PRÁCTICA FINAL

Patricia Cuesta Ruiz

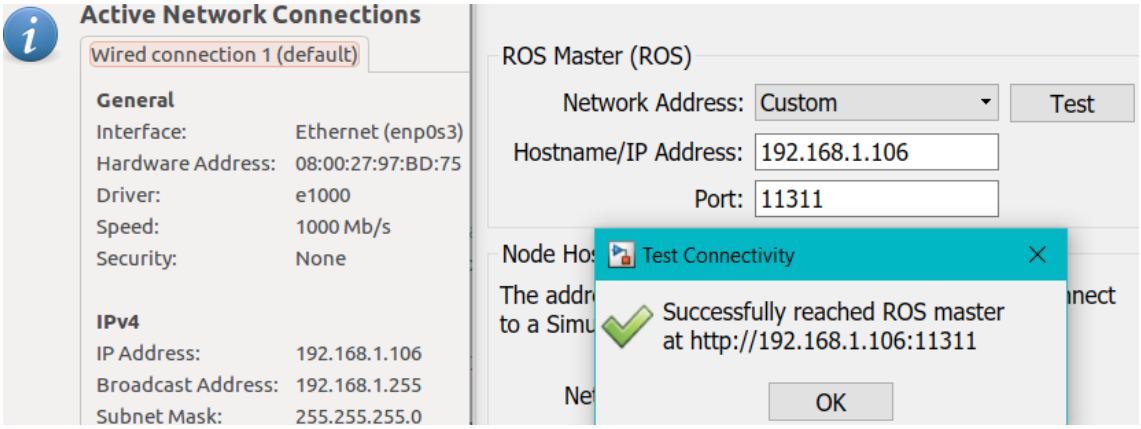
Víctor Gamonal Sánchez

PARTE 1: Diseño manual de un control borroso de tipo Mamdani

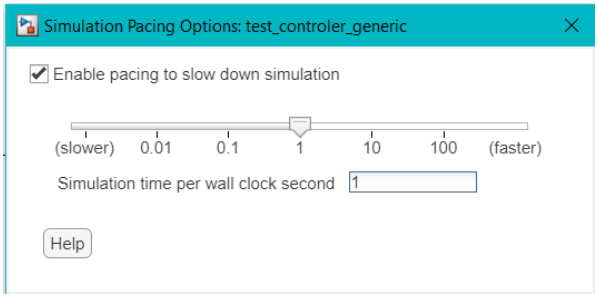
El desarrollo de esta parte de la práctica ha consistido en la creación de un controlador borroso de tipo Mamdani que permita a nuestro robot moverse libremente por los mapas facilitados por los profesores de la asignatura de tal modo que evite los obstáculos que se encuentra en algunos de ellos a lo largo de su recorrido, además de recorrer el mapa sin problema si no hubiera obstáculos.

Es importante que tengamos en cuenta las siguientes consideraciones previas a la creación de este controlador:

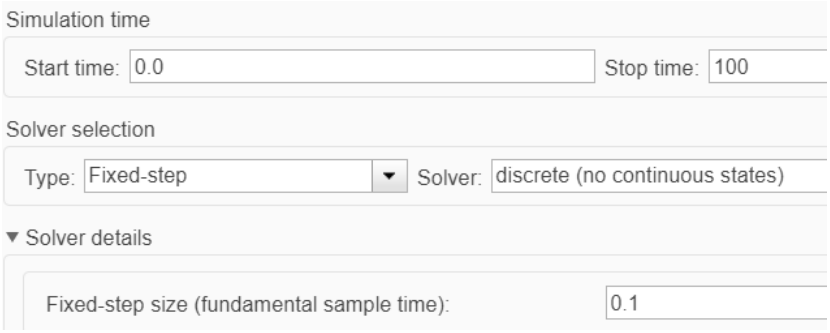
- En primer lugar, hemos configurado el Subscriber de nuestro robot en Simulink (Windows) de tal forma que pueda conectarse a nuestro ROS MASTER localizado en la máquina virtual de Ubuntu para el correcto funcionamiento de la práctica, proporcionando la IP y el puerto de esta.



- Acto seguido, y tal y como especificaba el documento de guía rápida de instalación de ROS facilitado por los profesores, configuramos los siguientes parámetros para la simulación del robot:
  - Habilitamos la opción ***Simulation Pacing***.



