MINIPROYECTO: DISEÑO DE CONTROLADOERS BORROSOS Y NEUROBORROSOS PARA UN ROBOT MÓVIL

Pablo Acereda García
Department of Computer Science
University of Alcala
28805 - Alcalá de Henares, Madrid, Spain
pablo.acereda@edu.uah.es

Laura Pérez Medeiro
Department of Computer Science
University of Alcala
28805 - Alcalá de Henares, Madrid, Spain
l.perezm@edu.uah.es

September 12, 2020

1 Introducción

El presente documento tiene como finalidad el dar una mayor explicación de las soluciones dadas para cada uno de los apartados propuestos en el enunciado de la práctica, detallando el proceso de resolución que se ha seguido y las deducciones que se han hecho para llegar a ese punto.

Va a desarrollarse y diseñarse un controlador de velocidad para un robot, de manera que este sea capaz de evitar obstáculos estáticos en un circuito cerrado. En el enunciado, se marca como objetivo reducir al mínimo el tiempo que toma recorrer dicho circuito [1] de manera completa.

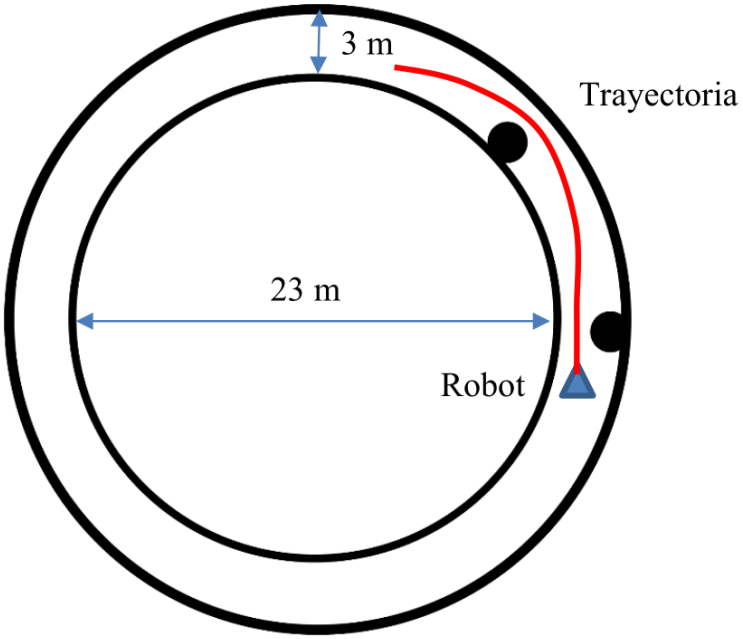


Figure 1: Circuito a recorrer.

Debe tenerse en cuenta que el robot tiene el siguiente tamaño y características [1].

Table 1: Parámetros del robot.

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

Estas medidas afectan a la hora de maniobrar. Puede ser que el robot no esté capacitado para hacer una maniobra y este acabe colisionando contra un objeto o contra la pared del circuito.

Para medir distancias a obstáculos, el robot cuenta con 8 sensores de ultrasonidos y un sonar. Estos pueden ser usados para conocer la cercanía de un objeto respecto al robot y de esta manera evitar así colisiones mediante control automático (control borroso o neuroborroso).

Los sensores quedan distribuidos de la manera especificada en la figura [2]

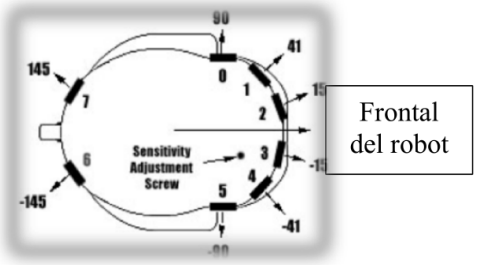


Figure 2: Sensores del robot.

2 Entorno Matlab/ROS

Para poder ejecutar la simulación del circuito es necesario:

- Máquina virtual con:
  - Linux - en este caso se usa Ubuntu.
  - ROS.
- Archivo de configuración y descripción del mapa: `EntornoPracticaFinal.yaml`
- Archivo *launch* que lanza el STDR, el mapa del entorno y el robot: `PracticaFinal.launch`
- Imágenes empleadas como mapa:
  - `EntornoSinObstaculos.png`
  - `EntornoConObstaculos.png`
  - `EntornoConObstaculos2.png`
- Archivo de simulación del robot: `amigobot.xml`

Si bien todos estos archivos son necesarios, esto no implica que la simulación se lance de manera automática. Para que esta funcione, hay que ejecutar el siguiente comando en la terminal:

```
1 roslaunch stdr_launchers PracticaFinal.launch
```

Siguiendo estos pasos como documentación, tal y como se esperaba, debería conseguirse el siguiente entorno de simulación (debe tenerse en cuenta que puede ser necesario modificar el contenido de `EntornoPracticaFinal.yaml` para usar el el mapa que se muestra en la figura 3):

