Universidad de Alcalá

# Práctica Final

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

## Sistemas de Control Inteligente

**Grado en Ingeniería Computadores Grado en Ingeniería Informática Grado en Sistemas de Información**

**Universidad de Alcalá**

**Eduardo García Huerta y Jaime Díez Buendía**

# Desarrollo del trabajo

El desarrollo de la práctica consta de dos partes que se describen a continuación.

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

En cuanto al control borroso, hemos decidido que los sensores que más necesitamos para obtener la información necesaria para poder realizar un aparcamiento en batería son los sónares 5, 7, 8 y 10. En el archivo de simulink "ackerman_ROS_controller" tenemos que conectar las salidas del demultiplexor correspondiente a estos sensores recientemente mencionados con el controlador gracias a un multiplexor, de la siguiente manera:

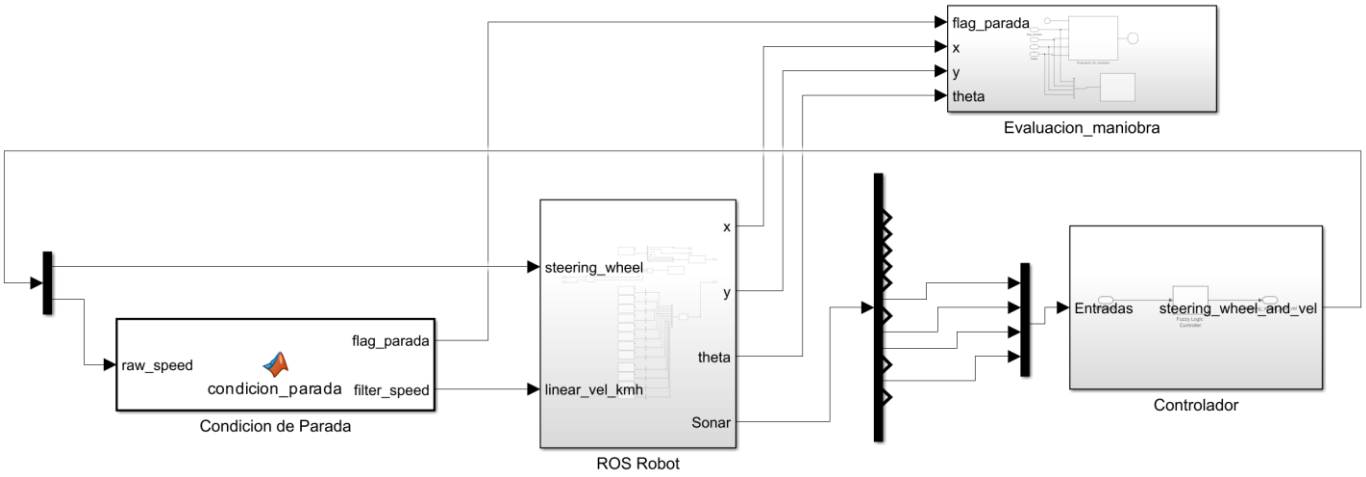



Figura 1. ackerman_ROS_controller.slx

## Archivo. fis del controlador borroso:

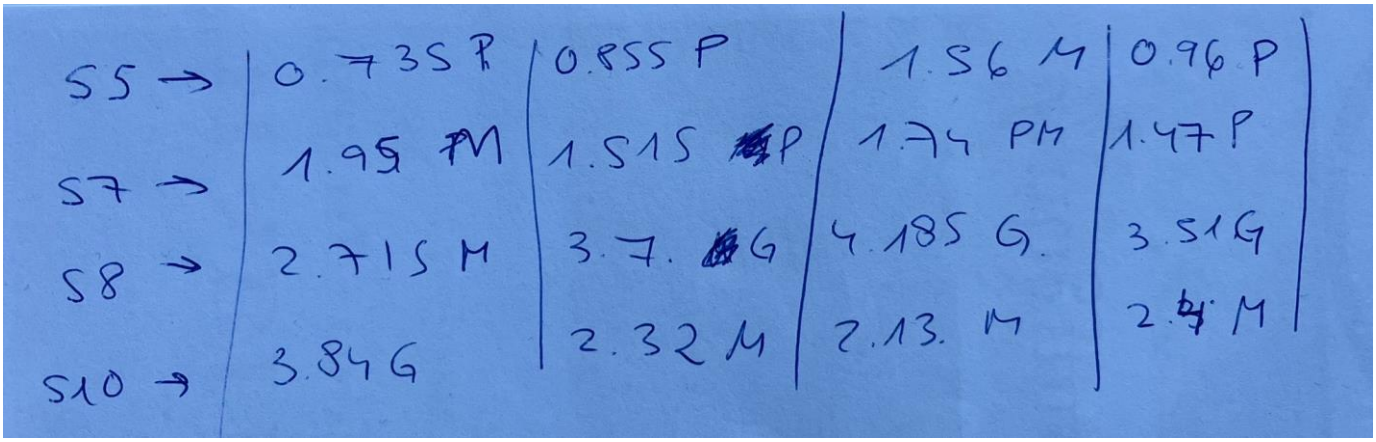
Por otro lado, se ha creado un controlador borroso llamado parking_linea_v7.fis. Este tiene 4 entradas que corresponden con los sónares previamente mencionados, los cuales tienen un rango de 0 a 5 metros, por otro lado, hay dos salidas, la primera siendo W y la segunda V (es importante este orden ya que, si se introducen inversamente, el programa no funcionará). A continuación, se explicará en detalle cada input y output:

# 1. Funciones de pertenencia:

Un aspecto resaltable es que, como se puede ver a continuación, hemos creado las funciones de tal forma que no hay lugar a dudas cuando empieza una y termina otra. De esta forma nunca estarán cumpliéndose dos a la vez (a excepción de un caso en el sonar_8). Asimismo, las funciones de los sónares cubren todo el campo de valores posible para que el programa no pueda quedarse pillado entre dos.

Las distancias que abarcan cada uno de los conjuntos las hemos definido con algunos cálculos o a ojo según las necesidades que fuéramos teniendo y, luego las hemos limado cuando en alguna ejecución alguna no actuaba como queríamos. Es decir, establecemos por ejemplo una regla en la que utilizamos sonar_5 pequeño. Ejecutamos para probar esa regla y, cuando en programa se queda parado antes de finalizar, gracias a los comandos **rostopic echo /robot0/sonar_5** medimos las distancias de cada sonar a las paredes y, así detectamos la razón de la parada antes de tiempo. Si hay alguna distancia que queremos que siga siendo, por ejemplo, pequeña para que la regla haga que avance más el coche en el simulador, entonces hacemos que la función sonar_5 pequeña abarque hasta el valor que vemos que necesitamos.

