PRÁCTICA FINAL: Diseño de Controladores Borrosos y Neuronales para la maniobra de aparcamiento de un vehículo

Sistemas de Control Inteligente

Alba Calvo Herrero
María Sanz Espeja

Grupo 11 - A3 lunes
Grado Ingeniería Informática

Índice

Introducción

El propósito principal de esta práctica es desarrollar y afinar un sistema de control para la velocidad lineal y angular de un vehículo, con el fin de optimizar su capacidad para realizar maniobras de aparcamiento de manera efectiva y en el menor tiempo posible. Esta tarea implica un profundo entendimiento tanto de la dinámica del vehículo como del entorno en el que debe operar.

Descripción del Entorno de Aparcamiento:

La maniobra de aparcamiento debe llevarse a cabo en un entorno específicamente diseñado, cuyas dimensiones y características se detallan a continuación:

- Dimensiones del entorno: El área designada para el aparcamiento tiene unas dimensiones totales de 24 x 13,4 metros, incluyendo las paredes que delimitan el espacio.
- Ancho del carril: El carril por el que se desplaza el vehículo mide 4,3 metros de ancho, lo que plantea ciertos retos en términos de maniobrabilidad y precisión en el control.
- Ubicación de la plaza de aparcamiento: La plaza de aparcamiento, que mide 7,8 x 4,3 metros, se sitúa a la izquierda del vehículo. Es importante destacar que el vehículo comienza la maniobra estando de espaldas a la plaza.
- Criterio de aparcamiento ideal: La maniobra se considera exitosamente completada cuando el vehículo logra posicionarse de manera centrada dentro de la plaza, tanto longitudinal como lateralmente, y además se encuentra perfectamente alineado con el borde de la plaza.

Estructura del entregable:

El entregable de la práctica se divide en dos carpetas:

- Matlab
  - Controlador borroso:
    - ackerman_parking_bateria_v4.fis
    - ackerman_ROS_controller_v2.slx
  - Controlador neuronal
    - ackerman_ROS_controller_v2_neuronal.slx
    - avanzar_ackerman.m
    - datos entrenamiento.mat
    - function_conversion_steering_to_linear_angular.m
    - Key_Down_sensores_V_steering
    - maniobra_park_ackerman_datos entrenamiento _alumnos.m
    - neuronal.m
    - parking_data.mat

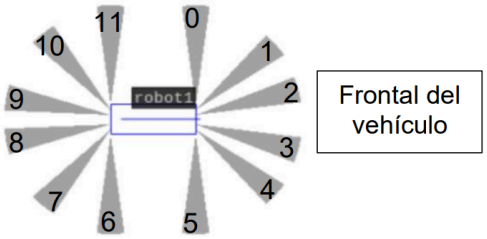
- STDR
  - Archivo de simulación del vehículo:
    - emulated_ackerman.xml
  - Archivos del mapa:
    - parking1600x895.yaml
    - parking1600x895.png
  - Archivo “launch” que lanza la simulación del mapa y el vehículo:
    - SCI-parking_ackerman_batería.launch
  - Para lanzar la simulación del mapa y el vehículo:

- roslaunch stdr_launchers SCI-parking_ackerman_bateria.launch

Vehículo

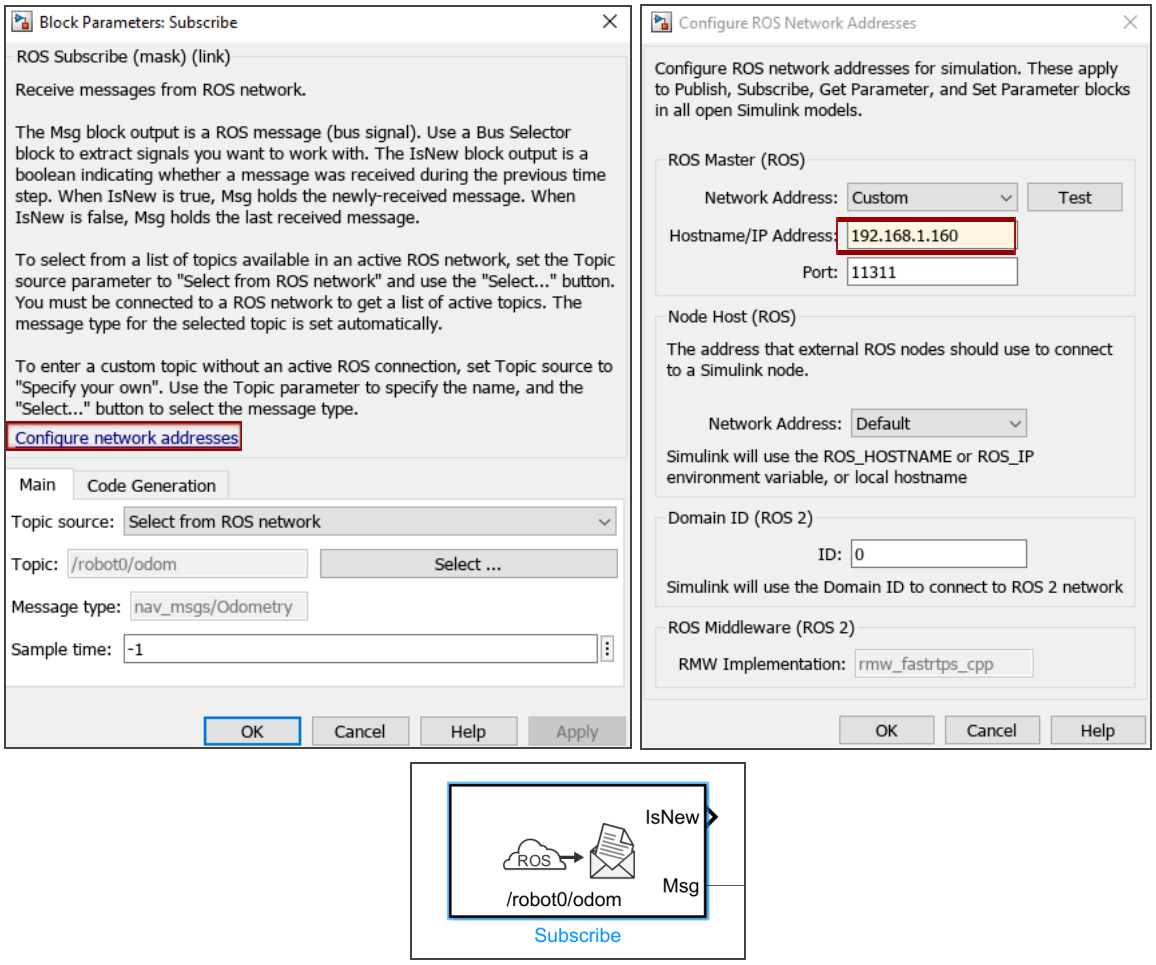
El vehículo posee una configuración de **Ackerman**, con dimensiones de 4.5 metros de longitud y 1.5 metros de anchura. Es capaz de alcanzar una velocidad lineal máxima de 30 km/h y puede girar su volante hasta 90 grados. Sin embargo, no puede cambiar su orientación sin movimiento lineal, una característica inherente a la configuración de Ackerman.

Para la detección y la navegación, el vehículo está equipado con **12 sensores** de ultrasonidos llamados sonar que tienen un rango de 0 a 5 metros.



Configuración de ROS y Matlab

La simulación del comportamiento del vehículo y su interacción con el entorno se lleva a cabo en la plataforma **ROS-STDR**, y para el correcto funcionamiento de la simulación, hemos tenido que poner la dirección IP de la máquina virtual, **192.168.1.160**, en la dentro de un bloque subscribe del ROS Robot en Simulink en el archivo ackerman_ROS_controller_v2.slx como podemos observar en la siguiente imagen.



Este proceso será el mismo para el archivo del controlador neuronal.