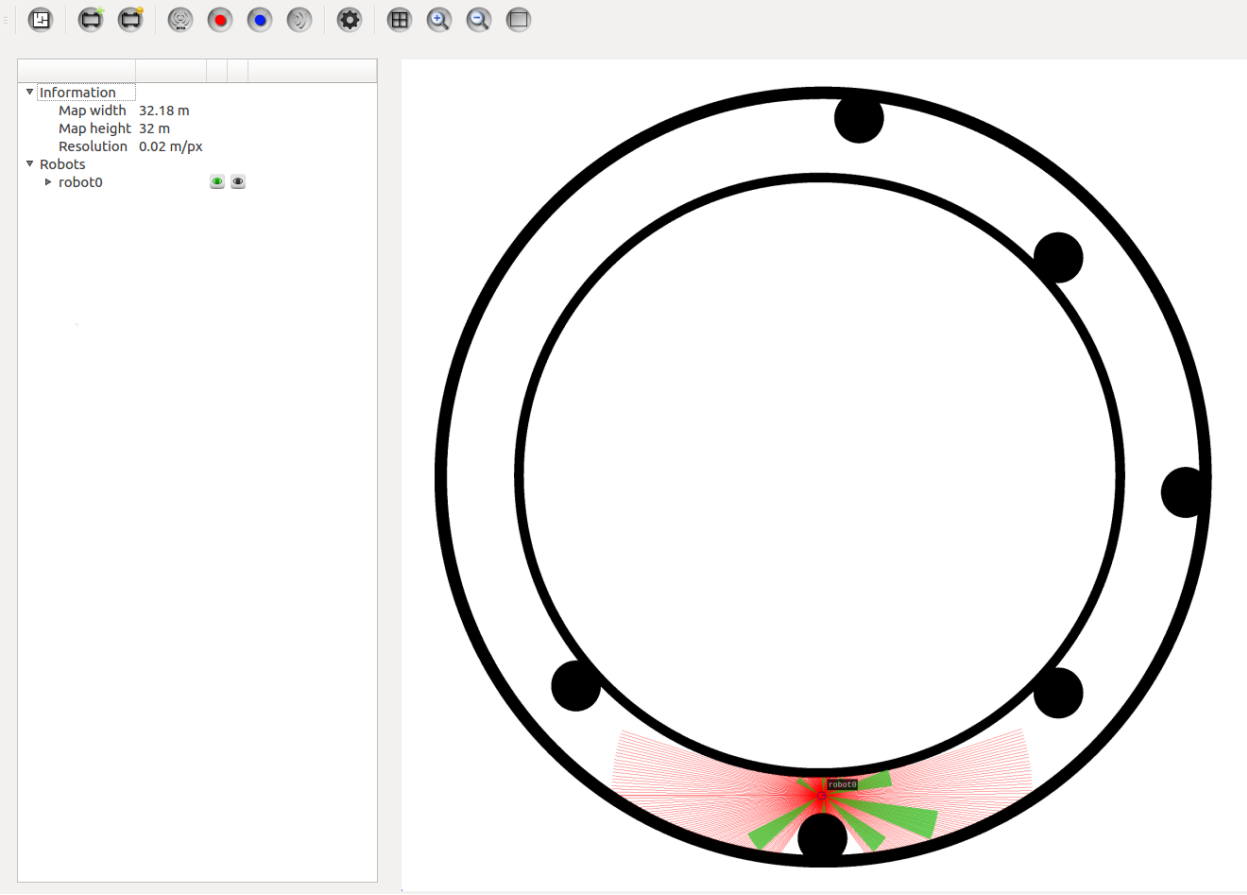


Figure 3: Circuito de simulación del robot.

Hasta aquí solo se incluye una parte de los componentes que realmente son necesarios, debido a que esto ejecuta la simulación, pero no permite que el robot se mueva. Para esto, es necesario incluir en MATLAB la extensión correspondiente. Para ello ir a: `Menu > Apps > Get More Apps` y ahí buscar o seleccionar `Control System Toolbox`, `Fuzzy Logic Toolbox` y `ROS Toolbox` e instalar[1].

Una vez instalados los componentes, ir al archivo `RobotROS.slx` [4]. Dentro de este fichero puede encontrarse el bloque del robot que se emplea para moverse. Esto no indica que sea su *controlador*, sino más bien lo que hace cambiar que este cambie de avance y orientación. Los *controladores* serán los componentes a desarrollar y los que se encargarán de calcular la posición y orientación del robot.

En esta ocasión, es necesario configurar la IP de la máquina virtual dentro del bloque del robot, para que puede haber una correcta comunicación entre ambos componentes. Para ello, debe hacerse doble click en cualquiera de los bloques de *subscripción* a ROS, hacer doble click dicho bloque y finalmente seleccionar *Configure network addesses* [5][2].

[1] Se da por sentado que `Simulink` se encuentra ya instalado. En caso contrario, instalarlo.
[2] La dirección IP puede conseguirse dentro de la configuración de la máquina virtual.

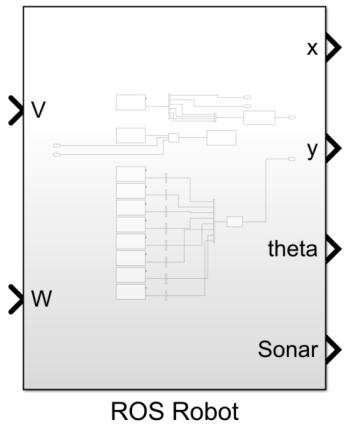


Figure 4: ROS Robot.

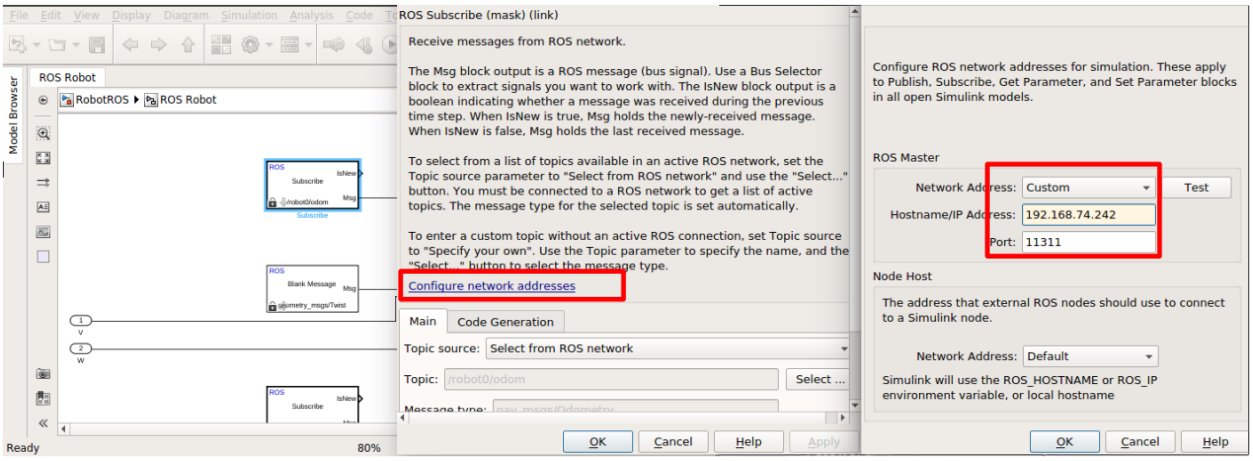


Figure 5: Configurar subscripción entre bloque ROS Robot y el simulador de la máquina virtual

3 Diseño manual de un control borroso de tipo MAMDANI

Dentro de la carpeta 'Part1_Fuzzy' encontramos dos controladores borrosos (`Cont_conObs.fis` y `Cont_sinObs.fis`) con las reglas, funciones de pertenencia y rangos de las mismas necesarios para completar el circuito sin que el robot colisione contra ningún obstáculo o contra las paredes del mismo; así como el archivo de Simulink `test_controller_generic.slx` donde se encuentra el robot conectado al controlador que utilizará[3]. De este esquema se ha añadido el controlador de lógica difusa a emplear y las conexiones entre demultiplexor y multiplexor para el uso de los sónares.

