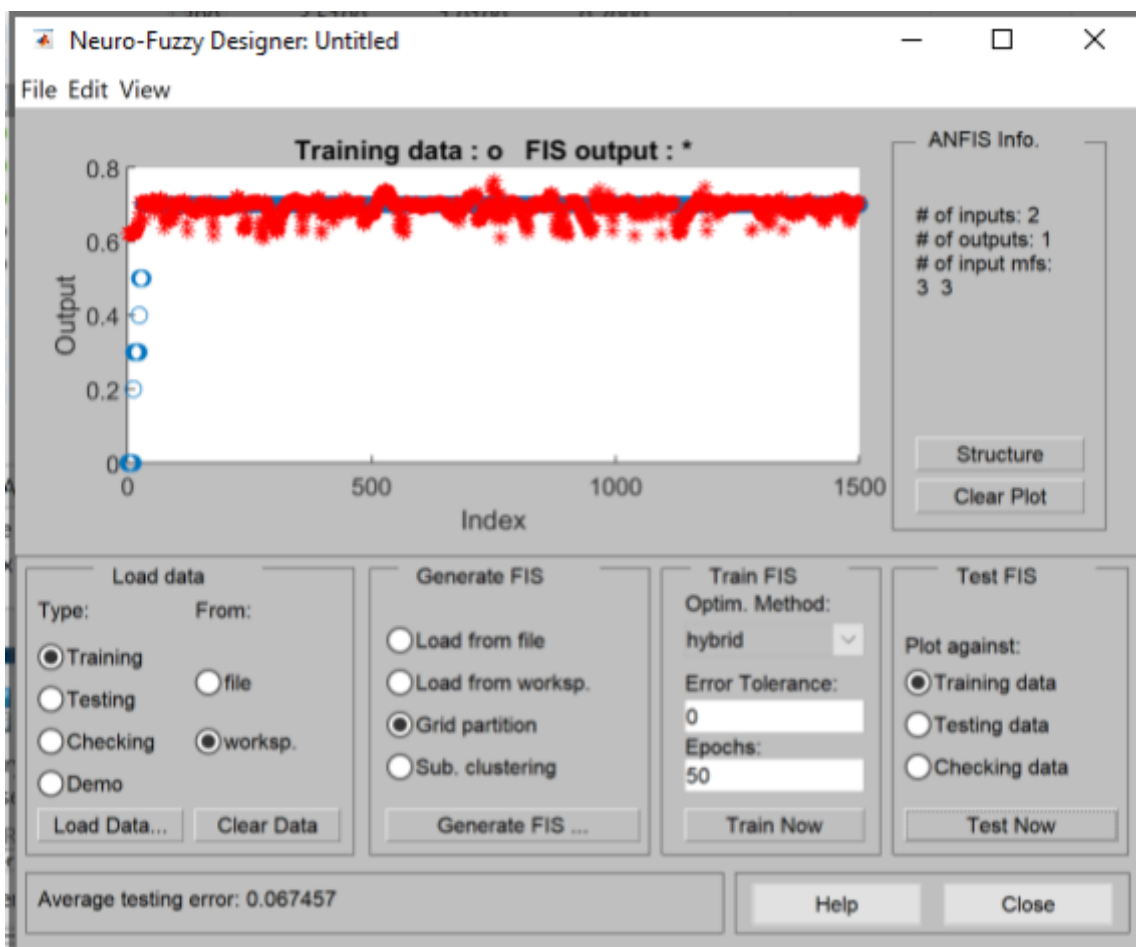


Figura 39. Resultado final del entrenamiento lineal.

Para terminar, modificamos el esquema *test_controler_generic.slx* de la Figura 40. En la entrada del controlador de velocidad lineal (ControladorV) se hace la conexión con la salida de los sensores 2 y 3, al igual que la entrada del controlador de velocidad angular (ControladorW). Los displays se siguen manteniendo. Al igual que anteriormente se añade el nombre del fichero *ControlNBV.fis* dentro del ControladorV en el bloque Fuzzy Logic Controller y el fichero *ControlNBW* dentro del ControladorW en el bloque Fuzzy Logic Controller. Finalmente se comprueba la IP de la máquina virtual (Figura 12).