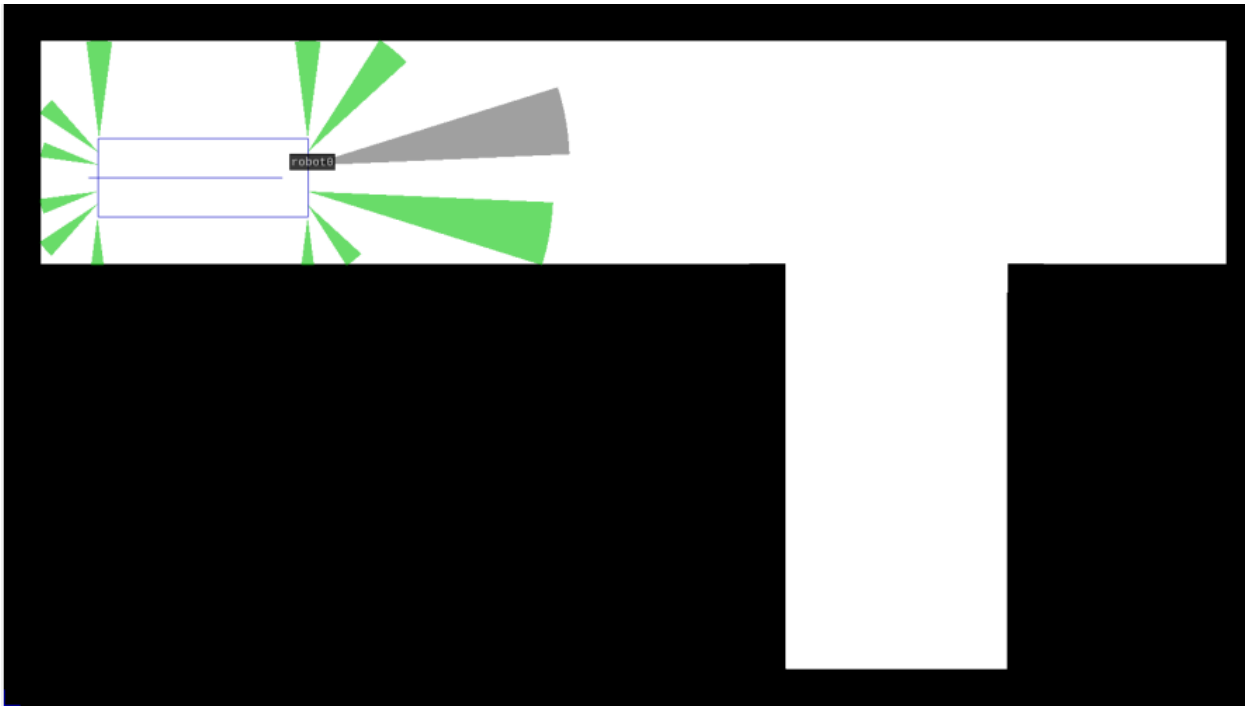
Además se han tenido que introducir los archivos dados por la asignatura de STDR en las carpetas correspondientes.

Entorno original y entorno definitivo:

Los controladores borroso y neuronal deben funcionar en ambos entornos. Para cambiar el entorno del original al definitivo se debe cambiar la posición inicial del robot en el archivo ***SCI-parking_ackerman_bateria.launch*** cambiando los parámetros **5 10.5 3.14159** por **5.3 10.1 3.14159.**

SCI-parking_ackerman_bateria.launch

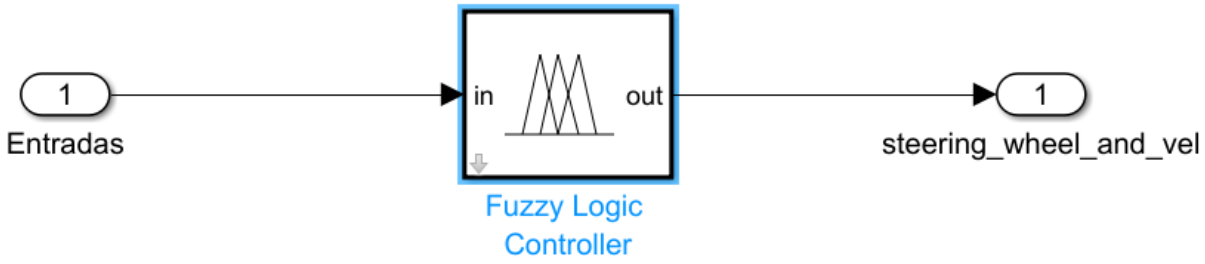
```
<launch>

    <include file="$(find stdr_robot)/launch/robot_manager.launch" />

    <node type="stdr_server_node" pkg="stdr_server" name="stdr_server" output="screen" args="$(find stdr_resources)/maps/-
parking1600x895.yaml"/>

    <node pkg="tf" type="static_transform_publisher" name="world2map" args="0 0 0 0 0 0  world map 100" />

    <include file="$(find stdr_gui)/launch/stdr_gui.launch"/>

    <node pkg="stdr_robot" type="robot_handler" name="$(anon robot_spawn)" args="add $(find stdr_resources)/resources/robots/-
emulated_ackerman.xml 5.3 10.1 3.14159" />

</launch>
```


robot0

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

1. Objetivo y descripción del sistema

El objetivo es crear un controlador que interprete las señales de los sensores de ultrasonidos del vehículo y ajuste en consecuencia la velocidad lineal y el ángulo de giro del volante para realizar una maniobra de aparcamiento eficiente.



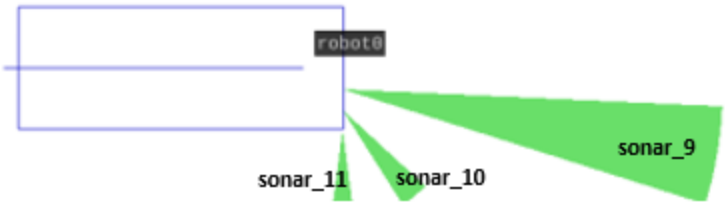
Para desarrollar nuestro controlador borroso, seguimos estos pasos:

1. Seleccionamos los sensores de ultrasonidos necesarios para detectar los obstáculos durante el aparcamiento.
2. Creamos el sistema borroso especificando primero los inputs y outputs y luego formulando las reglas borrosas.
3. En Simulink, conectamos los puertos correspondientes a los sensores elegidos con el controlador borroso.
4. Finalmente, ajustamos la condición de parada en el modelo para que refleje el fin exitoso de la maniobra de aparcamiento.

2. Desarrollo de la práctica

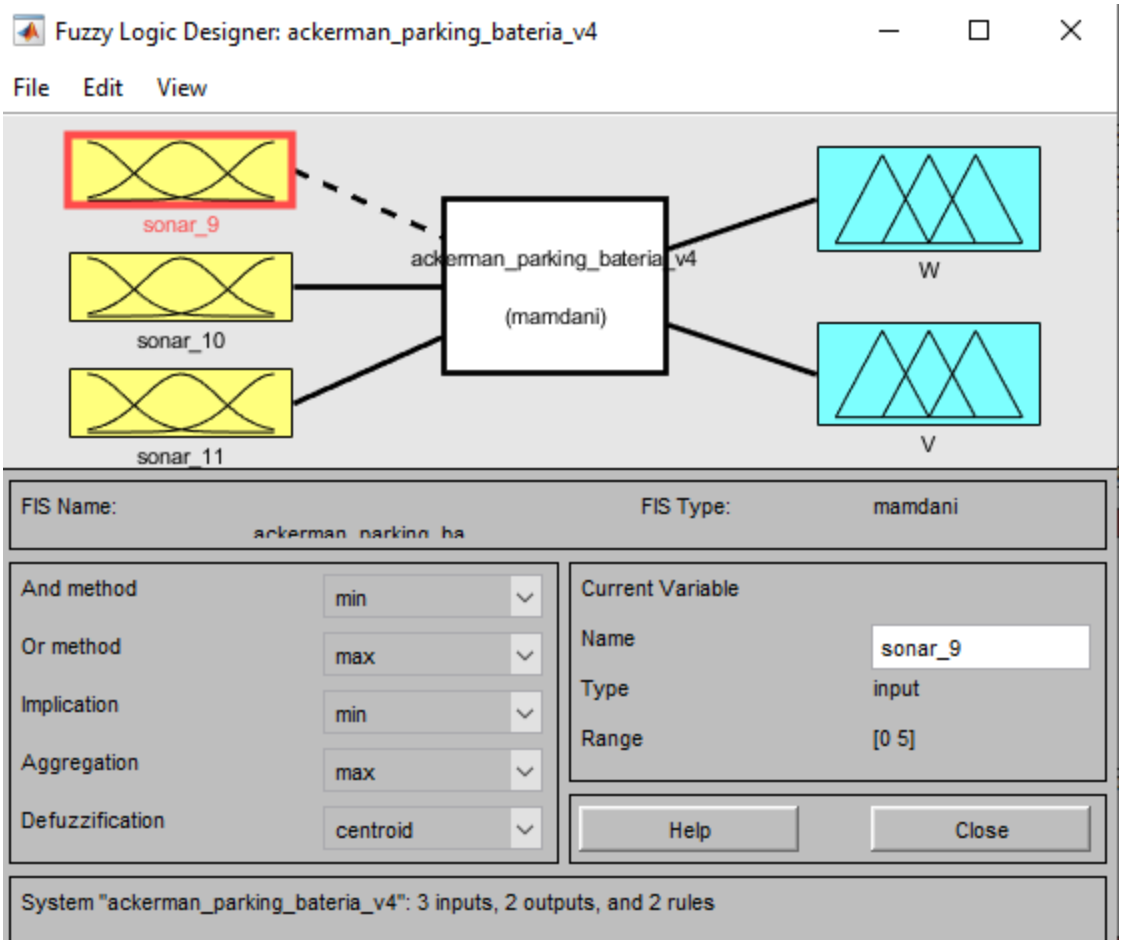
Selección de sensores

Para el desarrollo de la práctica se ha decido utilizar los sonares **9, 10 y 11**. Con ellos se evita que el coche choque y se detecta cuando hay un espacio para poder aparcar a la derecha del coche. Se adjunta una foto de los sensores utilizados:



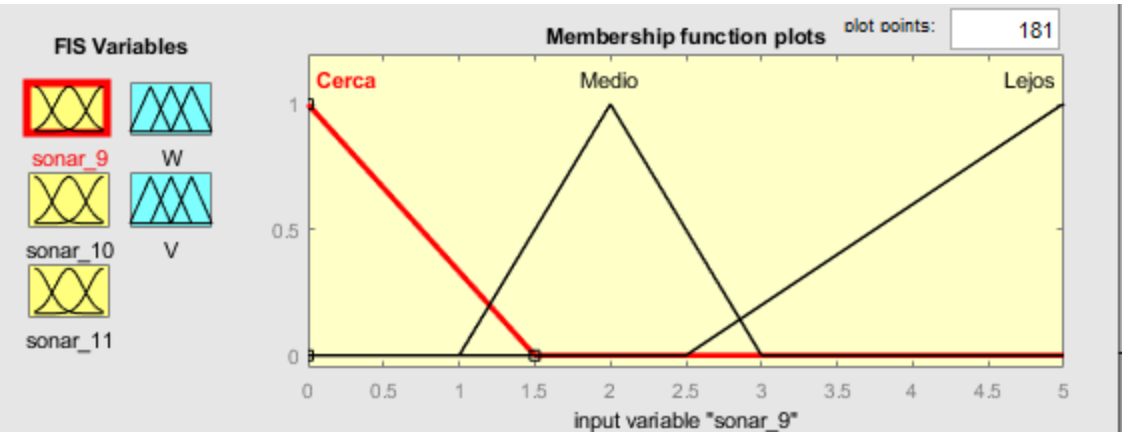
Creación del sistema borroso

Creamos el sistema borroso especificando primero los inputs y outputs y luego formulando las reglas borrosas. Ahora se va a analizar más en detalle, visualizando los datos en Fuzzy:



Se observa los tres sonar mencionados anteriormente y la velocidad angular y velocidad lineal, todos ellos unidos al archivo ackeman_parking_bateria_v4 que es un sistema de control difuso utilizando la lógica difusa tipo Mamdani.

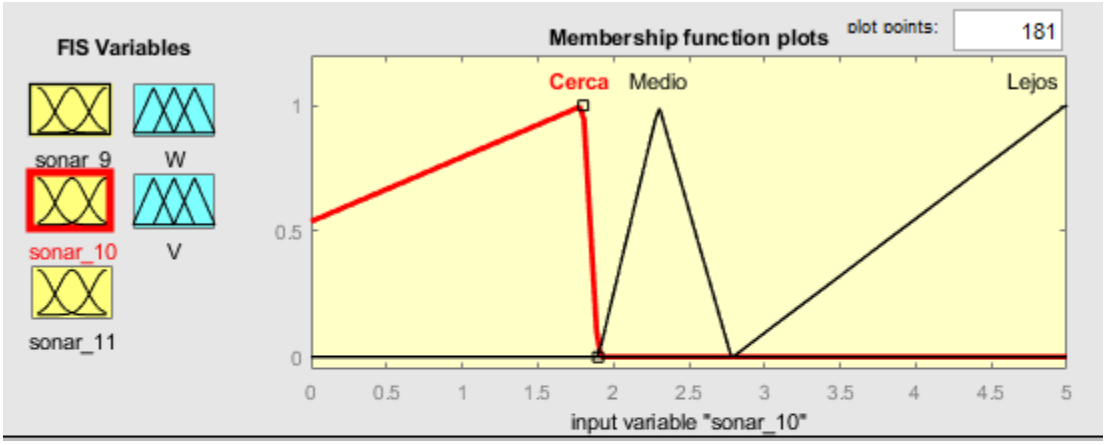
El primero sonar que se encuentra es el **sonar 9**:



Entrada 1 ("sonar_9"):
Rango de valores: [0, 5]
Número de funciones de membresía (NumMFs): 3
Funciones de Membresía:

- "Cerca" (trimf): [0, 0, 1.5]
- "Medio" (trimf): [1, 2, 3]
- "Lejos" (trimf): [2.5, 4.99, 20]

El siguiente es el **sonar 10**:

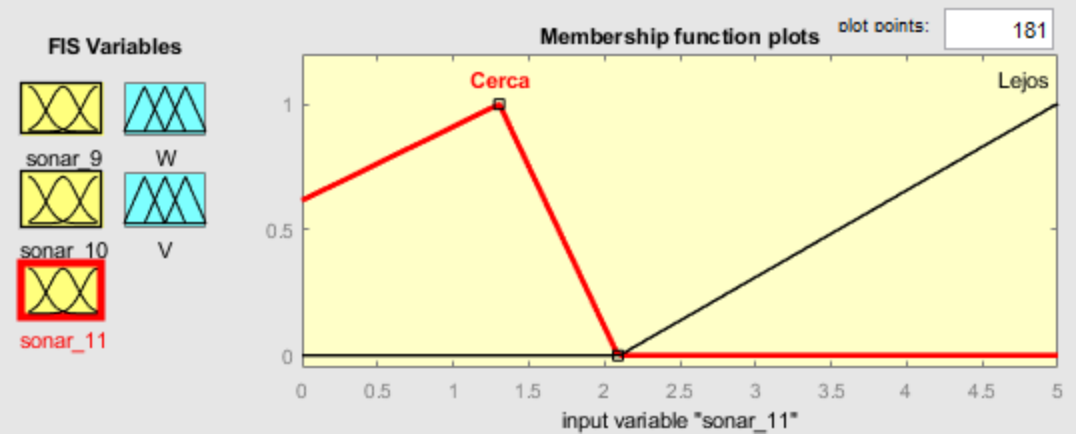


Entrada 2 ("sonar_10"):
Rango de valores: [0, 5]
Número de funciones de membresía (NumMFs): 3
Funciones de Membresía:

- "Cerca" (trimf): [-2.1, 1.8, 1.899]
- "Medio" (trimf): [1.9, 2.3, 2.78]
- "Lejos" (trimf): [2.79, 4.99, 20]

En este caso, los decimales contienen más decimales para ser más precioso los giros.

El siguiente es el **sonar 11**:



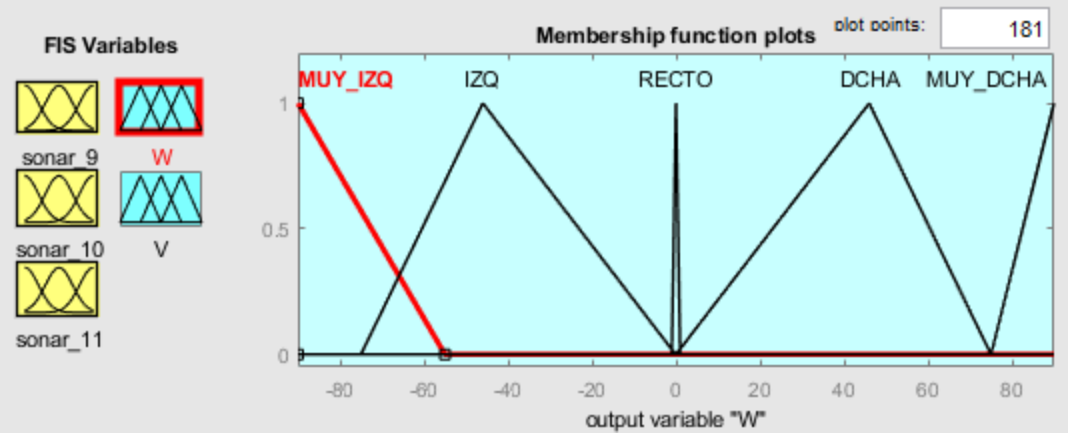
Rango de valores: [0, 5]

Número de funciones de membresía (NumMFs): 2

Funciones de Membresía:

- "Cerca" (trimf): [-2.1, 1.305, 2.09]
- "Lejos" (trimf): [2.1, 4.99, 20]

Una vez finalizados los sensores, se observará las velocidades. El primero que se tiene es la **velocidad angular**:



Salida 1 ("W"):

Rango de valores: [-90, 90]

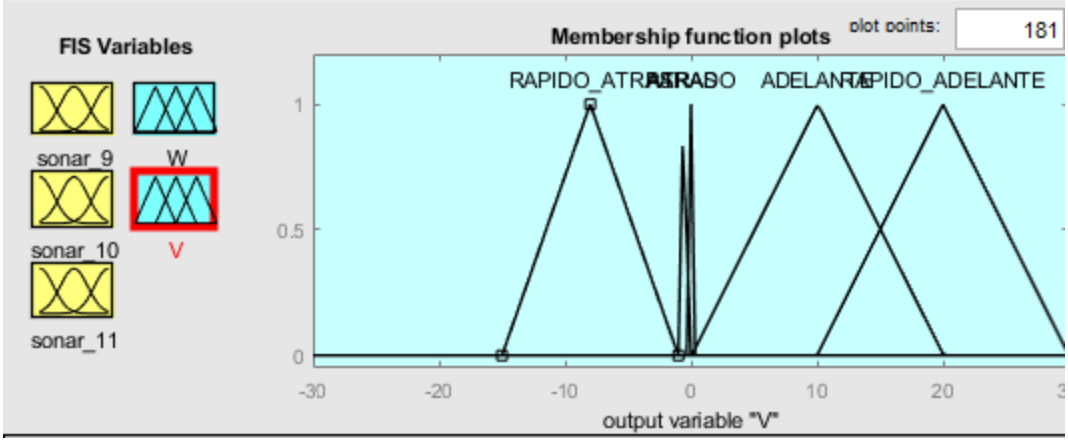
Número de funciones de membresía (NumMFs): 5

Funciones de Membresía:

- "MUY_IZQ" (trimf): [-90, -90, -55]
- "IZQ" (trimf): [-75, -46, -0.1]
- "RECTO" (trimf): [0, 0, 0]
- "DCHA" (trimf): [0, 46, 75]
- "MUY_DCHA" (trimf): [75, 90, 90]

Aunque en este trabajo sólo se utilice muy derecha, un giro comprendido entre 75 y 90 grados hacia la derecha, y recto; se implementan los demás valores de posibles giros por si en un futuro fuese necesario utilizarlo.