Adjuntamos imagen de las últimas pruebas que hicimos con esta técnica para que se entienda mejor:




En general, en todos los sensores hemos creído justo establecer 3 funciones, PEQUEÑO, MEDIANO y GRANDE, que, ajustándolas adecuadamente, nos sirven suficientemente paras nuestros propósitos.

Las salidas las hemos estimado de una forma distinta que será explicada a continuación.

Cada variable fis queda de la siguiente forma:

- **Sonar_5:** Este output tiene 3 funciones trapezoidales, que son PEQUEÑO, MEDIANO y GRANDE y se activarán en función a las medidas captadas por este sensor, activándose la función PEQUEÑO con unas distancias pequeñas, y así con el resto.
  Las dimensiones de estas 3 funciones se pueden ver en la imagen, abarcando, por ejemplo, el pequeño desde 0 a 1.3 metros.

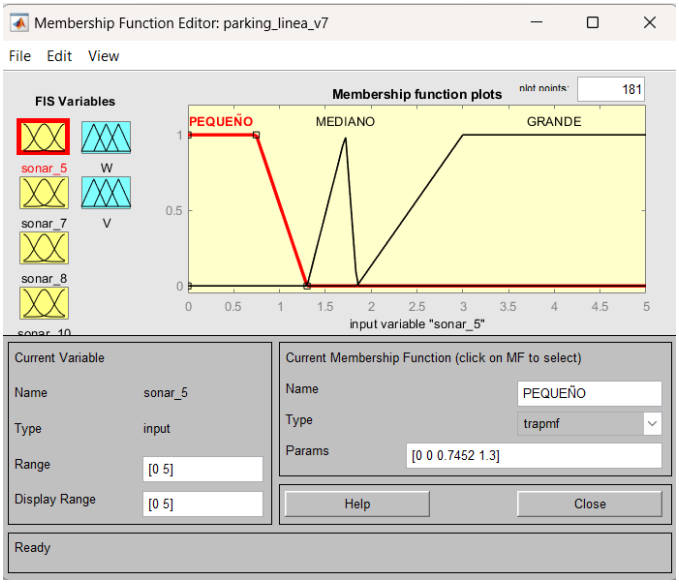



Figura 2. Sonar_5

- **Sonar_7:** Este output es igual al anterior, lo único que cambia son las distancias que abarcan los 3 conjuntos, al igual que en los siguientes.

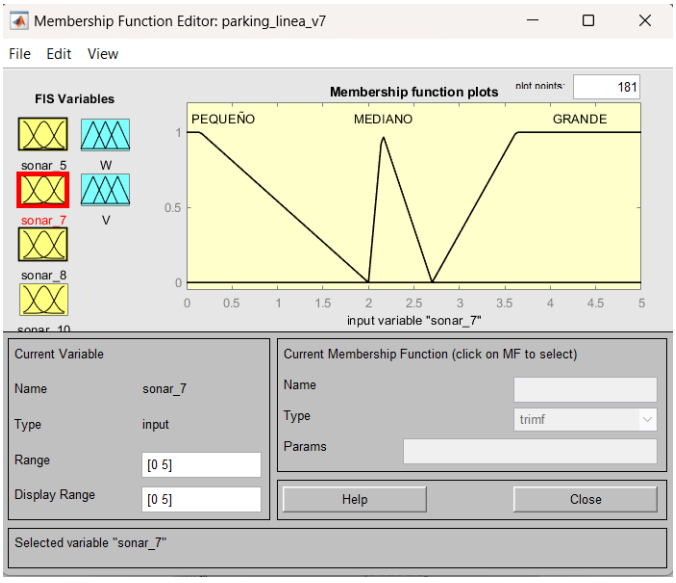



Figura 3. Sonar_7

- **Sonar_8:**

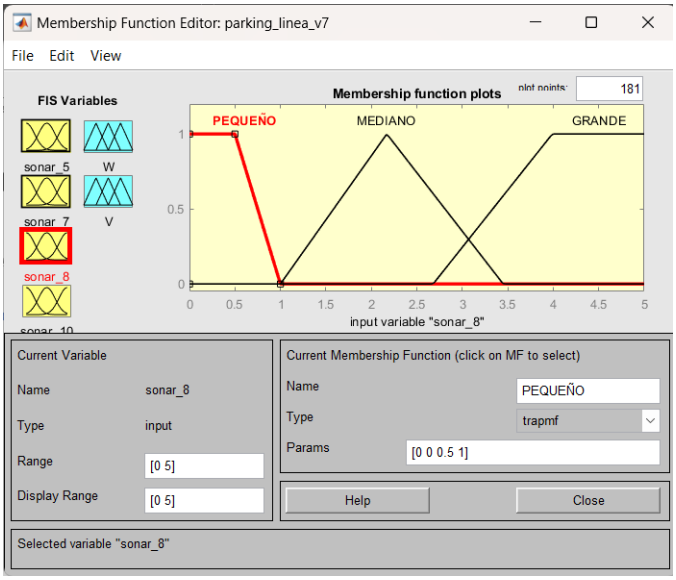



Figura 4. Sonar_8

- **Sonar_10:**

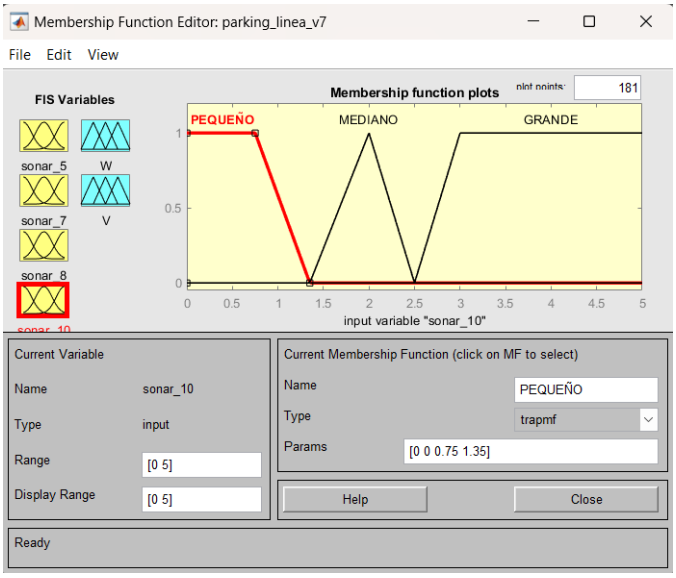



Figura 5. Sonar_10

- **W:** Este output tiene un rango de -90 a 90 grados, que corresponde con el giro del volante, siendo un giro a la izquierda cuando el ángulo es negativo, un giro a la derecha cuando el ángulo es positivo y no hay giro cuando es 0.
  Hay 5 funciones, MUY_NEG, MUY_POS (ambas trapezoidales que se usan cuando se quiere hacer un giro al máximo, contundente), NEG, POS y CERO (trapezoidales que se usan para hacer giros más suaves a la izquierda, derecha o sin girar respectivamente).
  A las funciones MUY_NEG y MUY_POS les hemos dado esta forma ya que así podíamos acentuar mucho más el ángulo de giro que con un triángulo.