El controlador con 3 sensores queda de la siguiente manera [6]:

Circuito sin obstáculos

Tal y como se intuye, el circuito de este apartado es el mismo de la figura 3, solo que en esta ocasión no tiene círculos negros como obstáculos distribuidos a través del mismo [7].

Para superar este circuito se ha creado un controlador borroso, que usa el fichero `Cont_sinObs.fis`. Debido a la simplicidad del circuito (dos circunferencias concéntricas) y que no hay elementos que causen que el robot tenga que desviarse de la circunferencia que esté trazando, solamente son necesarios dos sensores para asegurarse del funcionamiento correcto del robot:

[3]El esquema del robot ha sido proporcionado por los profesores de la asignatura.

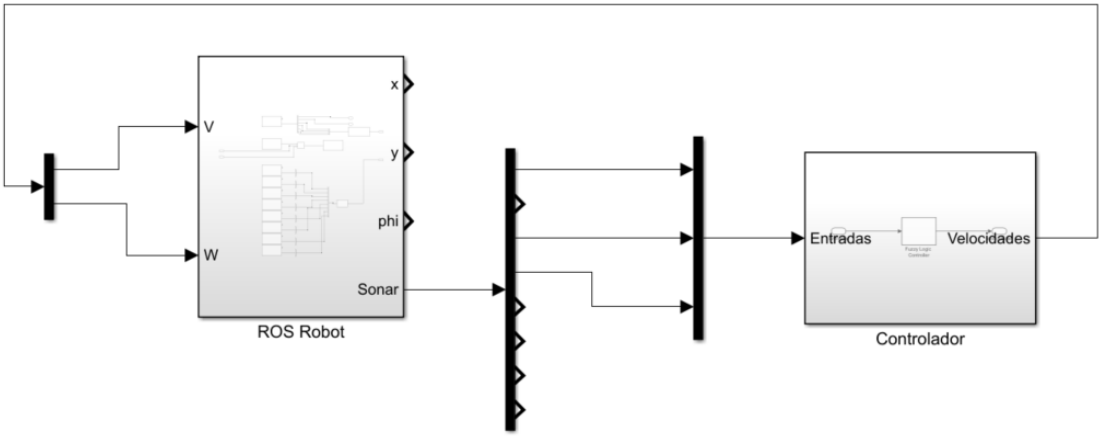


Figure 6: Esquema controlador borroso

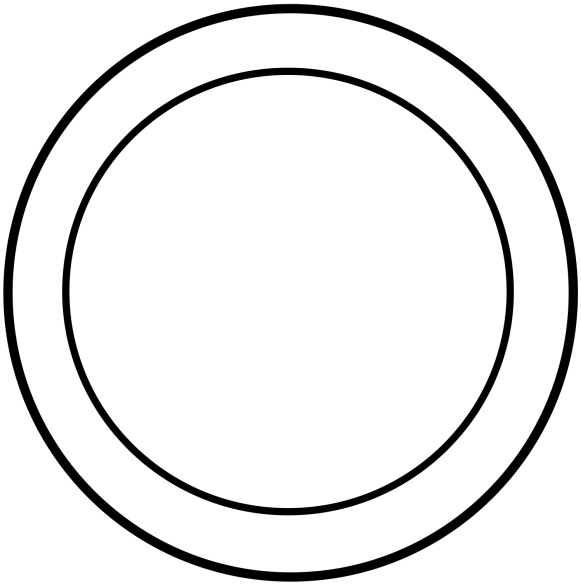


Figure 7: Circuito Sin Obstáculos

- `sonar_0`. Situado en el lateral izquierdo del robot (90º)[4].
- `sonar_2`. Sensor frontal izquierdo (15º).

Respecto a las funciones de pertenencia, ambos sónares cuentan con las tres mismas funciones:

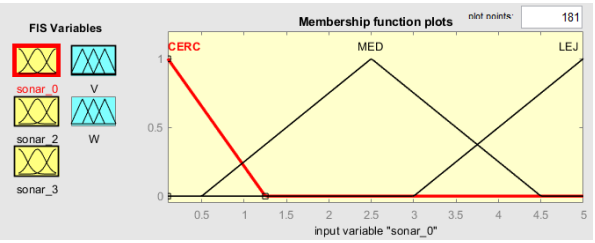
- `CERC`. Distancia cercana.
- `MED`. Distancia media.
- `LEJ`. Distancia lejana.

La manera de elegir los rangos de entrada de los sónares ha sido a partir de la distancia que estos son capaces de interpretar. Esta va desde los 0m hasta los 0.5m (5 como límite superior del rango de pertenencia). Debido a que si la

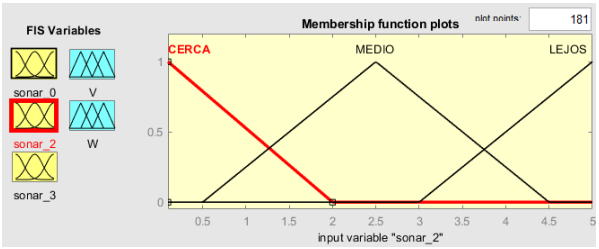
[4]Partiendo de que la parte frontal del robot sean los sensores `sonar_2` y `sonar_3`, tal y como se indica en el enunciado de la práctica.

distancia es menor a 0.01m, puede considerarse que el robot ha colisionado, el límite inferior será este (0.1 transformado en función de pertenencia).

Cada una de las pendientes es personalizada para tratar de obtener el mejor rendimiento a la hora de completar el circuito[5].