Por último se encuentra la **velocidad lineal**:



Salida 2 ("V"):

Rango de valores: [-30, 30]

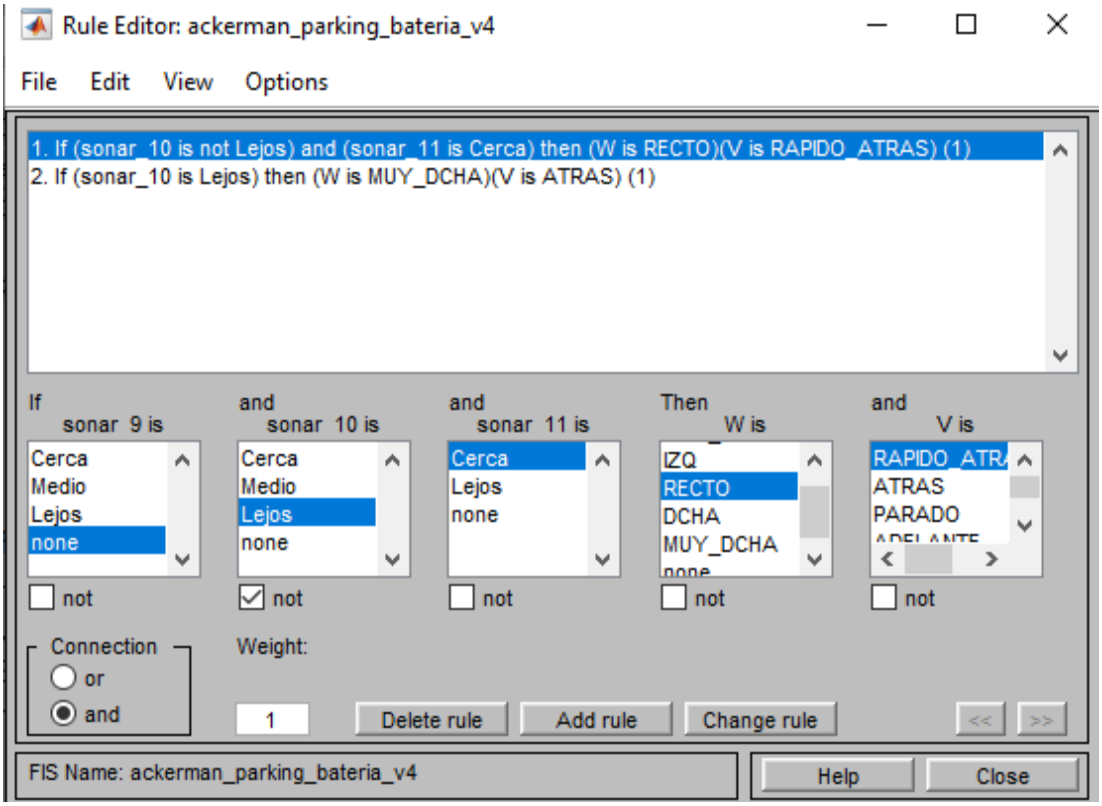
Número de funciones de membresía (NumMFs): 5

Funciones de Membresía:

- "RAPIDO_ATRAS" (trimf): [-15, -8, -1]
- "ATRAS" (trimf): [-1, -0.6, -0.1]
- "PARADO" (trimf): [-0.1, 0, 0.06]
- "ADELANTE" (trimf): [0.06, 10.06, 20.06]
- "RAPIDO_ADELANTE" (trimf): [10, 20, 30]

Al igual que en la velocidad angular, se implementan diferentes velocidades aunque no sean utilizadas.

Para un correcto funcionamiento, se implementan unas reglas que el robot deberá de seguir:



Regla 1:

- Condición: Si sonar_10 no está Lejos y sonar_11 está Cerca.
- Consecuencia: Entonces W es RECTO y V es RAPIDO_ATRAS.
- Explicación: Esta regla se activa cuando los sonares 10 y 11 detectan cerca un obstáculo en la parte inferior. De esta forma el robot puede ir rápido hacia atrás de forma recta.

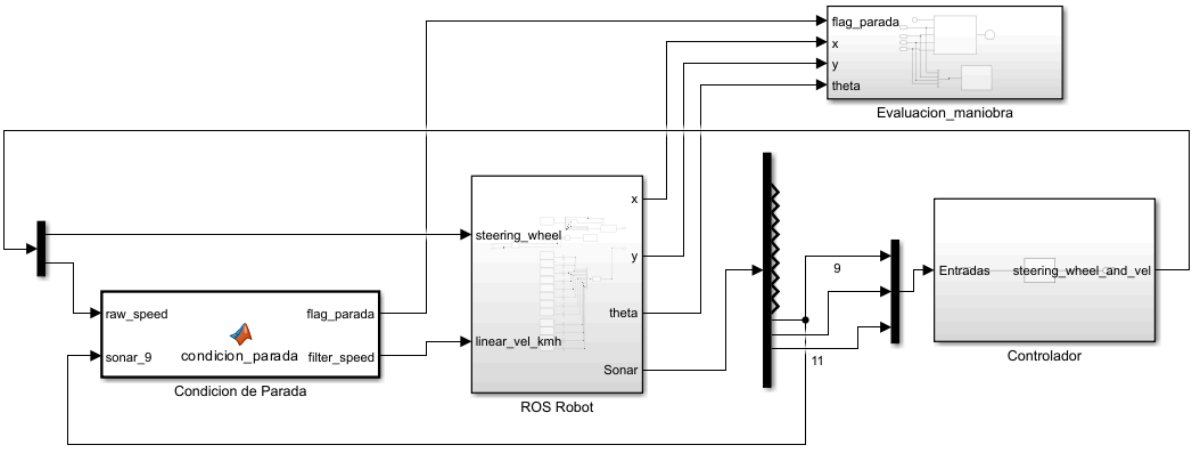
Regla 2:

- Condición: Si sonar_10 está Lejos.
- Consecuencia: Entonces W es MUY_DCHA y V es ATRAS
- Explicación: Esta regla se activa cuando el sonar 10 detecta obstáculos lejos, es decir, cuando detectan un hueco para aparcar. El vehículo debe girar muy a la derecha (W=MUY_DCHA) y avanzar rápidamente hacia atrás (V=ATRAS).

Estructura simulador controlador borroso

Mencionar que para su correcto funcionamiento se ha debido de conectar anteriormente los puertos 10, 11 y 12 de los sonar al controlador.

Así, al emplear el esquema proporcionado en los archivos de la práctica, la estructura de nuestra red se representa de la siguiente manera:



Condición de parada

Finalmente, para explicar la condición de parada que se tiene en el controlador borroso, disponemos de la siguiente función, la cual se basa únicamente en la información del sonar_9. Esta función evalúa la distancia necesaria para posicionar el robot correctamente en el espacio destinado al estacionamiento. Si la condición se satisface, se activará la señal de parada, y el vehículo procederá a detenerse. La distancia seleccionada es 2.1, es decir, una vez el sonar_9 detecte una distancia menor se detendrá.

```
function [flag_parada, filter_speed] = condicion_parada(raw_speed, sonar_9)
% Condicion de parada:
% - debeis decidir cuando se finaliza la maniobra de aparcamiento en base
%  a la información disponible:
%       - velocidad_lineal
%       - ángulo de volante
%       - sensores
% Una vez que decidais cual es el criterio de parada debeis:
%    - Parar el vehículo con v_lineal = 0.
%    - Activar el flag_parada a 1 para indicar que ha finalizado la maniobra.
if sonar_9 < 2.1
   %Es necesario parar el vehículo con v_lineal = 0 y activar el
   %flag_parada a 1 para indicar que ha finalizado la maniobra.
   filter_speed = 0.0;
   flag_parada=1;
else
   filter_speed = raw_speed;
   flag_parada=0;
end
```

Parte 2. Diseño automático de un controlador neuronal

1. Objetivo y descripción del sistema

El objetivo de esta sección es desarrollar un controlador neuronal que automatice la maniobra de aparcamiento de nuestro vehículo. El enfoque se centra en utilizar técnicas de aprendizaje automático para que el sistema aprenda a partir de ejemplos de entrenamiento generados a través del telecontrol.