Figura 6. W

- **V:** Este output tiene un rango de -30 a 30, siendo -30 la máxima velocidad marcha atrás y 30 la máxima velocidad hacia adelante. Se han creado 4 funciones ADELANTE_MANIOBRA, ATRÁS, QUIETO Y ATRÁS_MANIOBRA, las cuales sirven para mover hacia adelante o hacia atrás el robot.
  Vemos que están todas en un rango de números bastante parejo. Unos centímetros son clave para que el coche aparque bien o no lo haga y, a velocidades muy altas, cubre mucho más espacio en menos tiempo, por lo que, si el coche coge demasiada velocidad, luego tiene menor velocidad de reacción para parar y girar el volante para maniobrar hacia la plaza de parking y como consecuencia chocaba con la pared. Esto es simplemente por el hecho de que si el controlador le envía la señal y tarda pongamos 0.1 segundos en llegar, habrá avanzado más espacio.

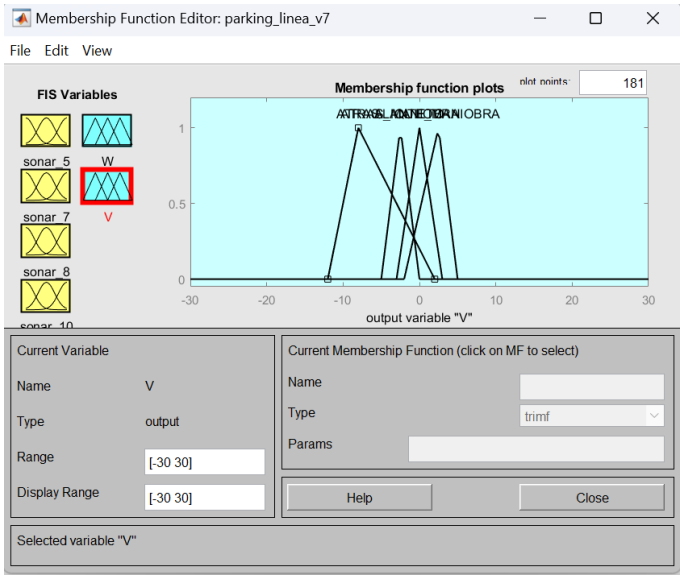
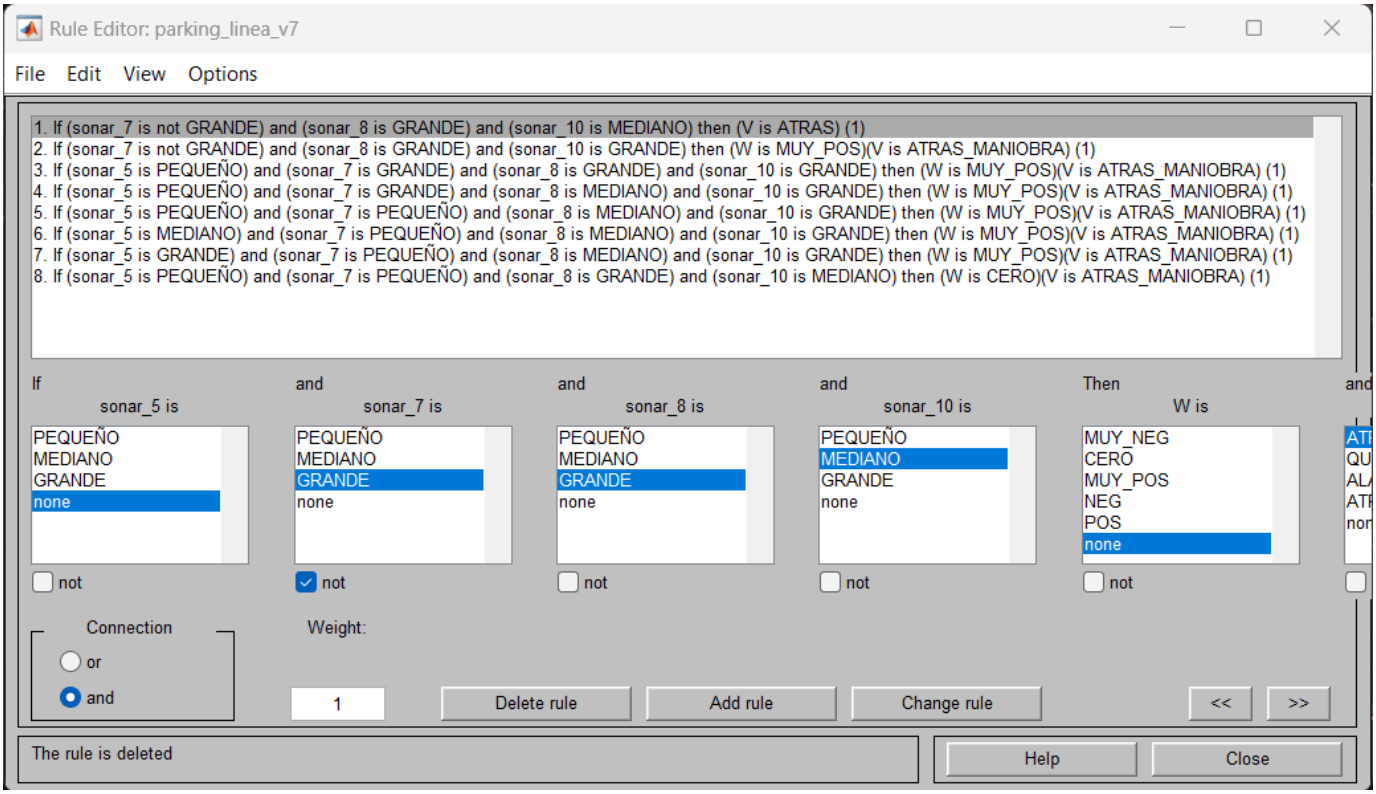
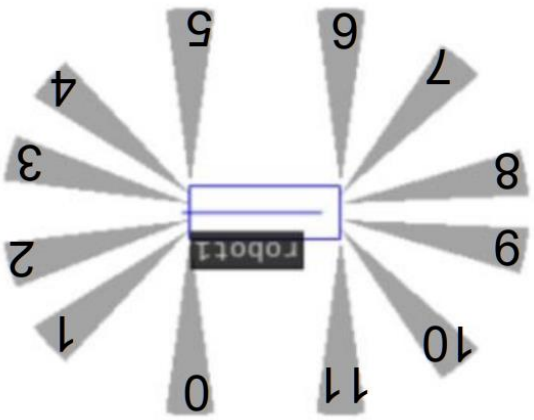