(a) Sonar 0

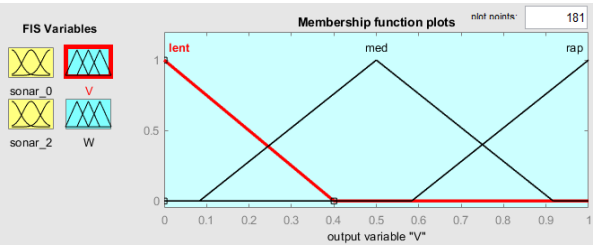


(b) Sonar 2

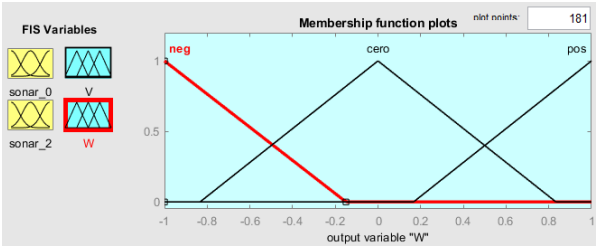
Figure 8: Funciones de pertenencia de entrada sin obstáculos

Por otro lado, están las funciones de salida, que serán la velocidad lineal y la velocidad angular. Estas tienen definidos sus rangos por los límites que presenta el robot de movilidad. Su velocidad puede llegar hasta $1m/s$ y su giro puede ser de hasta $1rad/s$. Estos valores han sido transformados en rangos de las funciones de pertenencia, quedando un rango para la función de velocidad que va desde 0 (parado) hasta 1(máxima velocidad); para la función de velocidad angular se va desde -1 (girar hacia el exterior) hasta 1 (girar hacia el interior).

Las funciones quedan representadas de la siguiente manera:



(a) Sonar 0



(b) Sonar 2

Figure 9: Funciones de pertenencia de salida sin obstáculos

Se emplean las siguientes reglas para hacer funcionar el controlador:

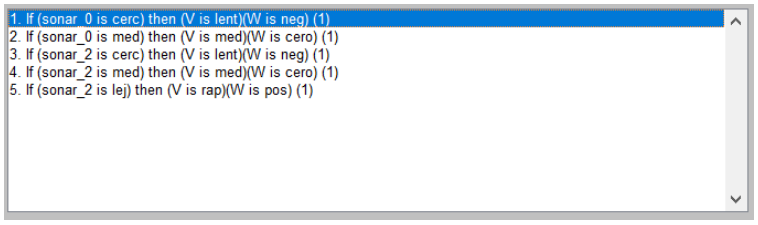


Figure 10: Reglas para controlador sin obstáculos

De las reglas 1 a la 3 se entiende que son necesarias para saber la posición global del robot respecto a la pared. Para nunca alejarse demasiado de esta ni acercase en exceso, debido a que supondría una colisión con la parte exterior/interior de la misma. Lo mismo puede aplicarse a los obstáculos.

De las reglas 4 a 6 se garantiza que el robot esté en todo momento orientado hacia la cara interior del circuito.

[5]Se han empleado las mismas imágenes que las tomadas para el controlador con obstáculos.

Circuito con obstáculos

De la misma manera que en el apartado anterior se tenía un circuito sin obstáculos, en este apartado se cuenta con el mismo circuito con objetos distribuidos a lo largo de la pista por la que el robot debe pasar. Se entiende que colisionar con uno de estos objetos supone que el robot ha chocado y que, por lo tanto, no puede proseguir con el circuito [11].

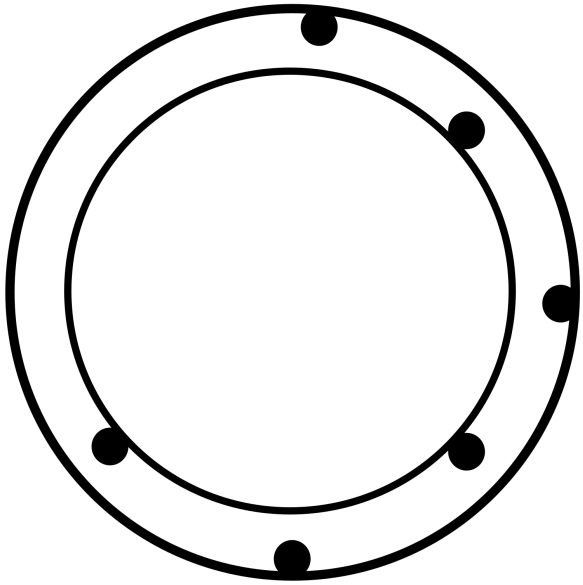


Figure 11: Circuito Con Obstáculos

En esta ocasión, no es suficiente con emplear los dos sensores utilizados hasta ahora, también hace falta añadir un tercer sensor, el `sonar_3`. Está situado inmediatamente después del sensor `sonar_2`. Proporciona lecturas de la parte superior derecha del robot (su ángulo es de -15º).

Se planteó el usar el sensor `sonar_5` (con orientación -90º), pero debido a la forma del circuito se ha visto como innecesario, ya que el resto de sensores son capaces de controlar si el robot se acerca en en demasía al borde exterior o a un objeto situado en la parte derecha del recorrido.

Puede entenderse, por lo tanto, que las reglas de las funciones de pertenencia se ven afectadas de la siguiente manera.