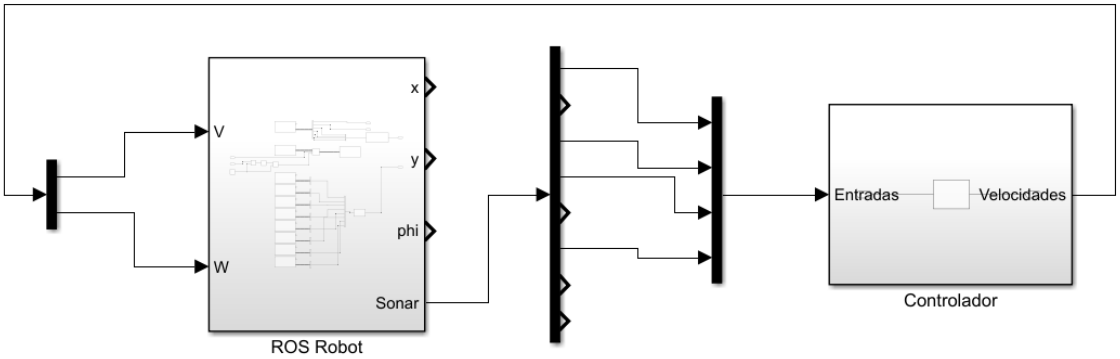
- Configuramos el Time Step de tal forma que le dimos valor 0.1s (es decir, el sondeo de los sensores del robot se hará, al menos, cada 0.1 segundos relativos a la simulación); añadimos también 100 segundos como tiempo máximo de simulación.



Una vez realizadas todas estas configuraciones previas, procedemos a definir el funcionamiento de nuestro robot.

MODELO DE SIMULINK

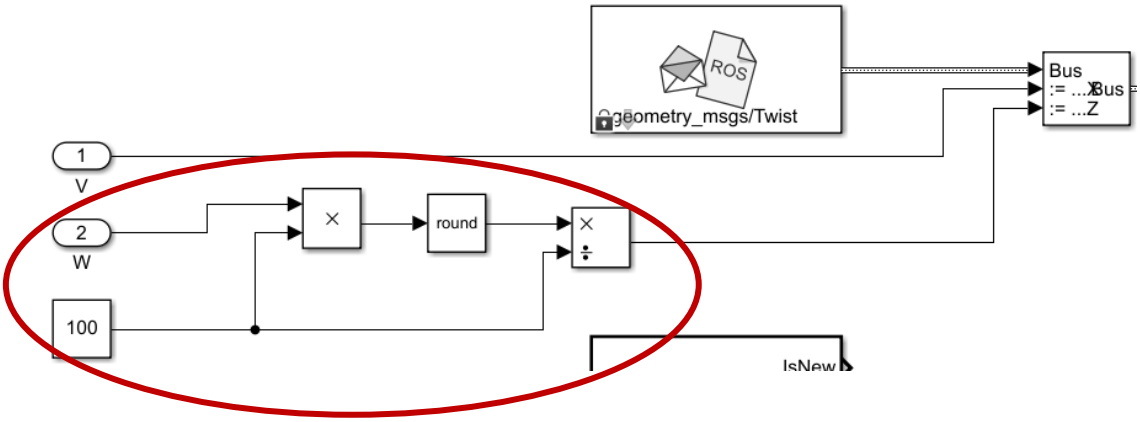
En primer lugar, es importante definir el esquema general del Modelo de Simulink (facilitado por los profesores y bajo el nombre de "***test_controler_generic.slx***") y que se puede apreciar a continuación:



Este esquema cuenta con el bloque del robot, el bloque del controlador borroso para regular su trayectoria y los multiplexores y demultiplexores necesarios para hacer las conexiones.

Como se puede observar en el multiplexor principal, únicamente se han hecho uso de **4 sensores**, siendo estos el 0, 2, 3 y 5, de izquierda a derecha respectivamente. El resto de las salidas del bloque del robot no se utilizan.

Si profundizamos en el **bloque del robot**, queremos hacer hincapié en una pequeña modificación que hemos realizado en su interior.