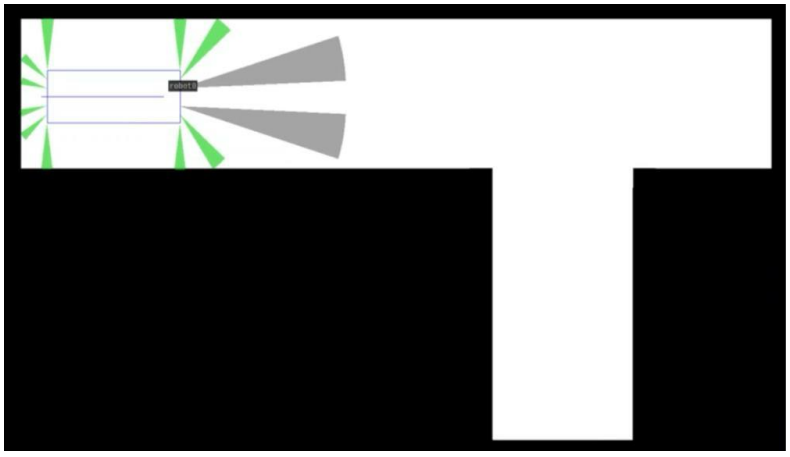

Figura 7. V

## 2. Las reglas que hemos utilizado son las siguientes:




Adjunto también foto de los sensores y del mapa para facilitar la comprensión de estas reglas:



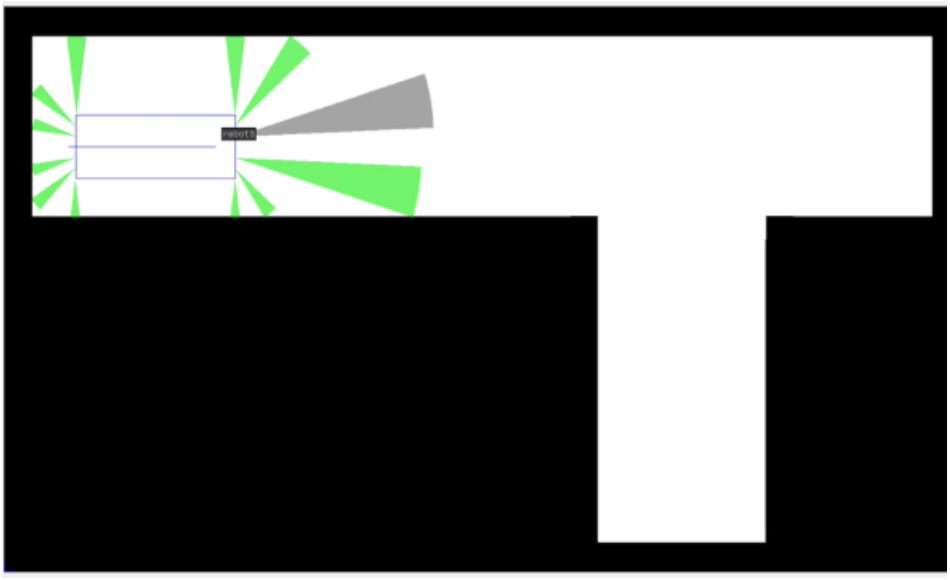



Al estar el coche al revés, adjuntamos la imagen volteada para que la situación de los sensores se corresponda con la realidad.

## Regla 1:

1. If (sonar_7 is not GRANDE) and (sonar_8 is GRANDE) and (sonar_10 is MEDIANO) then (V is ATRAS) (1)

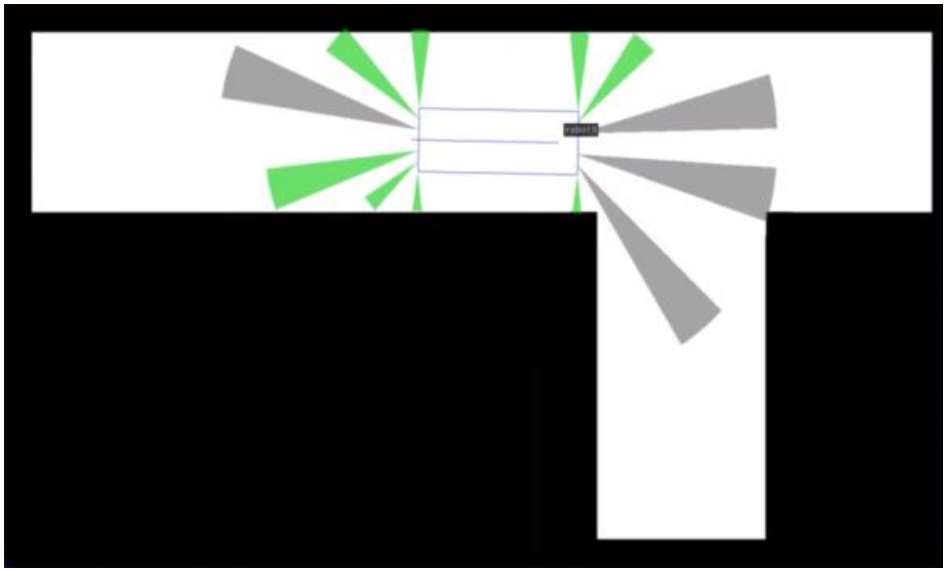
Sirve para el arranque inicial en el que el coche va marcha atrás en línea recta.