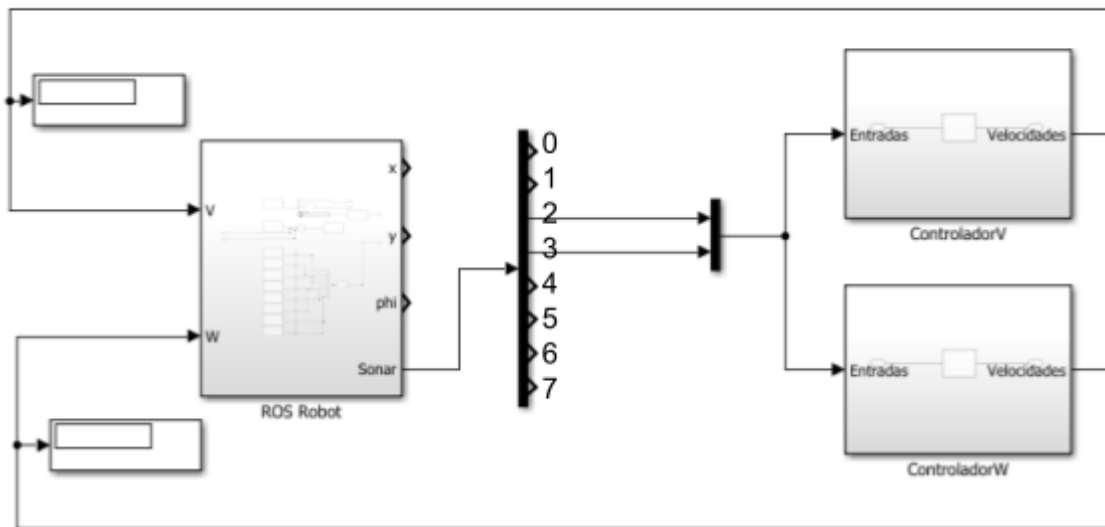


Figura 40. Esquema final con los dos Controladores Neuroborroso.

Una vez diseñados los controladores y configurados los parámetros necesarios para un correcto funcionamiento, mostramos cómo el robot realiza el recorrido en el siguiente video:

- <https://www.youtube.com/watch?v=mBQ8rl9hqfk>

Competición

Para la competición decidimos competir con el controlador borroso que hemos diseñado anteriormente, siendo un único controlador el que controla V y W. En ningún caso se sobrepasan los límites de V y W que figuran en la tabla de características del Amigobot (Figura 41) ya que en la velocidad lineal se mantiene en 1 m/s, la máxima permitida, y la velocidad angular oscila entre un máximo aproximado entre 0.5 y 0.6 rad/s y un mínimo entre -0.5 y -0.6 rad/s.


Longitud	33 cm	
Anchura	28 cm	
Altura	15 cm	
Velocidad lineal máx.	1 m/s	
Velocidad angular máx.	1 rad/s	

Figura 41. Características del Amigobot.

Tampoco se ha fijado ninguna de las velocidades (V y/o W) a un valor constante. Como se puede ver en la gráfica de la función GRANDE de la variable V (Figura 42), usamos una función tipo delta para representar la única función perteneciente a esta variable.

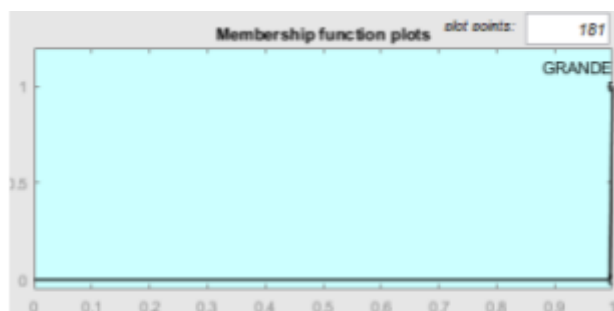


Figura 42. Gráfica de las funciones de la variable V.

Volvemos a adjuntar el mismo video que el que hemos usado para demostrar el correcto funcionamiento del diseño de un control borroso capaz de recorrer el entorno con obstáculos:

- <https://www.youtube.com/watch?v=5lhJNkv2mWg>

En la grabación del recorrido de nuestro robot no se realizan cortes ni edición y se muestra claramente el cronómetro visible en el simulador STDR en la parte inferior izquierda.

Se puede observar que el robot empieza a moverse aproximadamente unos milisegundos después del instante 1.5 sec. Una vuelta completa finaliza cuando el cronómetro de forma estimada marca 1 min y 4 sec, por lo tanto, el robot tarda en realizar el recorrido más o menos 1 min y 2.5 sec. Estas mediciones se han realizado a ojo sobre el cronómetro del simulador STDR, por lo que no son exactas.