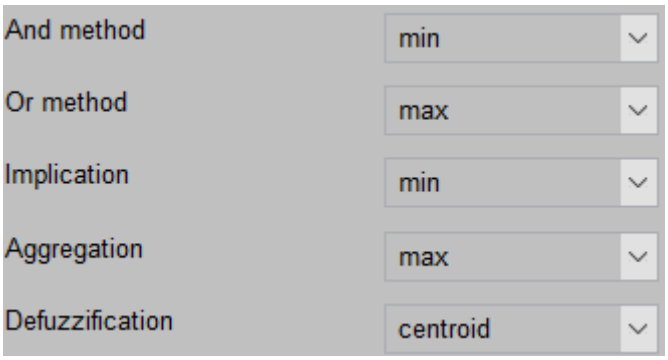
En la entrada de valores V (velocidad lineal) y W (velocidad angular) de nuestro robot, y más concretamente en la de W, nos hemos dado cuenta de que ROS tiene conflicto con el rango de valores (-0.01, 0.01) para la velocidad angular, por lo que hemos decidido que:

- Si su valor se encuentra entre (-0.01, 0] se redondee a -0.01.
- Si su valor se encuentra entre [0, 0.01) se redondee a 0.01.

Esto se hace así para evitar que el robot pueda pararse.

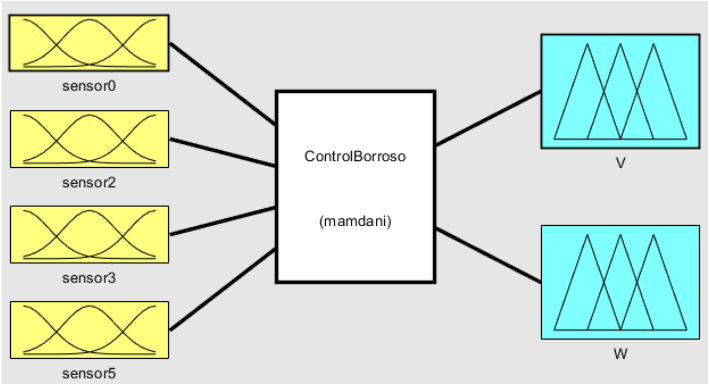
CONTROLADOR BORROSO MAMDANI

En lo referido al controlador, hemos usado la siguiente configuración en lo que a borrosificación, desborrosificación y normas a la hora de hacer inferencia se refiere:

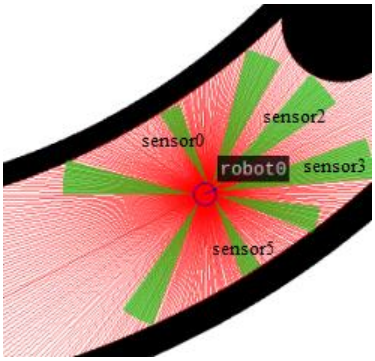


Hemos utilizado esta configuración ya que es la que viene por defecto y obtuvimos muy buenos resultados en prácticas anteriores.

Como comentábamos antes al observar nuestro esquema del modelo de Simulink, nuestro controlador borroso cuenta con un total de 4 entradas, una para cada sensor (que recordamos eran los 0, 2, 3 y 5). El controlador tiene además 2 salidas: una para la velocidad lineal (V) y otra para la velocidad angular (W) que el robot debe tomar para seguir con la trayectoria de forma correcta a lo largo del recorrido del mapa.

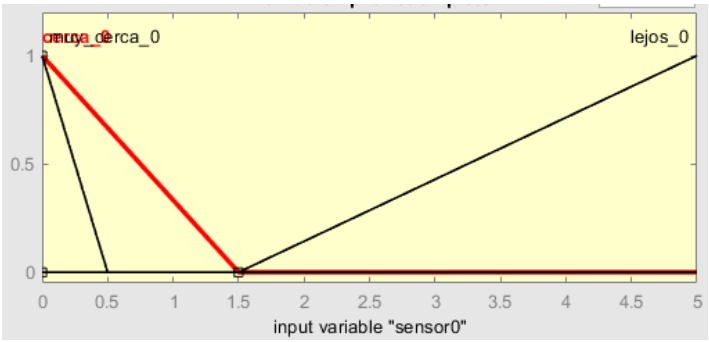


Aquí se pueden apreciar los sensores de nuestro robot que están en uso.



La disposición de los sensores de nuestro robot se podría decir que es simétrica; los sensores 2 y 3 (los interiores) se abren con el mismo ángulo, al igual que pasa para los sensores 0 y 5 (los exteriores) pero siempre con signo contrario entre ellos. Esto ha sido primordial a la hora de **definir las funciones de pertenencia** de cada una de estas entradas, así como a la hora de **definir las reglas**.