Pasos para el Diseño del Controlador Neuronal:

1. **Generación de Datos de Entrenamiento**: Utilizamos el **telecontrol** para operar manualmente el vehículo y generar un conjunto de datos que capturan la relación entre las lecturas de los sensores y las acciones de control correspondientes.
2. **Construcción y Entrenamiento de la Red Neuronal**: Creamos la arquitectura de la red neuronal y la entrenamos con el conjunto de datos de entrenamiento, ajustando los parámetros para mejorar la precisión de las predicciones de control.
3. **Integración con el Controlador**: Incorporamos la red neuronal entrenada en nuestro modelo de control para que el vehículo pueda realizar la maniobra de aparcamiento de forma autónoma basándose en las entradas sensoriales.
4. **Evaluación y Ajuste**: Probamos el controlador neuronal en el entorno de simulación para evaluar su desempeño

Este enfoque nos permite crear un controlador que no solo sigue reglas predefinidas, sino que también tiene la capacidad de aprender y mejorar su comportamiento a través de la experiencia, lo cual es fundamental para adaptarse a las complejidades del entorno de aparcamiento real.

2. Desarrollo de la práctica

Generación de datos de entrenamiento

En nuestro caso hemos usado el telecontrol como generador de datos de entrenamiento. Debido a que no nos dejaba avanzar más de 3.5 metros sin error, hemos hecho un total de 6 avances. Este código en MATLAB establece una conexión con un entorno de simulación ROS, controla un vehículo tipo Ackerman mediante comandos predefinidos y recopila datos de sensores y movimientos para entrenamiento. Inicia cerrando la sesión ROS actual, configura suscriptores para odometría y sonares, y un publicador para comandos de movimiento. Ejecuta una serie de maniobras específicas, registrando los datos en 'training_data', y finalmente guarda estos datos para su posterior uso en entrenamiento de modelos de control.

maniobra_ackerman_datos_entrenamiento_alumnos.m

```
%% conectar
%**********************
rosshutdown
clear all
close all
rosinit('192.168.1.160')
global steering_wheel_angle;
global vel_lineal_ackerman_kmh;
%% ini_simulador_ACKERMAN
%**********************
%DECLARACIÓN DE SUBSCRIBERS
%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%Odometria
sub_odom=rossubscriber('/robot0/odom');
%Laser
%sub_laser = rossubscriber('/robot0/laser_1', rostype.sensor_msgs_LaserScan);
%Sonars
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);
sonar_8 = rossubscriber('/robot0/sonar_8', rostype.sensor_msgs_Range);
sonar_9 = rossubscriber('/robot0/sonar_9', rostype.sensor_msgs_Range);
sonar_10 = rossubscriber('/robot0/sonar_10', rostype.sensor_msgs_Range);
sonar_11 = rossubscriber('/robot0/sonar_11', rostype.sensor_msgs_Range);
%DECLARACIÓN DE PUBLISHERS
%%%%%%%%%%%%%%%%%%%%%%%%%%
%Velocidad
pub_vel=rospublisher('/robot0/cmd_vel','geometry_msgs/Twist');
%GENERACION DE MENSAJES
%%%%%%%%%%%%%%%%%%%%%%%
msg_vel=rosmessage(pub_vel);
%Definimos la periodicidad del bucle
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
r=robotics.Rate(10);
%Nos aseguramos de recibir un mensaje relacionado con el robot
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
while (strcmp(sub_odom.LatestMessage.ChildFrameId,'robot0')~=1)
    sub_odom.LatestMessage
end
disp('Inicialización ACKERMAN finalizada correctamente');
%% **********************
training_data=[];
% Recorrido de aparcamiento para obtener datos de entrenamiento.
%AVANCE 1
distancia= 3.5;
vel_lineal_ackerman_kmh = -7     %(km/h)
steering_wheel_angle =  0       % desde -90 a 90 grados.
avanzar_ackerman
% AVANCE 2
distancia= 3.5;
vel_lineal_ackerman_kmh = -7     %(km/h)
steering_wheel_angle =  0       % desde -90 a 90 grados.
avanzar_ackerman
% AVANCE 3
distancia= 1.8;
vel_lineal_ackerman_kmh = -7     %(km/h)
steering_wheel_angle =  0       % desde -90 a 90 grados.
avanzar_ackerman
% AVANCE 4
distancia= 3;
vel_lineal_ackerman_kmh = -1     %(km/h)
```

```
steering_wheel_angle =  85       % desde -90 a 90 grados.
avanzar_ackerman
% AVANCE 4
distancia= 2.73;
vel_lineal_ackerman_kmh = -1     %(km/h)
steering_wheel_angle =  85       % desde -90 a 90 grados.
avanzar_ackerman
%AVANCE 5
distancia= 3.5;
vel_lineal_ackerman_kmh = -7     %(km/h)
steering_wheel_angle =  0       % desde -90 a 90 grados.
avanzar_ackerman
save datos_entrenamiento training_data
```

Construcción y Entrenamiento de la Red Neuronal

Generamos la red neuronal mediante el script neuronal.m. Primero, seleccionamos los datos de los sonares 9, 10 y 11 como entradas y los valores de velocidad lineal y angular como salidas. Los datos son procesados para reemplazar los valores infinitos y se convierten al tipo double para su uso en la red neuronal.

A continuación, creamos una red neuronal feedforward con tres capas ocultas, cada una con 8 neuronas. Esta elección se basa en numerosas pruebas realizadas con diferentes neuronas, comenzando con valores inferiores y de una sola capa. Sin embargo, observamos que con estas pruebas no se conseguía un rendimiento óptimo, por lo que, investigando en varias páginas web vimos que podíamos ampliar las capas ocultas para un mejor rendimiento. Finalmente, nos quedamos con que obteníamos el mayor rendimiento con 3 capas y 8 neuronas.

Configuramos la red con los datos de entrada y salida y luego la entrenamos. El entrenamiento ajusta los pesos de la red para que las salidas de la red coincidan lo mejor posible con los datos de entrenamiento proporcionados.

Una vez entrenada la red, utilizamos la función *gensim* para generar un modelo de Simulink de la red entrenada. Esto nos permite integrar la red en nuestro entorno de simulación y probar su desempeño en tiempo real.

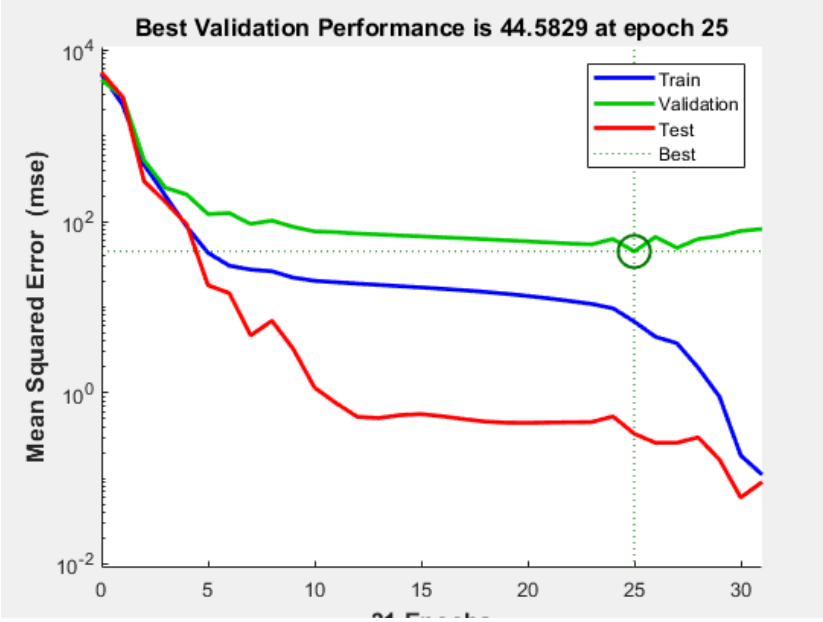
neuronal.m

```
Ts = 100e-3;
% Obtenemos datos de los sonar 9, 10 y 11
inputs = training_data(:,[10,11,12])';
% Obtenemos la salida de la velocidad lineal y angular
outputs = training_data(:,[18, 19])';
% Procesamos los datos
inputs(isinf(inputs)) = 5.0;
inputs = double(inputs);
outputs = double(outputs);
% Generamos la red
net = feedforwardnet([8, 8, 8]);
net = configure(net,inputs,outputs);
% Net training
net = train(net,inputs,outputs);
% Net generation
gensim(net, Ts);
```

Obtenemos los siguientes resultados:

Training Progress

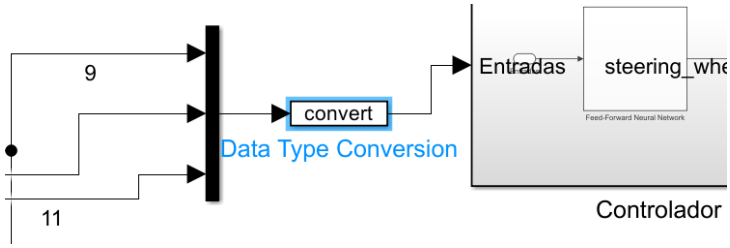
Unit	Initial Value	Stopped Value	Target Value
Epoch	0	31	1000
Elapsed Time	-	00:00:00	-
Performance	5.24e+03	0.11	0
Gradient	1.2e+04	15.9	1e-07
Mu	0.001	0.1	1e+10
Validation Checks	0	6	6



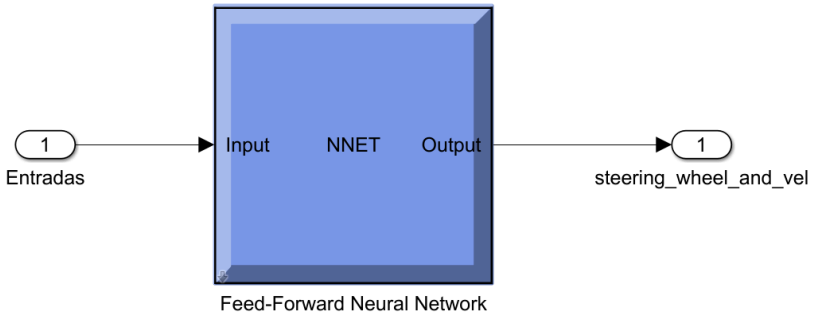
Hemos conseguido un valor de rendimiento de 0.1. Como nuestra velocidad en los datos de entrenamiento varía entre -7 y -1, un error de 0.1 km/h representa aproximadamente un 1,67% del rango total. Esto significa que la predicción de la velocidad está bastante cerca del valor real en la mayoría de los casos.