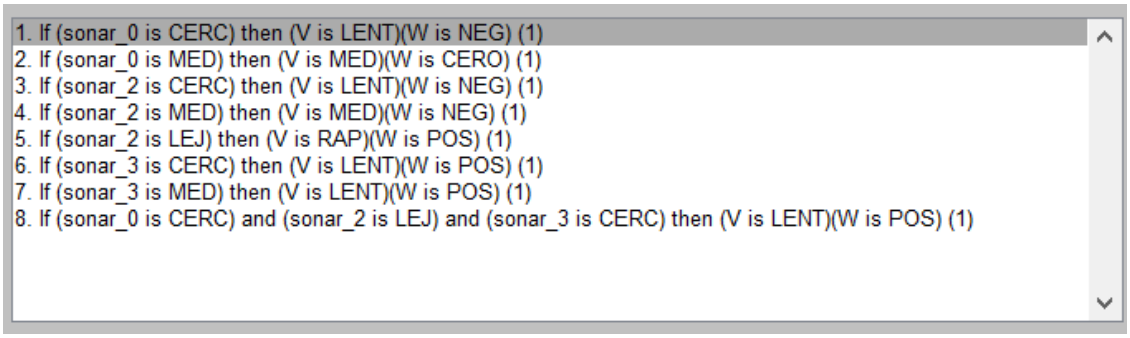


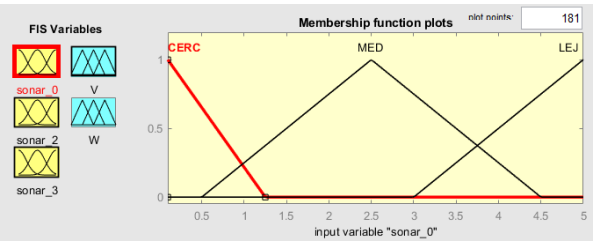
Figure 12: Reglas usadas para el controlador con obstáculos

Estas reglas han sido empleadas para que el robot sea capaz de adaptarse a cualquier situación planteada (dentro de este circuito). Se ha eliminado, una regla. Esta indicaba lo siguiente: `if (sonar_0 is LEJ) then (V is LENT)(W is POS)`. No debería entrar en acción dentro de este circuito, debido a que una distancia lejana (como se verá en la figura 13a) es a partir de 3 metros, que es precisamente la diferencia de distancia entre la circunferencia interna y externa.

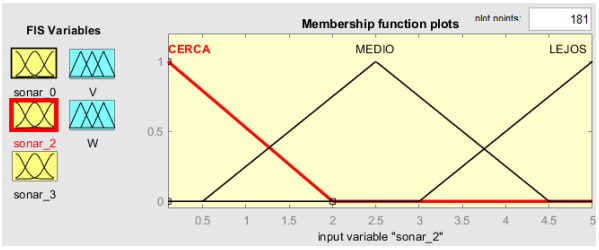
Se emplean las mismas reglas de la 1 a la 6, que en el circuito sin obstáculos. Las reglas 7 y 8 evitan colisionar con objetos situados en la parte derecha del robot. La regla 9 impide colisionar en el momento exacto en el que el robot está

a medio camino de superar un obstáculo. En ese momento, el `sonar_2` deja de detectar un obstáculo, pero el `sonar_3` sigue detectándolo, a la vez que el sonar `sonar_0` detecta una de las paredes[6].

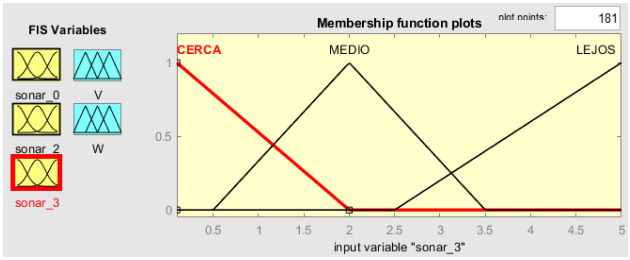
Las funciones de pertenencia quedan de la siguiente manera [13]:



(a) Sonar 0



(b) Sonar 2



(c) Sonar 3

Figure 13: Funciones de pertenencia de entrada con obstáculos

Como puede comprobarse, las funciones para el `sonar_0` y para `sonar_2` son las mismas que en [8].

Las salidas se mantienen de la misma manera que en la figura [9].

Todos los límites y funciones de pertenencia han sido calculadas de manera empírica, tratando de ajustar los límites de manera que se obtenga el mejor resultado posible sin causar la colisión del robot.

4 Diseño automático de un controlador neuroborroso de tipo SUGENIO

Para esta parte del proyecto es necesario controlar el robot de manera manual a través de los circuito sin/con obstáculos. Esos datos se recogen y se emplean para entrenar el controlador del robot. Se utiliza un controlador neuronal que aprende los movimientos y orientación del robot a cada instante para producir un controlador. Este se exporta a un controlador borroso (de ahí que sea neuroborros) que es empleado como controlador del robot.

Antes de entrar en detalle con el proceso seguido para los controladores con y sin obstáculos, hay una serie de pasos que son comunes a ambos.

Es necerio comenzar mediante recolección de datos. Para ello se genera un script que conecta MATLAB con master ROS. Se invoca el código que permite controlar el robot[7] y se recolectan todos los datos posibles a cada instante (movimientos, orientación, velocidad, medidas de sensores, etc.).

```
1 clear all;
2 close all;
3
4 rosshutdown
5
6 global vel_angular;
7 global vel_lineal;
8 global incAngular;
```

[6] Puede que se de la situación inversa entre `sonar_0` y `sonar_3`.