## Regla 2:

2. If (sonar_7 is not GRANDE) and (sonar_8 is GRANDE) and (sonar_10 is GRANDE) then (W is MUY_POS)(V is ATRAS_MANIOBRA) (1)

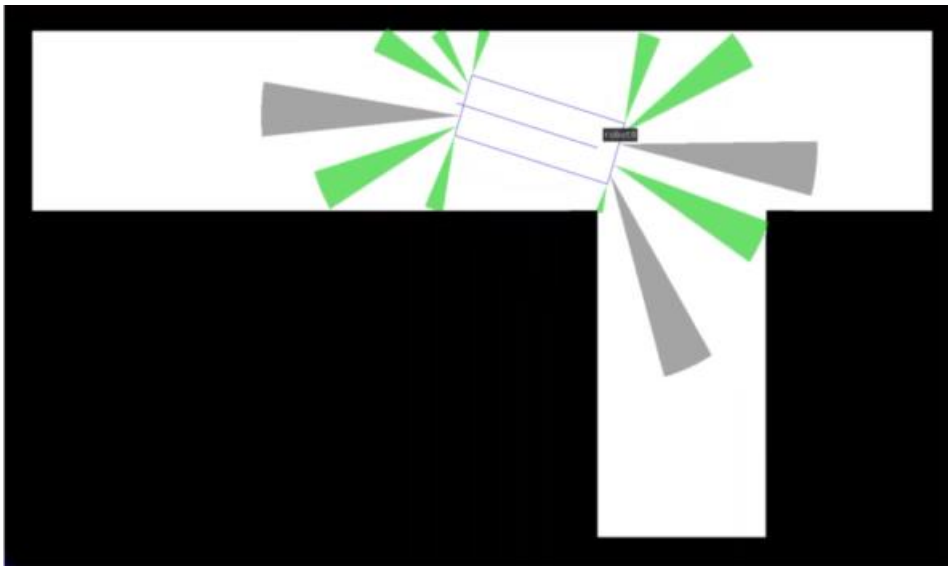
Cuando el sonar_10 detecta hueco, giramos el volante a tope y reducimos la velocidad a la adecuada para maniobrar.



## Regla 3:

3. If (sonar_5 is PEQUEÑO) and (sonar_7 is GRANDE) and (sonar_8 is GRANDE) and (sonar_10 is GRANDE) then (W is MUY_POS)(V is ATRAS_MANIOBRA) (1)

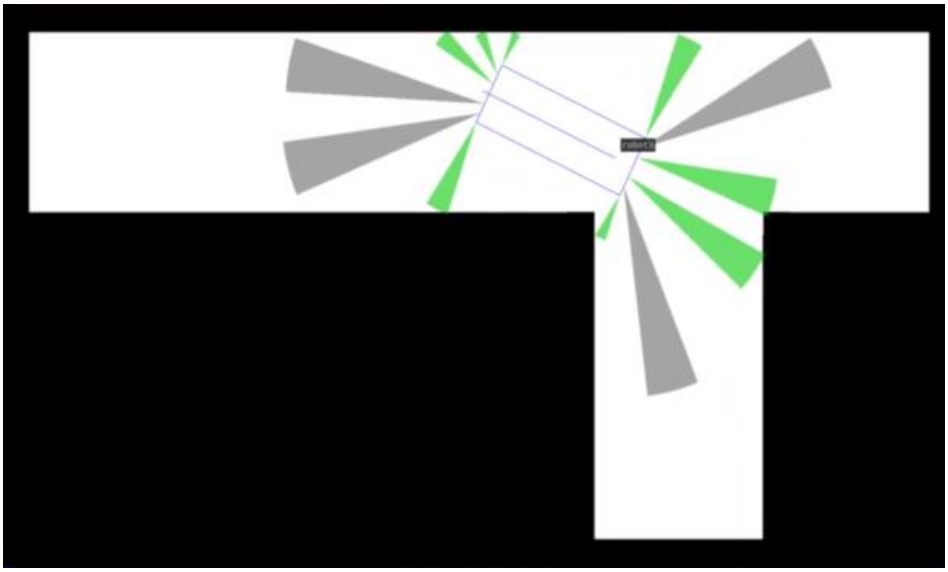
Igual que la anterior, pero en este caso debido a que ya ha girado sustancialmente, la distancia del sónar_7 cambia a grande, por lo que hay que contemplarlo para que siga girando.



## Regla 4:

4. If (sonar_5 is PEQUEÑO) and (sonar_7 is GRANDE) and (sonar_8 is MEDIANO) and (sonar_10 is GRANDE) then (W is MUY_POS)(V is ATRAS_MANIOBRA) (1)

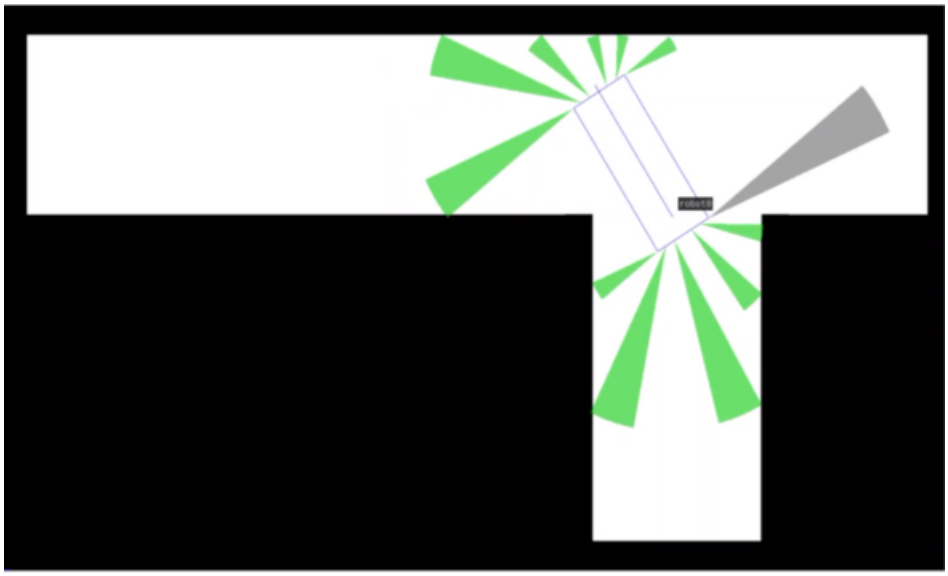
Idéntico caso, pero, en este caso el que cambia es el valor de sonar_8, que comienza a detectar pared.



### Regla 5:

5. If (sonar_5 is PEQUEÑO) and (sonar_7 is PEQUEÑO) and (sonar_8 is MEDIANO) and (sonar_10 is GRANDE) then (W is MUY_POS)(V is ATRAS_MANIOBRA) (1)