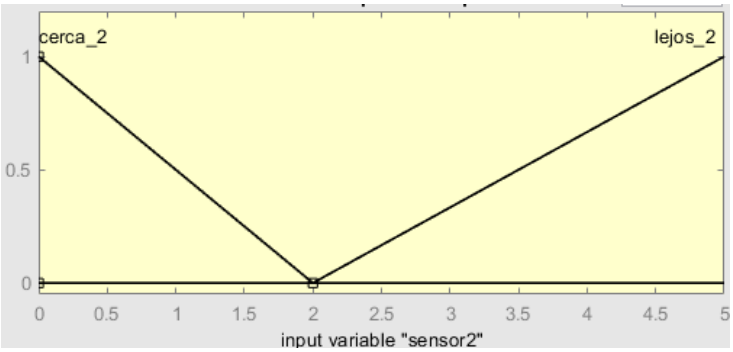
- **Entradas**: el **rango** que hemos escogido para cada una de las entradas (los sensores del robot) es **[0, 5]** ya que esta es la distancia máxima a la que un sensor puede leer información.
    - Sensores 0 y 5 (sensor_0, sensor_5): en el caso de los sensores externos hemos definido las siguientes funciones de pertenencia:



Se ha definido de igual forma para ambos sensores.

Se han definido 3 funciones de pertenencia (***muy_cerca, cerca y lejos***) de tal modo que podamos controlar la distancia al obstáculo de la forma más exacta posible y que el robot pueda reaccionar ante él a tiempo y pueda esquivarlo. Los valores de estas funciones se han declarado así tras numerosas pruebas.

- ***Sensores 2 y 3 (sensor_2, sensor_3)***: en el caso de los sensores internos hemos definido las siguientes funciones de pertenencia:

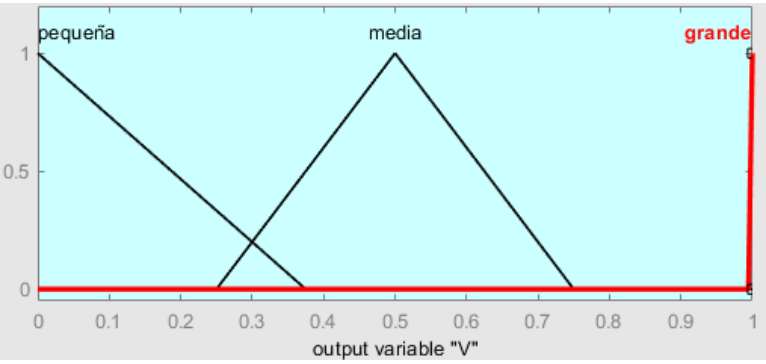


Se ha definido de igual forma para ambos sensores.

Se han definido 2 funciones de pertenencia (cerca y lejos) de tal modo que modo que podamos controlar la distancia al obstáculo pero de una forma no necesariamente tan precisa como ocurría para los sensores externos pero siempre que el robot pueda esquivarlo. Los valores de estas funciones se han declarado así tras numerosas pruebas.

– **Salidas**:

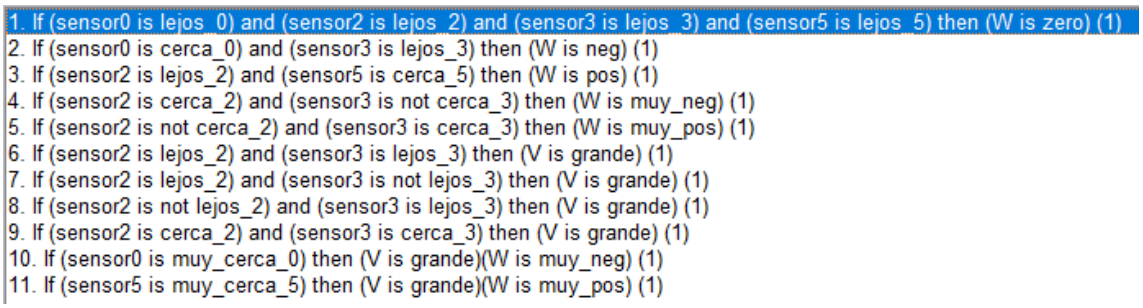
- ***Velocidad lineal (V)***: en el caso de la velocidad lineal hemos establecido que su rango de valores esté comprendido en [0, 1] siendo 1 m/s la velocidad máxima a la que el robot podrá avanzar, tal y como recoge el enunciado de la práctica.