[7] Este script ha sido proporcionado por los profesores de la asignatura. No se ve necesario comentarlo o mostrar el código, debido a que este no ha sido modificado.

```
global incLineal;
global vel_angular_max;
global vel_lineal_max;
global stop

ROS_MASTER_IP = '192.168.1.70'
ROS_IP        = '192.168.1.66'

rosinit(['http://',ROS_MASTER_IP,':11311'],'NodeHost',ROS_IP)

%DECLARACION DE PUBLISHERS
%Velocidad
pub_vel=rospublisher('/robot0/cmd_vel','geometry_msgs/Twist');
%%DECLARACION DE SUBSCRIBERS
odom = rossubscriber('/robot0/odom'); % Subscripcion a la odometria
% Subscripcion a los sensores
sonar_0 = rossubscriber('/robot0/sonar_0', rostype.sensor_msgs_Range);
sonar_1 = rossubscriber('/robot0/sonar_1', rostype.sensor_msgs_Range);
sonar_2 = rossubscriber('/robot0/sonar_2', rostype.sensor_msgs_Range);
sonar_3 = rossubscriber('/robot0/sonar_3', rostype.sensor_msgs_Range);
sonar_4 = rossubscriber('/robot0/sonar_4', rostype.sensor_msgs_Range);
sonar_5 = rossubscriber('/robot0/sonar_5', rostype.sensor_msgs_Range);
sonar_6 = rossubscriber('/robot0/sonar_6', rostype.sensor_msgs_Range);
sonar_7 = rossubscriber('/robot0/sonar_7', rostype.sensor_msgs_Range);
%GENERACION DE MENSAJES
msg_vel=rosmessage(pub_vel);

vel_angular=0;
vel_lineal = 0;
incAngular = 0.1;
incLineal = 0.1;
vel_angular_max = 1;
vel_lineal_max = 1;
fig = figure('KeyPressFcn',@Key_Down);
  t = text(0.1,0.9,'Manten el foco en esta ventana','HorizontalAlignment','left');
  t.FontSize = 16;

  t = text(0.1,0.8,'- Pulsa Flechas ARRIBA/ABAJO para modificar V en 0.1 m/s','
    HorizontalAlignment','left');
  t.FontSize = 13;
  t = text(0.1,0.7,'- Pulsa Flechas IZQUIERDA/DERECHA para modificar W en 0.1 rad/
    s','HorizontalAlignment','left');
  t.FontSize = 13;
  t = text(0.1,0.6,'- Pulsa ESPACIO para finalizar','HorizontalAlignment','left');
  t.FontSize = 13;

  t = text(0.1,0.4,sprintf('Vel lineal actual %g m/s | Vel angular actual %g rad/s
    ', vel_lineal, vel_angular),'HorizontalAlignment','left');
  t.FontSize = 14;

  training=[];

stop = 0;
while (stop == 0)
    pos = odom.LatestMessage.Pose.Pose.Position;
    x   = pos.X;
    y   = pos.Y;

    ori   = odom.LatestMessage.Pose.Pose.Orientation;
    theta = quat2eul([ori.W ori.X ori.Y ori.Z]);
    theta = theta(1);

    s0 = sonar_0.LatestMessage.Range_;
    s1 = sonar_1.LatestMessage.Range_;
```

```
    s2 = sonar_2.LatestMessage.Range_;
    s3 = sonar_3.LatestMessage.Range_;
    s4 = sonar_4.LatestMessage.Range_;
    s5 = sonar_5.LatestMessage.Range_;
    s6 = sonar_6.LatestMessage.Range_;
    s7 = sonar_7.LatestMessage.Range_;

   training = [
       training;
       [s0, s1, s2, s3, s4, s5, s6, s7, x, y, theta, vel_angular, vel_lineal]
   ];
   msg_vel.Linear.X  = vel_lineal;
   msg_vel.Angular.Z = vel_angular;
   send(pub_vel, msg_vel);
   pause(0.1);
end
vel_lineal = 0;
msg_vel.Linear.X  = vel_lineal;
msg_vel.Angular.Z = vel_angular;
send(pub_vel, msg_vel);
save datos_entrenamiento_con_obstaculos training
```

Inicialmente, el script[8] también parseaba los datos para que pudiesen ser empleados directamente, pero esto suponía que si se deseaba repetir el proceso se debía repetir el circuito antes. Para evitar esto, los datos son guardados en un archivo y es otro script el que realiza el parseo de los mismos.

```
%data = matfile('datos_entrenamiento_sin_obstaculos.mat');
data = matfile('datos_entrenamiento_con_obstaculos.mat');
data = data.training;

% Selection of the needed resources for the training angular speed.
% Sonar_0, sonar_2, sonar_3 and angular speed.
%train_angular = data(:, [1, 3, 12]);     % No obstacles
train_angular = data(:, [1, 3, 4, 12]); % Obstacles
indices       = round(linspace(1, size(data, 1), 1500));
train_angular = train_angular(indices,:);
train_angular(isinf(train_angular)) = 5.0;
train_angular = double(train_angular);

% Selection of the needed resources for the training linear speed.
% Sonar_0, sonar_2, sonar_3 and linear speed.
%train_linear = data(:, [1, 3, 13]);     % No obstacles
train_linear = data(:, [1, 3, 4, 13]);  % Obstacles
indice       = round(linspace(1, size(data, 1), 1500));
train_linear = train_linear(indices, :);
train_linear(isinf(train_linear)) = 5.0;
train_linear = double(train_linear);
```

Ahora que los datos han sido recogidos, es momento de ejecutar el comando `anfisedit` (figura [14]) para crear el controlador neuronal. Se crean dos controladores separados, uno para la velocidad lineal y otro para la velocidad angular.

Para reducir el tiempo que tarda en generarse el controlador, se recomienda reducir el número de final que van a usarse para entrenar el sistema.

A partir de aquí deben seguirse los siguientes pasos:

[8]También proporcionado pro los profesores de la asignatura.

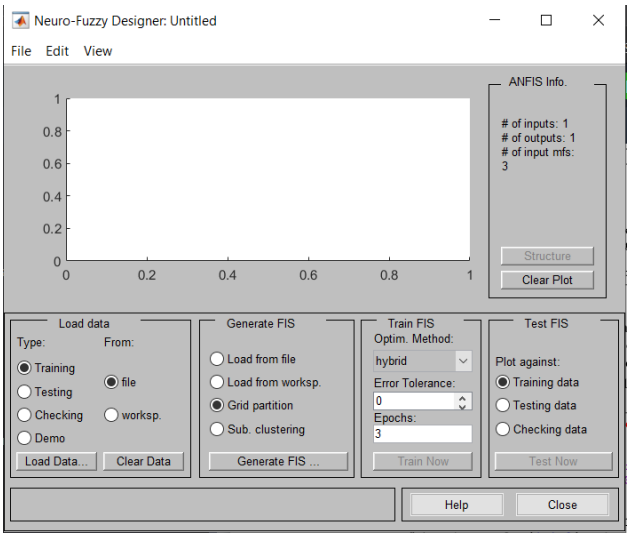


Figure 14: Herramienta Anfisedit Sin Datos

1. Cargar los datos de entrenamiento generados previamente con el script de control manual. Para ello se usa el script mencionado anteriormente `ParseTrainingData.m` desde donde cargamos la información del fichero `datos_entrenamiento.mat` al *workspace* de MATLAB. Recordar que este fichero contiene los datos de entrenamiento obtenidos a partir de `ControlManualRobot.m`. Para cargar los datos es tan sencillo como pulsar en el botón *Load Data*, en la sección de *Load Data* de *Anfisedit* (figura [15]) e introducir el nombre de la variable que se desea cargar.