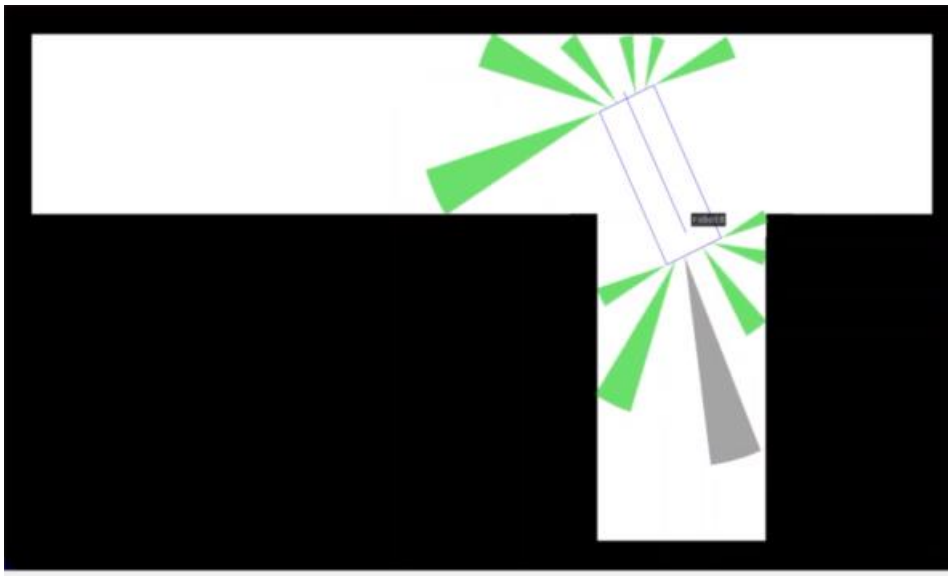
Ya ha comenzado a entrar un poco en el hueco y el sonar_7 se acerca bastante a la pared. Queremos que siga la maniobra de entrada a la plaza de parking, por lo que el ángulo del volante sigue igual y la velocidad de maniobra también.



### Regla 6:

6. If (sonar_5 is MEDIANO) and (sonar_7 is PEQUEÑO) and (sonar_8 is MEDIANO) and (sonar_10 is GRANDE) then (W is MUY_POS)(V is ATRAS_MANIOBRA) (1)

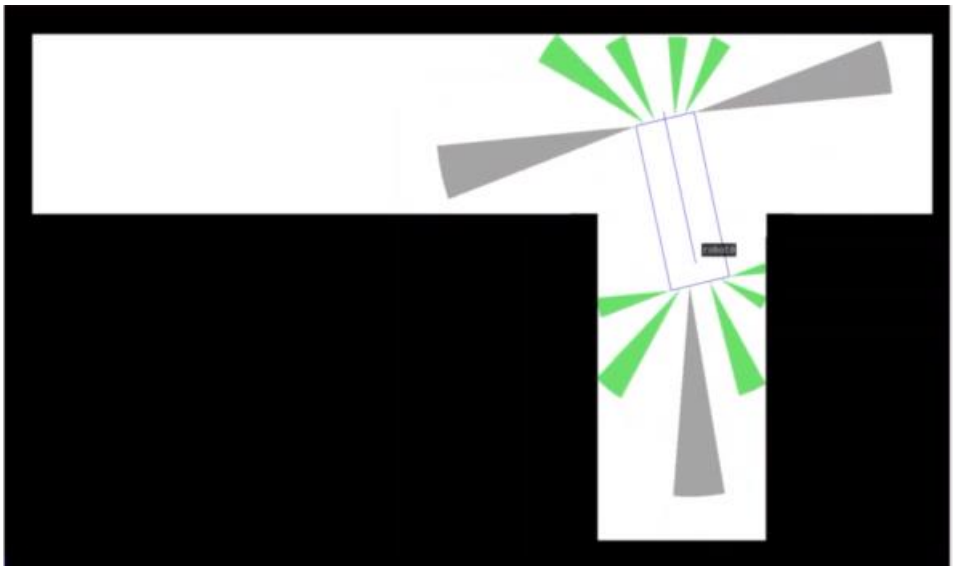
Sonar_5 empieza a estar lejos de la pared de arriba y el 7 cerca de la pared del parking.



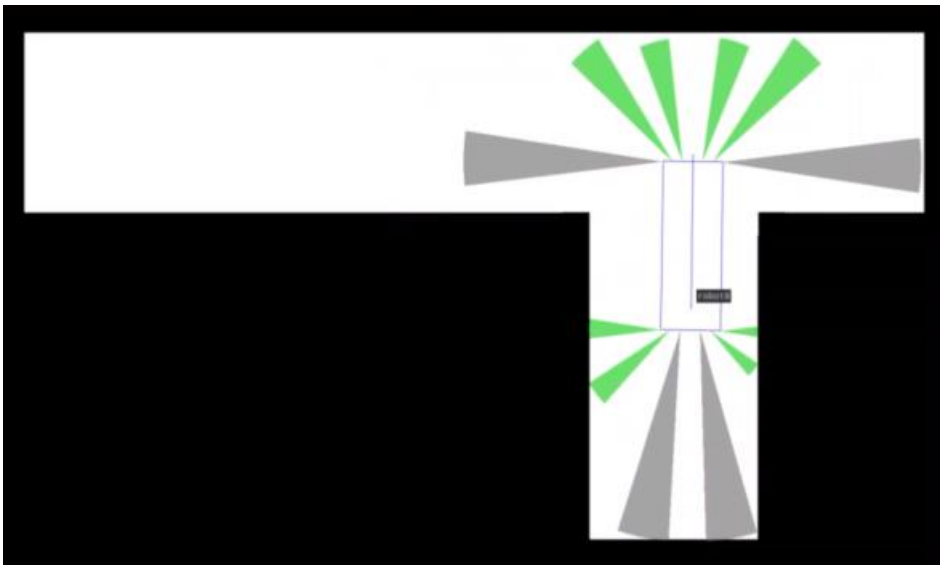
## Regla 7:

7. If (sonar_5 is GRANDE) and (sonar_7 is PEQUEÑO) and (sonar_8 is MEDIANO) and (sonar_10 is GRANDE) then (W is MUY_POS)(V is ATRAS_MANIOBRA) (1)

Sonar_5 se ha despegado la de la pared de arriba y lo que detecta es el hueco de la derecha.



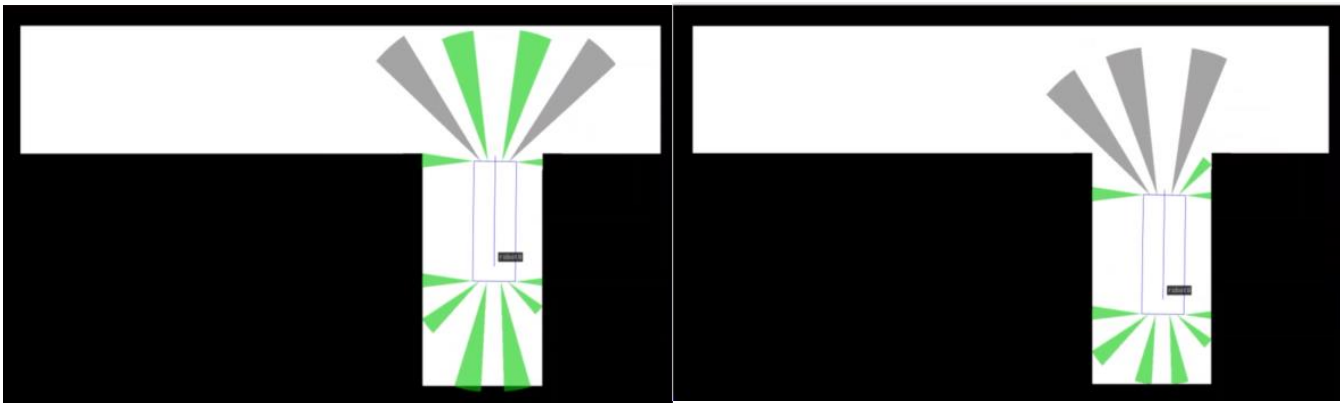
En este momento, vuelve a entrar en juego la regla 1 cuando el coche termina de ponerse recto. El coche avanza hacia atrás, con el ángulo del volante en 0.