Integración con el Controlador

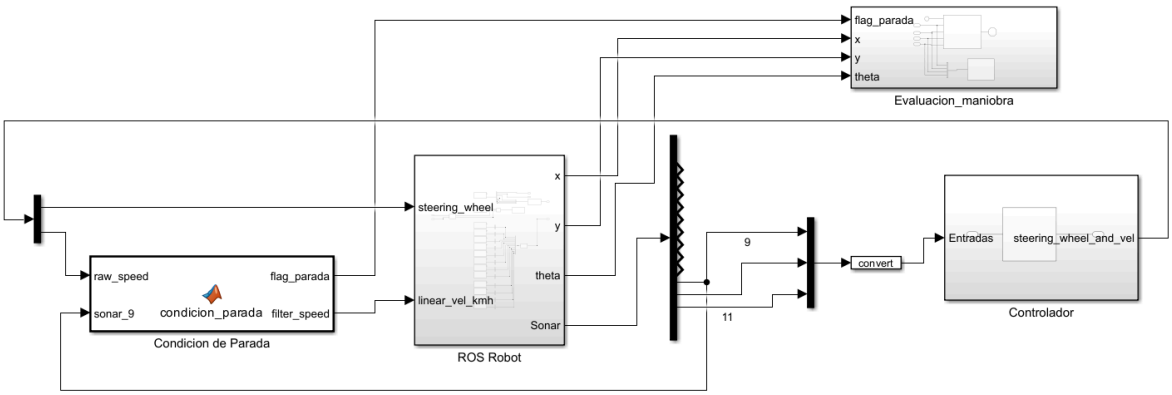
Para poder realizar el control neuronal, se deben conectar los puertos 10, 11 y 12, que corresponden a los sonares 9, 10 y 11 respectivamente, a un multiplexor y de este a un conversor. El conversor es el encargado de convertir los tipos, ya que se trabaja con single y double.



Por último, el bloque del convertidor se conectará al Controlador, donde se cambia el bloque del borroso por un Feed-Forward Neural Network, encargado de simular redes neuronales de tipo feedforward.



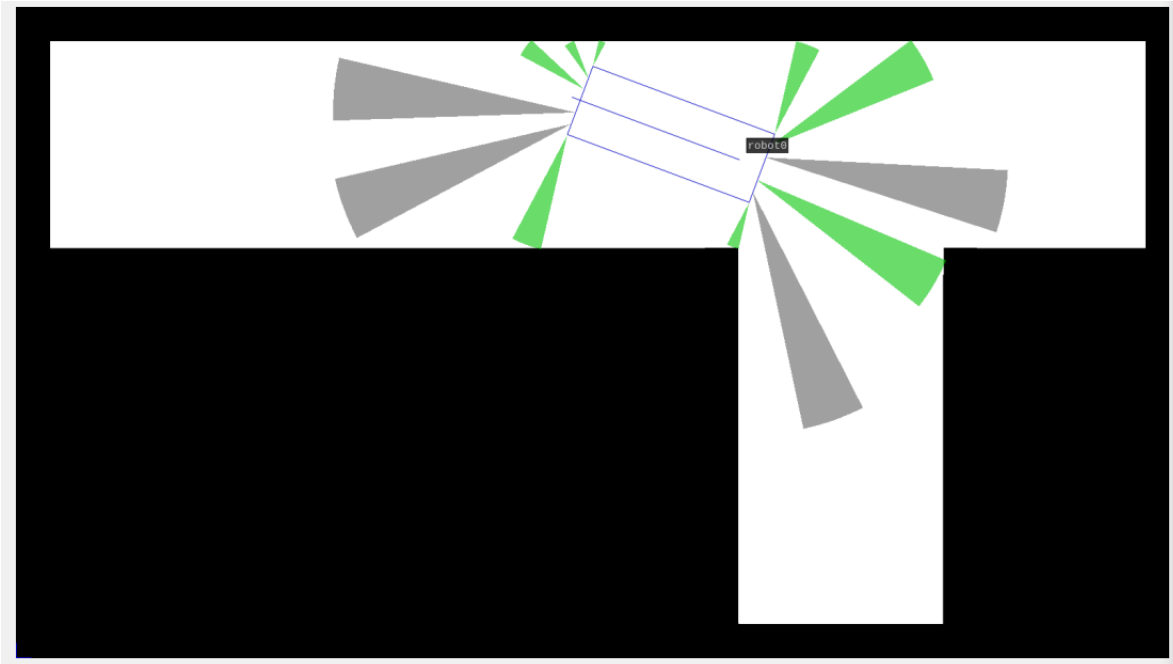
Finalmente el esquema se vería de la siguiente forma:

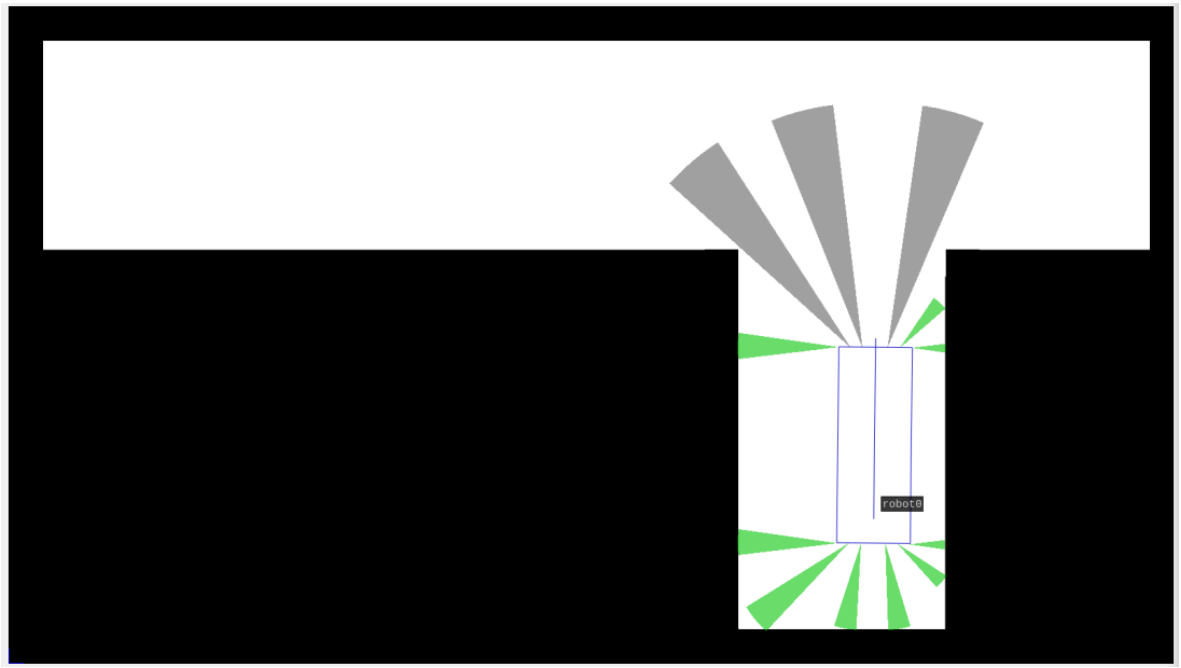


Resultados obtenidos

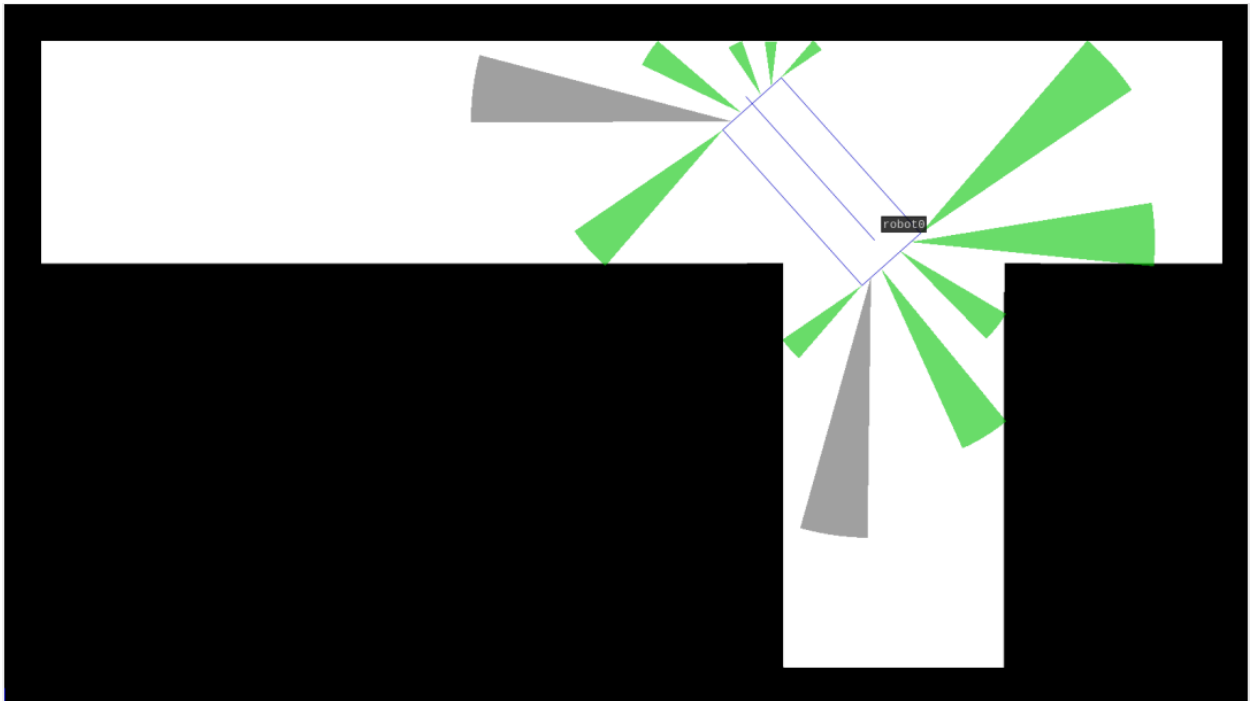
Para el entorno definitivo, no se tuvo que modificar nada en ninguno de los dos controladores ya que desde el principio se realizó una configuración correcta.

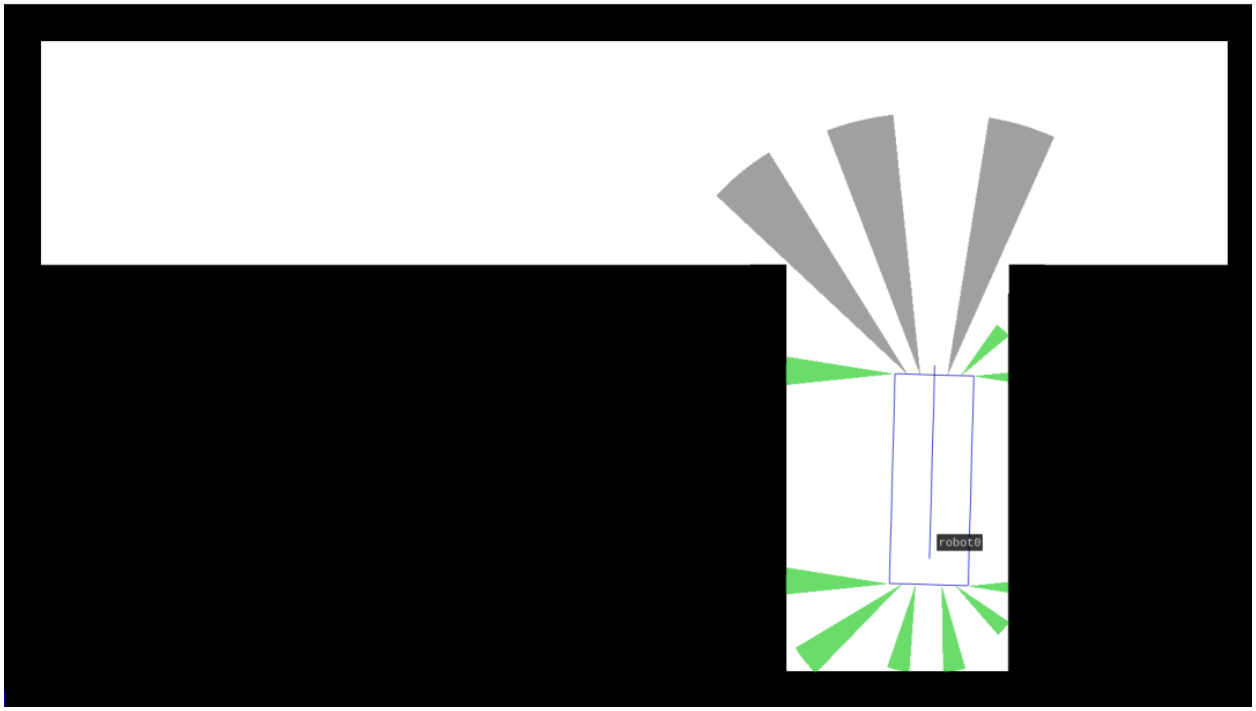
1. Resultado controlador borroso - entorno primero