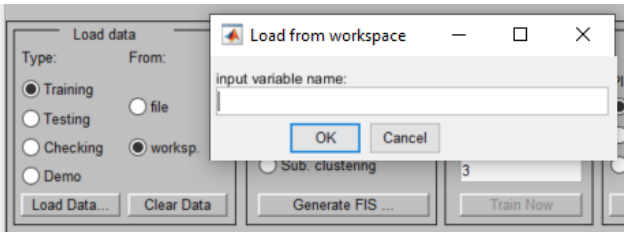


Figure 15: Cargar datos de entrenamiento

2. Esto debería representar los siguientes datos en la gráfica de *Training Data* (figure [16]):

3. Para la generación del FIS seleccionamos el modo *Grid partition* [9]. Se decide usar este método porque se observa que con él se obtienen mejores resultados aunque necesite de mayor tiempo de entrenamiento para finalmente generar el fichero FIS. Finalmente, pulsor sobre *Generate FIS...*.

   (a) Debería aparecer una ventana que permita configurar el númerp y tipo de funciones miembro de entrada y tipo de funciones miembro de salida (figura 17).

   (b) Se deben usar 3 *Number of MFs* debido a que se emplean 3 sensores, por tanto 3 funciones de entrada.

4. Posteriormente se debe entrenar el sistema, pulsando sobre *Train Now* (figura [18]) en la columna *Train FIS*. Se recomienda elegir entre los métodos de optimización *hybrid* o *backpropa* (*backpropagation*). También se recomienda cambiar el número de *epochs* al más adecuado. En nuestro caso se ha elegido *hybrid* con 100 *epochs*.

[9] Genera funciones de membresía al particionar de manera uniforme los rangos de variables de entrada, creando un sistema difuso Sugeno de salida única y cuya base de reglas difusas contiene una regla para cada combinación de funciones de membresía de entrada.

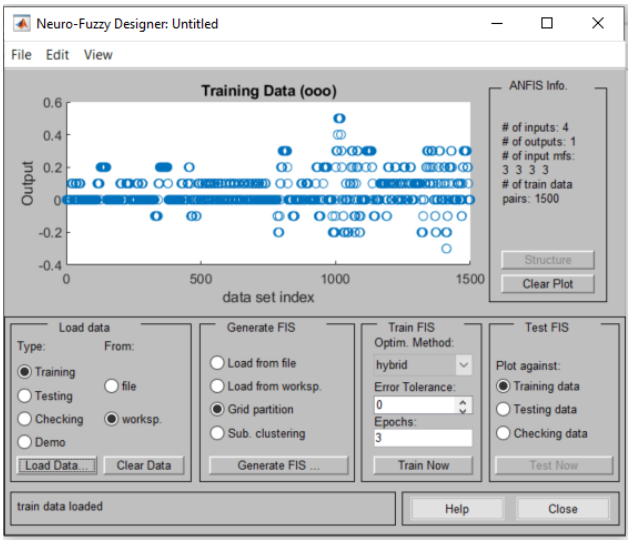


Figure 16: Datos de entrenamiento de velocidad angular

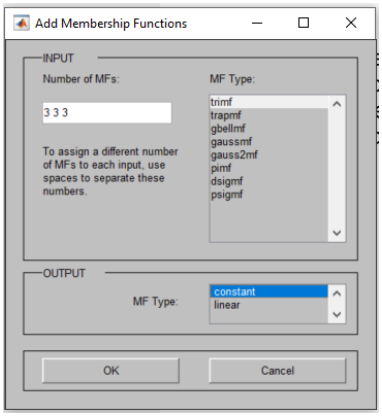


Figure 17: Configuración de funciones miembro

(a) Debe tenerse en cuenta que si el número de *epochs* es excesivo, puede darse *overfitting*, y que por tanto el modelo sea inútil ante cualquier caso que sea mínimamente diferente al usado para el entrenamiento - memoriza, no aprende.

5. Puede comprobarse el controlador generado contra los datos de entrenamiento, solo es necesario pulsar el botón *Test Now*, manteniendo seleccionada la opción *Traning Data* (figura [19]).

6. Por último, tras haber diseñado el controlador, es necesario guardarlo: *File/Export/To File*. El nombre elegido es `anfisV.fis` para la velocidad lineal y `anfisW.fis` para la velocidad angular [10].

Como cabe esperarse, esto implica que ya no puede emplearse tan solo un controlador. Es necesario duplicar el controlador, y usar un archivo *.fis* en cada uno de ellos - uno para el de la velocidad lineal y otro para la velocidad angular.

Dentro de la carpeta `Part2_NeuralFuzzy` puede encontrarse el fichero `test_controller_neuralfuzzy.slx`, representado por la siguiente figura [20]:

[10] A tenerse en cuenta, solo se ha mostrado el proceso con la velocidad angular. El proceso para la velocidad lineal es el mismo, solo que importando los datos correspondientes a esta.