A pesar de que hayamos definido tres funciones de pertenencia, finalmente sólo hemos hecho uso de una (***grande***) y es que en todo momento queremos que nuestro robot siga una velocidad constante y máxima de 1 m/s. Las otras dos (***pequeña*** y ***media***) fueron usadas durante el desarrollo del controlador para establecer que a medida que el robot llegase al obstáculo este redujera la velocidad y pudiera evitarlo más fácilmente; sin embargo, y como acabamos de comentar, finalmente fuimos capaces de optimizarlo para que siempre tuviese una velocidad constante y máxima.

- ***Velocidad angular (W)***: en el caso de la velocidad angular hemos establecido que su rango de valores esté comprendido en [-1, 1] ya que la velocidad angular máxima establecida en el enunciado de la práctica era de 1 rad/s.
  Hemos definido un total de cinco funciones de pertenencia de tal forma que:
  - Si el obstáculo se encuentra muy cerca, entonces el robot deberá realizar un giro más grande de lo normal (***muy_neg***, ***muy_pos***)
  - Si el obstáculo se encuentra cerca, entonces el robot se irá preparando para girar un poco más (***neg***, ***pos***).
  - Si los sensores no detectan cerca ningún obstáculo, entonces el robot avanzará normal (***zero***).

En cuanto a las **reglas de inferencia**, hemos definido un total de **11 reglas**.



A continuación, procedemos a explicar cada una de ellas:

1. Si ningún sensor localiza obstáculos cerca, entonces el robot puede avanzar en línea recta.
2. Si el sensor_0 (externo izquierda) tiene el obstáculo cerca y el sensor_3 (interno derecho) lo tiene lejos, entonces no hay peligro de choque inmediato, por lo que el robot realizará un giro constante con W negativo.
3. A contraparte de la regla 2, se realiza exactamente lo mismo pero en sentido contrario (utilizando sensor_2 como interno y sensor_5 como externo).
4. En caso de que el sensor_2 (interno) detecte un choque inmediato (por obstáculo o por pared), entonces ahora el giro tendrá más peso y será más pronunciado para evitar el choque.
5. A contraparte de la regla 4, se realiza exactamente lo mismo pero en sentido contrario (utilizando sensor_3 sensor que detecta este obstáculo o pared).
6. Si no hay ningún obstáculo cerca de los sensores internos, entonces la velocidad lineal será la máxima posible.
7. Aunque un sensor interno detecte cerca un obstáculo, la velocidad seguirá siendo la máxima posible.
8. Aunque el otro sensor interno al nombrado en la regla anterior detecte cerca un obstáculo, la velocidad seguirá siendo la máxima posible.
9. Aunque ambos sensores internos detecten cerca un obstáculo, la velocidad seguirá siendo la máxima posible.
10. Si la distancia al obstáculo/pared es muy pequeña para un sensor externo (en este caso para sensor_0) y el choque es inminente, entonces el robot corrige su trayectoria con un giro muy pronunciado con W negativo.
11. A contraparte de la regla 10, se realiza exactamente lo mismo pero en sentido contrario (es decir, W positivo) y cambiando el sensor por el sensor_5.

Como hemos podido observar en las reglas 6, 7, 8 y 9, independientemente de la situación en la que se encuentre el robot, la velocidad siempre será la máxima posible (1 m/s).

Con la configuración de estas reglas hemos obtenido que el robot siga una trayectoria bastante buena en la cual intenta avanzar por el centro del carril, apartándose de este únicamente cuando encuentra obstáculos próximos a él.

VÍDEOS DEL RECORRIDO DEL ROBOT

- Recorrido sin obstáculos (mapa: EntornoSinObstaculos.png)

https://youtu.be/u_QHqMMCLD8

- Recorrido con obstáculos (mapa: EntornoConObstaculos2.png)

https://youtu.be/KDVY0WcQZus

- **Recorrido del mapa final** (mapa: EntornoConObstaculos_ex.png)

https://youtu.be/PeCPDrb9fwQ

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

Para la realización de esta segunda parte de la práctica debemos generar datos para poder entrenar la red neuronal. Estos datos los recogemos del script de MatLab proporcionado por los profesores (***"ControlManualRobot.m"***); los datos captados por los sensores, posiciones, velocidad angular y lineal se almacenan cada 0.1 segundos como así lo establece el código.