## Regla 8:

8. If (sonar_5 is PEQUEÑO) and (sonar_7 is PEQUEÑO) and (sonar_8 is GRANDE) and (sonar_10 is MEDIANO) then (W is CERO)(V is ATRAS_MANIOBRA) (1)

El coche ya ha entrado en la plaza y, por tanto, los dos sónares de la derecha detectan una distancia pequeña y el coche se seguirá moviendo, en línea recta esta vez, mientras que el sonar_8 detecte una distancia grande. Cuando esto deja de ocurrir, al no haber más reglas, el coche se para y estará ya aparcado.



**Condición de parada:**

Se ha creado una condición de parada que es la encargada de terminar la ejecución del programa. Hemos diseñado el sistema de reglas de tal manera que el robot cuando aparca se queda ya parado. Consiguientemente, la función de parada que hemos utilizado es, que detecte que el robot está quieto y, que hayan pasado más de 17 segundos (tarda alrededor de 16 en aparcar). El código es el siguiente:

```
function [flag_parada, filter_speed] = condicion_parada(raw_speed, timmer)

    % disp(timmer)
    if raw_speed <0.1 && timmer > 17

        % 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
end
```

## Parte 2. Diseño automático de un controlador neuronal

### 1. Obtener los datos de entrenamiento.

Hemos decidido usar la opción b): Usar un recorrido prefijado, para ello hemos modificado el script "maniobra_park_ackerman_datos_entrenamiento_alumnos.m". Con este script, hemos realizado distintos aparcamientos, modificando en cada uno de ellos algunos parámetros de los avances 1,2 y 3 para que aparcase de manera distinta. Guardando los datos de cada uno de estos aparcamientos ligeramente distintos en "datos_entrenamientoX training_data", donde la X corresponde al número de cada conjunto y training_data es la tabla donde se archivan todos los valores captados durante la ejecución. En esta tabla, las columnas de la 1 a la 12 corresponden a los sensores y las columnas 18 y 19 con la velocidad angular y lineal respectivamente.

A continuación, se muestra el script creado:

```
%% conectar
%**********************

rosshutdown
clear all
close all

rosinit('http://192.168.5.171:11311', 'Nodehost', '192.168.5.129');

global steering_wheel_angle;
global vel_lineal_ackerman_kmh;

%% ini_simulador_ACKERMAN
%**********************

%DECLARACIÓN DE SUBSCRIBERS
%%%%%%%%%%%%%%%%%%%%%%%%%%
%Odometria
sub_odom=rossubscriber('/robot0/odom');

%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=8.4

vel_lineal_ackerman_kmh = -5     %(km/h)
steering_wheel_angle = 0       % desde -90 a 90 grados.
avanzar_ackerman


% AVANCE 2

distancia=5.4

vel_lineal_ackerman_kmh = -3.8     %(km/h)
steering_wheel_angle = 85       % desde -90 a 90 grados.
avanzar_ackerman

% AVANCE 3

distancia=3.9

vel_lineal_ackerman_kmh = -2.5     %(km/h)
steering_wheel_angle = 0      % desde -90 a 90 grados.
avanzar_ackerman


save datos_entrenamiento4 training_data
```

Con este script hemos creado hasta 7 datos de entrenamiento, los cuales usaremos para obtener los mejores resultados posibles.

## 2. Preparar los datos de entrenamiento/training.

Una vez obtenido los datos de entrenamiento, se necesita crear la red neuronal, para ello hemos pensado que los sensores más necesarios para el aparcamiento son los sonares 5, 6, 7, 8 y 10. Con estos se puede obtener la información necesaria para poder aparcar. Por otra parte, mediante la experimentación, hemos llegado a la conclusión que, con 16 neuronas en 3 capas, 10 en la primera y 3 en la segunda y la tercera, se obtienen los mejores resultados. Para ello hemos creado un script, llamado "GeneradorRed.m" que, al ejecutarse creará la red neuronal. El script contiene el siguiente código:

```
training_data = load("datos_entrenamiento6").training_data;

%Generar los inputs y los outputs
inputs = training_data(:,[6,7,8,9,11]);
outputs = training_data(:,[18,19]);
inputs(isinf(inputs)) = 5.0;
inputs = double(inputs');
outputs = double(outputs');

%Generación de la red
net = feedforwardnet([10,3,3]);
net = configure(net,inputs,outputs);
net = train(net,inputs,outputs);

%Formación de la red
gensim(net)
```

Tras hacer diversas pruebas el conjunto de entrenamiento que mejores resultados nos ha dado es el conjunto "datos_etrenamiento6", que se ha entrenado con el robot en la posición definitiva y realiza 3 movimientos para aparcar.

Al terminar de ejecutarse este script nos dará la red neuronal que implementaremos en un nuevo modelo de simulink llamado "RNackerman_ROS_controller". En este simulink se sustituye el controlador por la red neuronal para poder realizar el aparcamiento autónomo y debemos aumentar en 1 el número de entradas del demultiplexor. A continuación, se muestra una imagen de cómo se debe conectar:

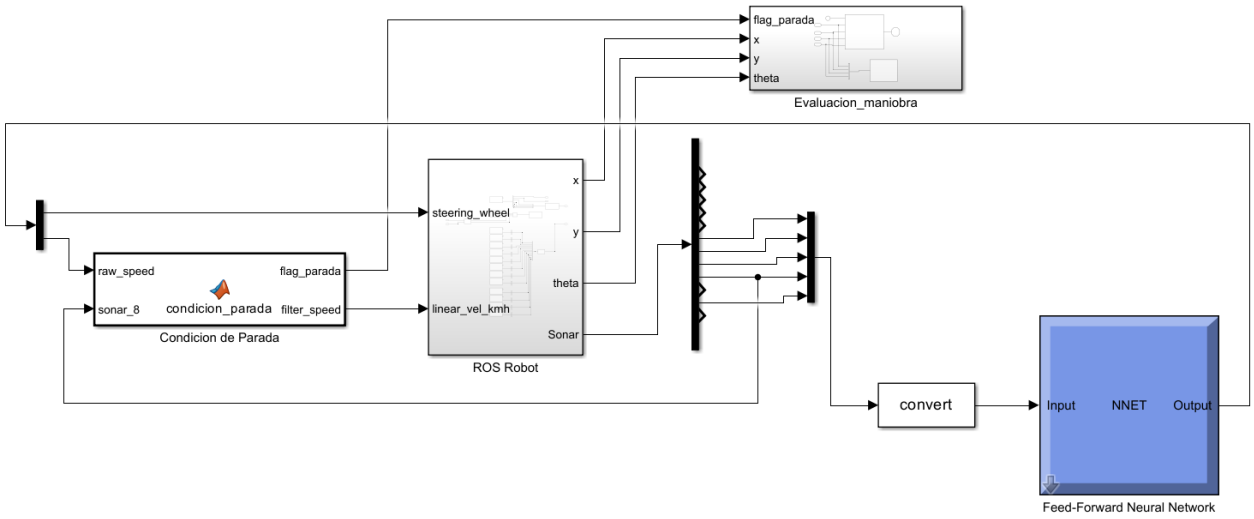
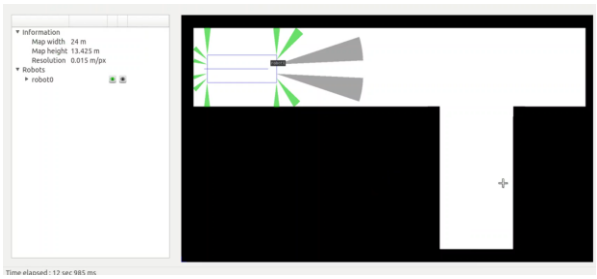


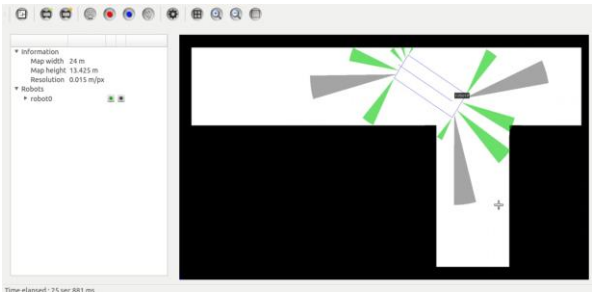

Figura 8. RNackerman_ROS_controller.smlk

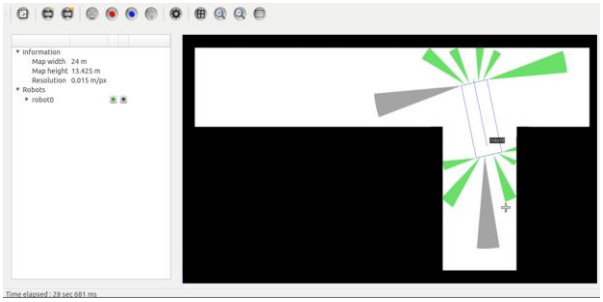
A diferencia de la parte 1 anterior (el sistema simulink con controlador borroso), se ha cambiado la condición de parada para que se ajuste más a este modelo. La nueva condición de parada es la orden de detención cuando la distancia del sonar_8 con la pared sea menor de 1 metro, que sólo se da cuando ha aparcado y el coche está ya muy cerca de la pared.

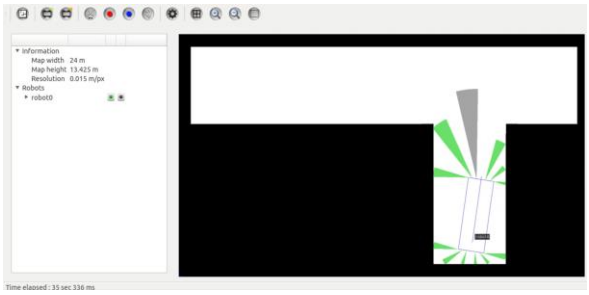
```
function [flag_parada, filter_speed] = condicion_parada(raw_speed, sonar_8)

    % Condición de parada:
    % - Velocidad igual a 0 y tiempo transcurrido mayor a 1 segundo

    if sonar_8 < 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
end
```

A continuación, se muestra un ejemplo de cómo aparcaría el robot en el escenario inicial:



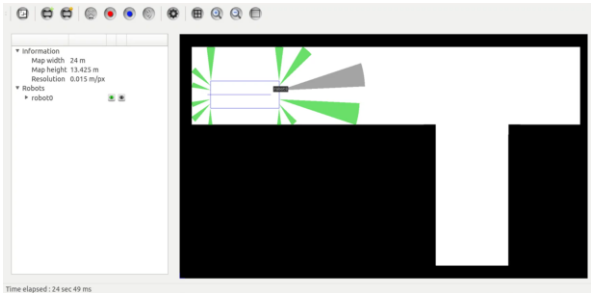


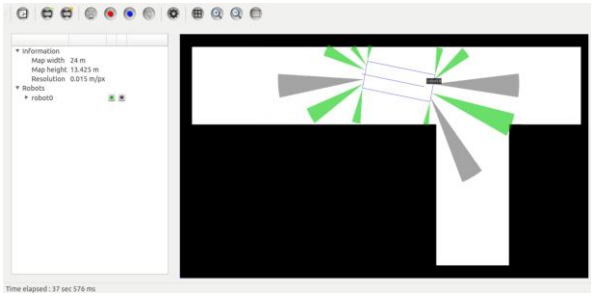


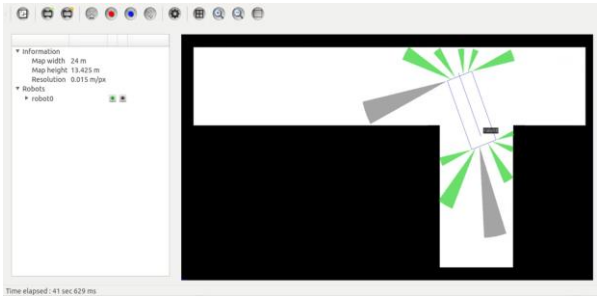


Como podemos observar, obtenemos un resultado exitoso al ser capaz de aparcar por sí mismo. Por otro lado, se ha realizado la misma prueba con el entorno definitivo, el cual es el mismo mapa, pero con la posición inicial del robot ligeramente más abajo.

A continuación, se muestra el recorrido que forma el robot:







Se puede observar que en ambos entornos el robot se comporta de manera similar, obteniendo así resultados exitosos en ambos casos, siendo capaz de aparcar tanto en el entorno original y el definitivo.