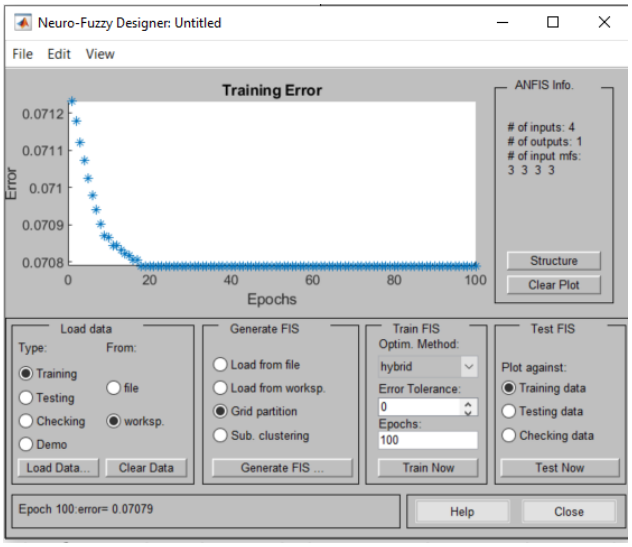


Figure 18: Error de entrenamiento velocidad angular

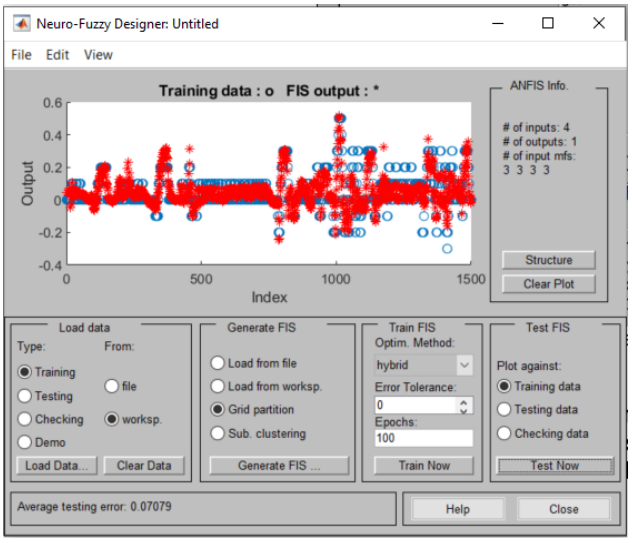


Figure 19: Resultado de entrenamiento en el conjunto de velocidad angular

Sin Obstáculos

De la misma manera que con el controlador MAMDANI, solamente son necesarios 2 sónares para poder superar este circuito (7). Se emplean los sónares `sonar_0` y `sonar_2`.

El robot es capaz de completar el circuito.

Con Obstáculos

Al igual que el correspondiente controlar MAMDANI, se debe superar el circuito de la figura [11].

También se emplean los sónares `sonar_0`, `sonar_2` y `sonar_3`.

El robot es capaz de completar el circuito.

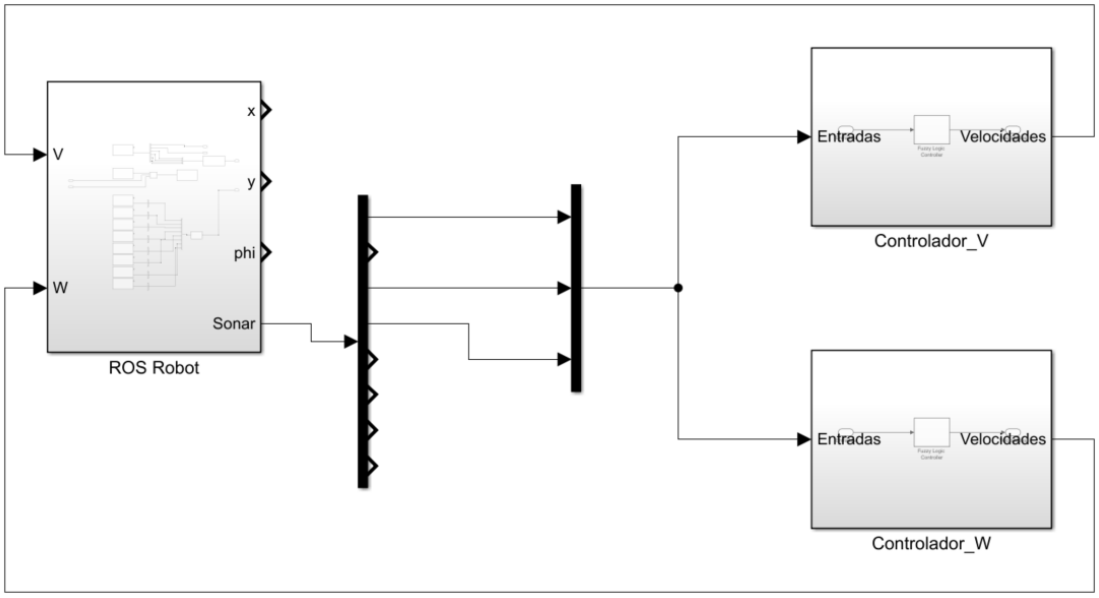


Figure 20: Esquema controlador neuro-borroso

5 Conclusión

Con los experimentos realizados se puede llegar a la conclusión de que el controlador borroso **ofrece una solución mejor** al controlador neuro-borroso.

Dentro del desarrollo de los controladores, y como matiz general, es fácil comprender que al meter obstáculos dentro del circuito sean necesarios más sensores. Ya no porque "haya obstáculos", sino porque cabe esperarse que parte de esos obstáculos aparezca en el lado derecho del robot; el cual no es necesario monitorizar cuando simplemente se está recorriendo el circuito - sin obstáculos.

El controlador borroso ha sido más sencillo de desarrollar, además de que para pasar de un circuito **sin** obstáculos a uno **con** obstáculos no ha sido necesario repetir todo el proceso anterior, sino meter un nuevo sensor, añadir nuevas reglas y retocar algunas funciones de pertenencia (mínimamente).

Esto se puede deberse a que el controlador neuro-borroso consta de datos provenientes de un experimento a partir del cual se obtiene el controlador, es decir, hay que repetir el experimento a cada nuevo controlador, o conseguir uno suficientemente genérico. Para ello, es necesario que un operador recoja los datos de manera manual. En este caso, supondría realizar cientos - quizás miles - de vueltas al mismo circuito, y a otros similares para conseguir un controlador que **quizás** sea capaz de adaptarse a cualquier circuito[11].Solamente para completar esta práctica, se ha necesitado toda una tarde para desarrollar el controlador neuro-borroso funcional empleado; de manera que el robot no colisionase contra una pared u objeto.

Por otro lado, el controlador borroso, ofrece la posibilidad de dar una serie de normas o reglas que, hechas con la suficiente cautela y precisión, son capaces de adaptarse a cualquier circuito.

Es cierto que dependen de la imaginación del creador de dichas normas/reglas, pero sigue siendo más rápido (a la hora de desarrollarlo) y adaptable que tener que dar un número extremadamente alto de vueltas controlando el robot.

Como apunte final, comentar que los controladores capaces de completar el circuito con obstáculos son además capaces de completar los circuitos sin obstáculos. Por lo que pueden usarse para ambos circuitos, en lugar de tener que mantener dos controladores. Bien es cierto, que el exceso de sensores pueda hacer que el robot vaya más despacio (al detectar la pared exterior como un obstáculo relativamente cercano).

[11] Con las mismas características de tamaño y forma pero con diferente posicionamiento de los obstáculos.