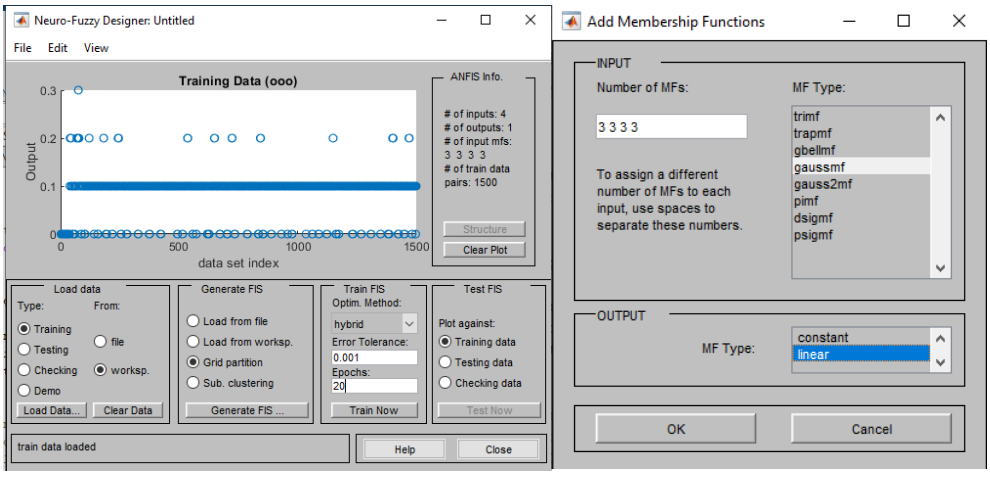
También es importante destacar que hemos realizado a mano los circuitos tanto con obstáculos como sin ellos con aproximadamente 4 vueltas cada uno, lo suficiente para generar más de 1500 filas en la matriz training, y dicha matriz de datos se ha guardado en archivos distintos para cada circuito. Una vez recogidos todos estos datos mencionados, hemos hecho uso de la herramienta **anfisedit** para generar y entrenar un controlador borroso únicamente para la velocidad angular, ya que para la **velocidad lineal hemos cogido siempre el valor de 1 m/s** (es decir, el máximo permitido) como se podrá ver más adelante en este documento.

ENTRENAMIENTO SIN OBSTÁCULOS

Para el circuito sin obstáculos hemos generado el siguiente script para cargar los datos de entrenamiento obtenidos; hemos escogido, para el controlador de la velocidad angular, los sensores **0, 1, 4 y 5** ya que son los que consideramos óptimos a ser los laterales y frontales exteriores.

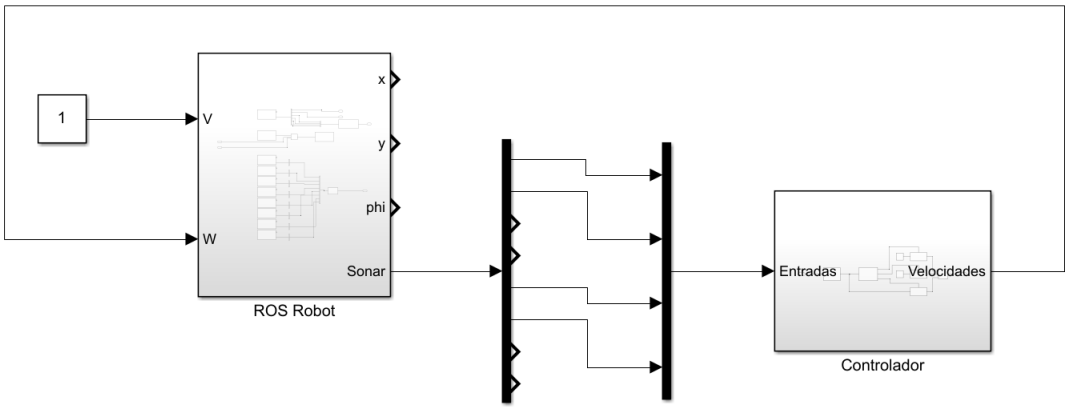
```
clear all;
close all;

% Entrenamiento angular
train_angular = training(:,[1,2,5,6,12])
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)
```

Tras la carga de datos y como mencionábamos antes, hemos utilizado la herramienta anfisedit desde la consola de comandos para cargar los datos de entrenamiento angular desde el workspace.