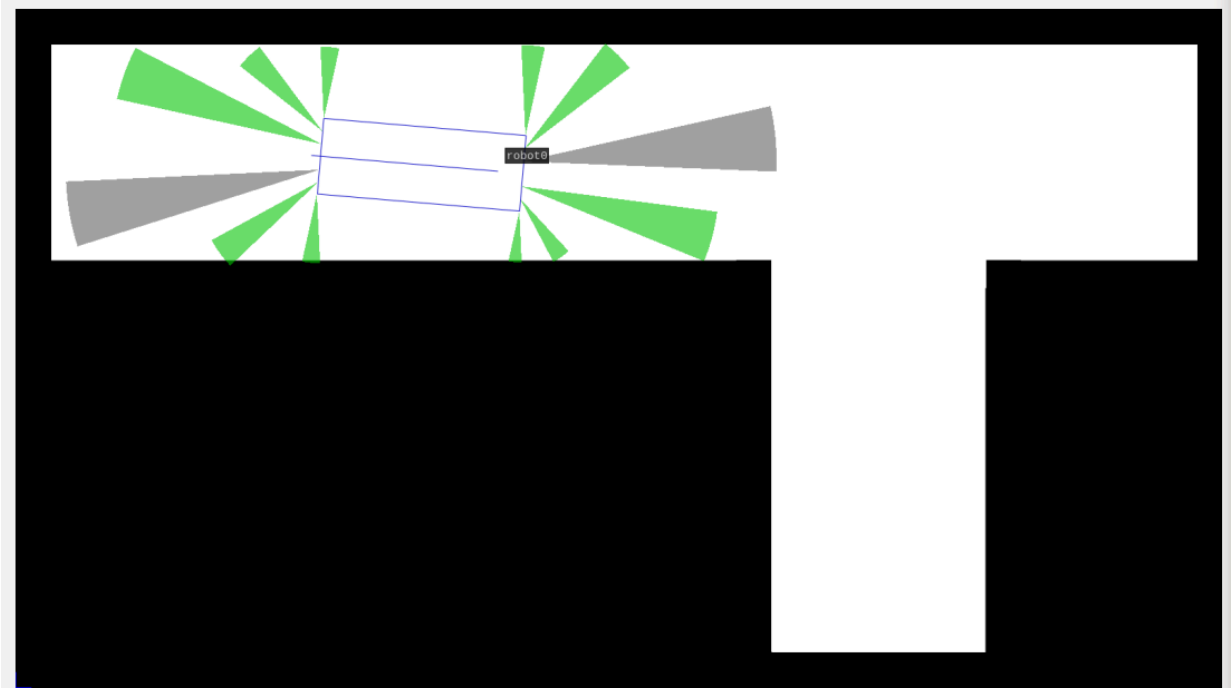


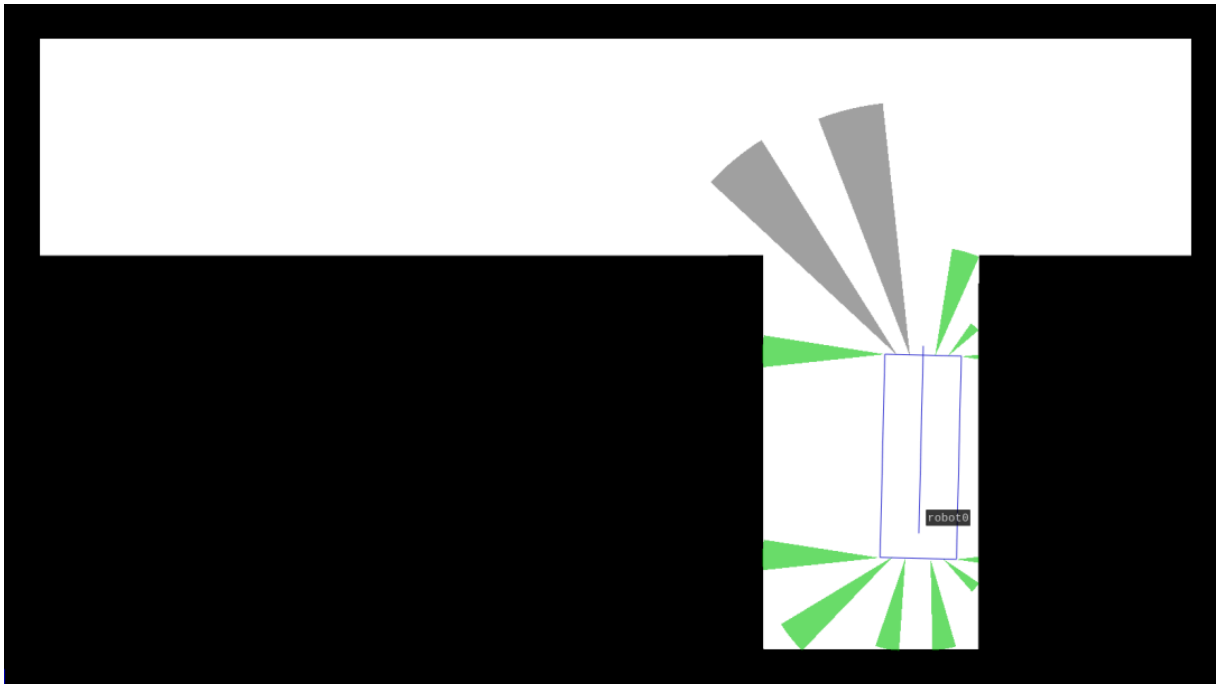
2. Resultado controlador borroso - entorno definitivo



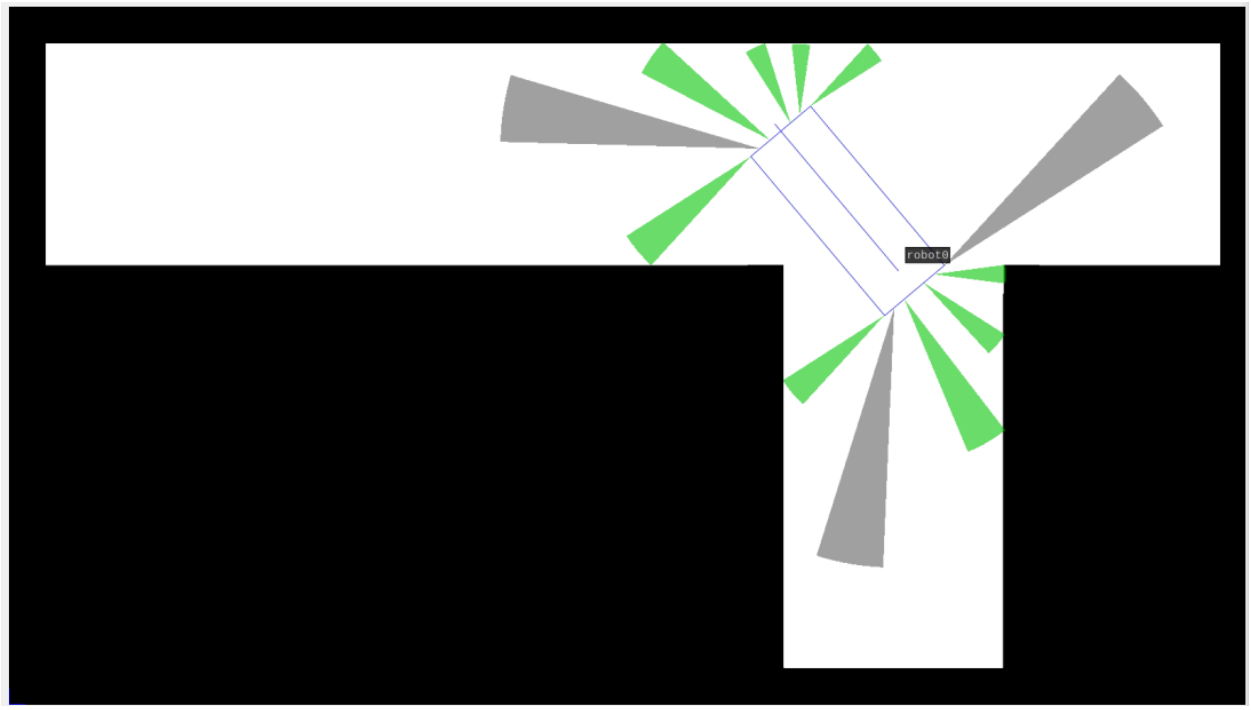


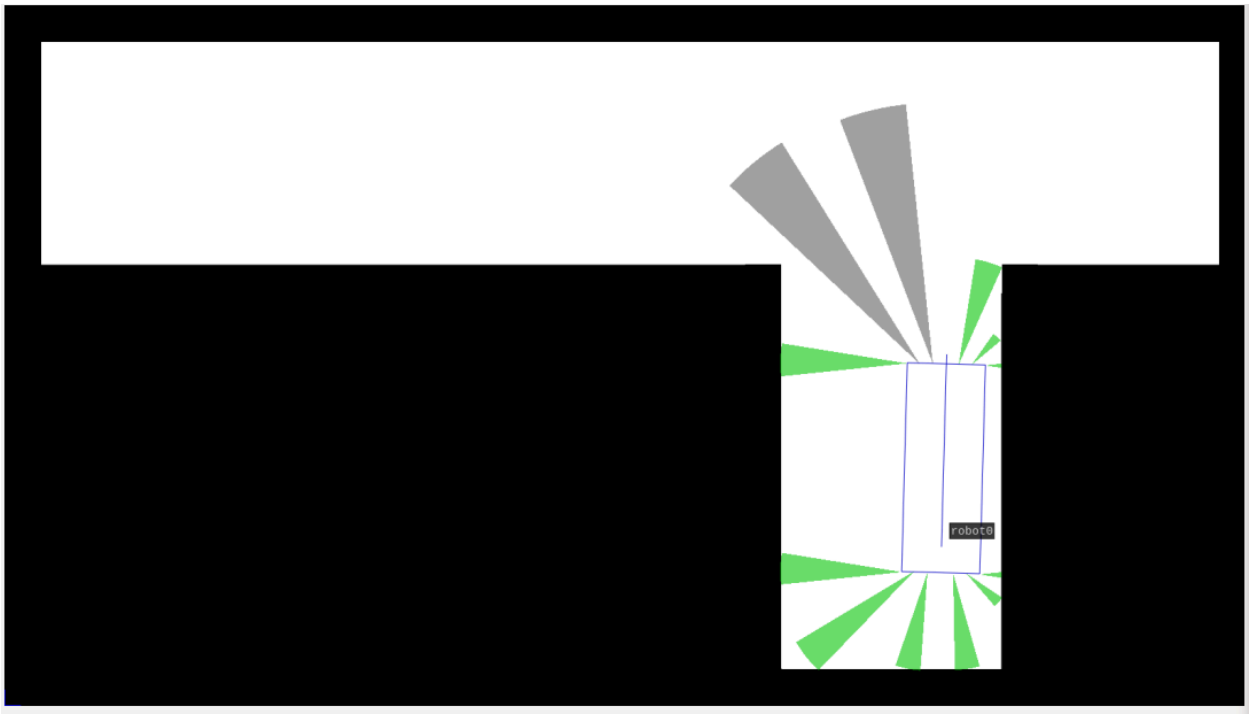
3. Resultado controlador neuronal - entorno primero





4. Resultado controlador neuronal - entorno definitivo


robot0


robot0

Problemas encontrados y soluciones

A lo largo de la práctica, nos hemos enfrentado a diversos inconvenientes. Inicialmente, nos topamos con la incompatibilidad de nuestros portátiles con la máquina virtual, un problema que se resolvió mediante la utilización de un sobremesa más potente.

Además, experimentamos dificultades al intentar evitar colisiones del coche; incluso al establecer reglas o condiciones de parada. El vehículo no se detenía ni dejaba de girar como se esperaba. La solución a este problema consistió en reducir la velocidad, ya que al disminuir la velocidad, las distancias detectadas por los sonares podían procesarse adecuadamente, evitando así colisiones.

Después de disminuir dicha velocidad, nos encontramos con que no era necesario la utilización de tantos sonares ni reglas, obteniendo un robot más óptimo.

También, a la hora de entrenar a nuestra red en la parte del control neuronal, no se detectaba los sonares a tiempo por lo que se tuvo que poner un ‘pause(6)’ para que diese tiempo a cargar todos los sonares sin fallo. Tampoco se podía avanzar más de 3.5 metros sin que diese error, por lo que se tuvo que entrenar en distintos avances.

Además, durante el entrenamiento de nuestra red en la sección de control neuronal, se enfrentó a la problemática de no detectar los datos de los sonares de manera oportuna. Como solución, se implementó un "pause(6)" para permitir la carga completa de los datos de los sonares sin errores. Asimismo, surgió la limitación de no poder avanzar más de 3.5 metros sin ocasionar errores, lo que motivó la necesidad de llevar a cabo el entrenamiento en segmentos de avance distintos.