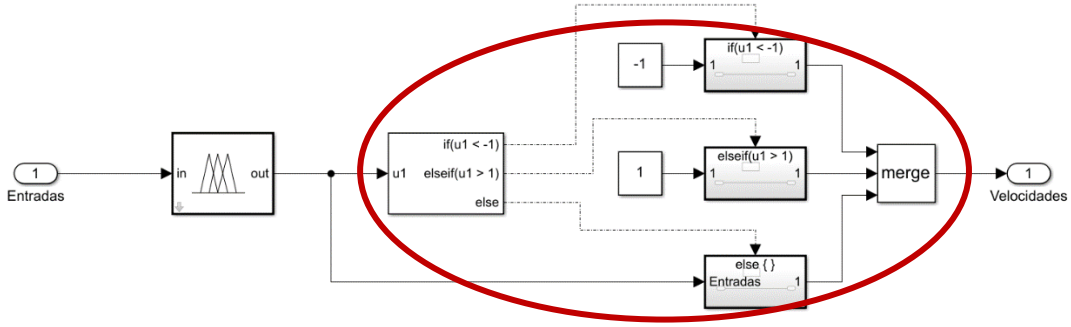


La configuración que hemos establecido se debe a numerosas pruebas que hemos realizado (ver qué ajustes generaban la menor tasa de error, el número de funciones de pertenencia que generamos, etc.) y finalmente hemos optado por realizar **20 epochs** para no sobreentrenar demasiado a la red y una tolerancia de error de 0.001, utilizando **gaussmf** como tipo de función de pertenencia para las entradas y **linear** para las salidas.

Una vez obtenidos los controladores, los exportamos a archivos .fis para poder cargarlos en Simulink, obteniendo como resultado el siguiente diagrama:



Donde se puede observar que ha sido diseñado de tal forma que la velocidad lineal será siempre la máxima (1) y la única que irá variando será la angular para realizar los giros necesarios para evitar chocarse. Además, dentro del bloque controlador nos encontramos lo siguiente:



Donde hemos añadido lo rodeado con un círculo para poder acotar el valor de la velocidad angular entre el rango de valores [-1, 1].

VÍDEO DEL RECORRIDO DEL ROBOT SIN OBSTÁCULOS

- Recorrido sin obstáculos (mapa: EntornoSinObstaculos.png)

https://youtu.be/RGMuLdl3E0I

ENTRENAMIENTO CON OBSTÁCULOS

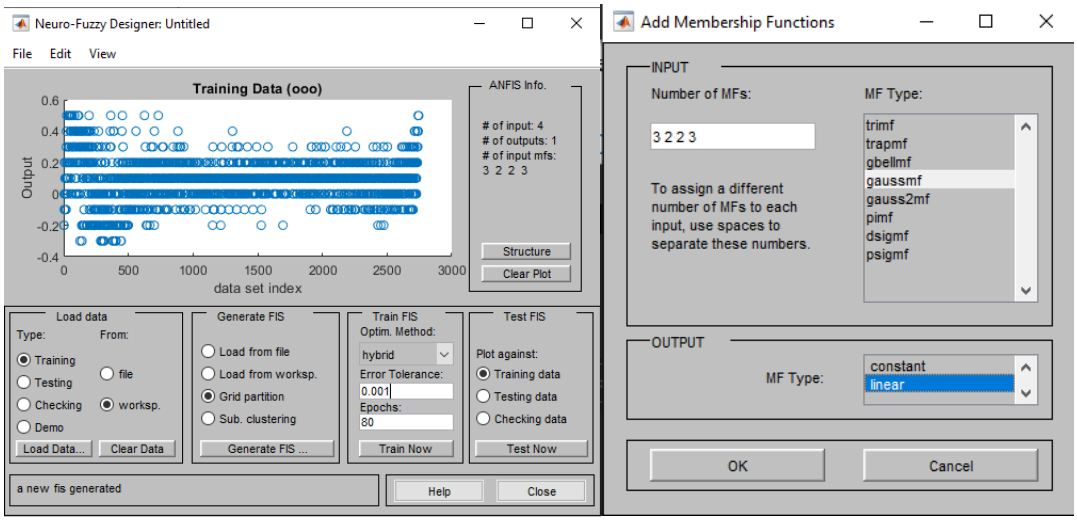
Para el caso del recorrido que realiza el robot en el circuito con obstáculos, hemos tenido en cuenta numerosas consideraciones.

En primer lugar, hemos escogido cuatro sensores para el **entrenamiento de la velocidad angular** pero a diferencia del caso anterior, estos son ahora los **sensores 0** y **5** (como sensores exteriores) y **2** y **3** (como sensores interiores); esto es así debido a los buenos resultados que obtuvimos con la misma elección de estos sensores a la hora de trabajar con el controlador borroso de tipo Mamdani en la parte 1 de esta práctica para la detección de los obstáculos. Para el caso de la **velocidad lineal**, hemos vuelto a escoger un valor constante de 1 m/s.

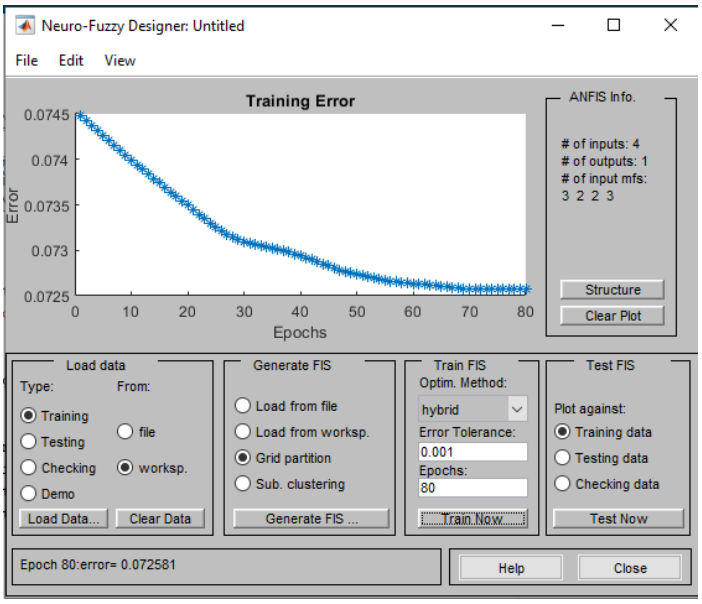
```
clear all;
close all;

% Entrenamiento con obstáculos
load datos_entrenamiento_con_obs training

% Entrenamiento angular
train_angular = training(:,[1,3,4,6,12]);
indices_angular = round(linspace(1,size(training,1),3000))
train_angular = train_angular(indices_angular,:)
train_angular(isinf(train_angular)) = 5.0
train_angular = double(train_angular)
train_angular = unique(train_angular(:,:), 'rows')
```

Otra consideración importante a tener en cuenta es que, como se puede ver en el código del script, esta vez hemos optado por limitar el total de filas de la matriz **training** a 3000 ya que nos hemos dado cuenta de que para **train_angular** obteníamos muchas filas idénticas (**las cuales son totalmente innecesarias ya que no aportan nada al aprendizaje de la red**) y hemos hecho uso de la función ***unique*** de Matlab para eliminar todas ellas; obtenemos, por tanto en nuestro caso, un total de 2700 filas aproximadamente.

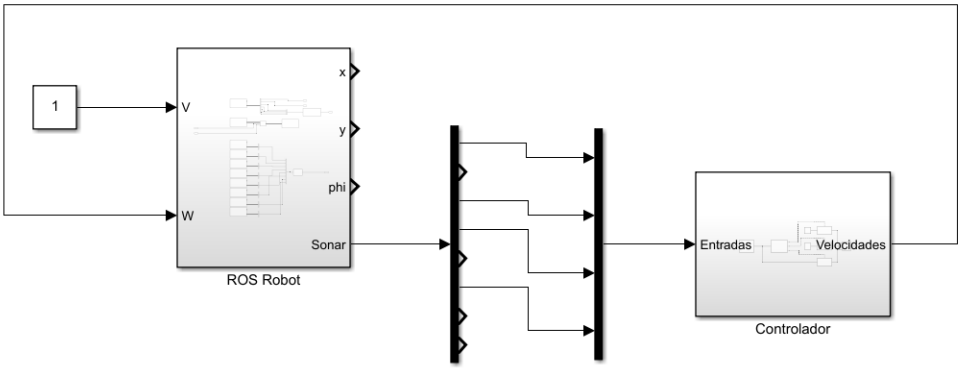
Una vez obtenido el entrenamiento angular, se tiene la siguiente configuración y funciones de pertenencia de tipo **gaussmf** y **linear**.



Se puede observar además, en la figura siguiente, cómo la red neuronal entrena y reduce el error conforme el tiempo avanza, ajustándose a la salida de los datos de entrenamiento.



En cuanto al modelo en Simulink, obtenemos el siguiente esquema en el que con respecto a la parte anterior únicamente hemos modificado las entradas al multiplexor, sustituyendo las de 1 y 4 (previas) por 2 y 3 (actuales).



VÍDEO DEL RECORRIDO DEL ROBOT SIN OBSTÁCULOS

- **Recorrido del mapa final** (mapa: EntornoConObstaculos_ex.png)

https://youtu.be/6X8DvjRjTAc