Sistemas de Control Inteligente

Práctica Final.

Jorge Revenga Martín de Vidales
Ángel Salgado Aldao

Grado en Ingeniería Informática
Universidad de Alcalá

14 de enero de 2024

Índice

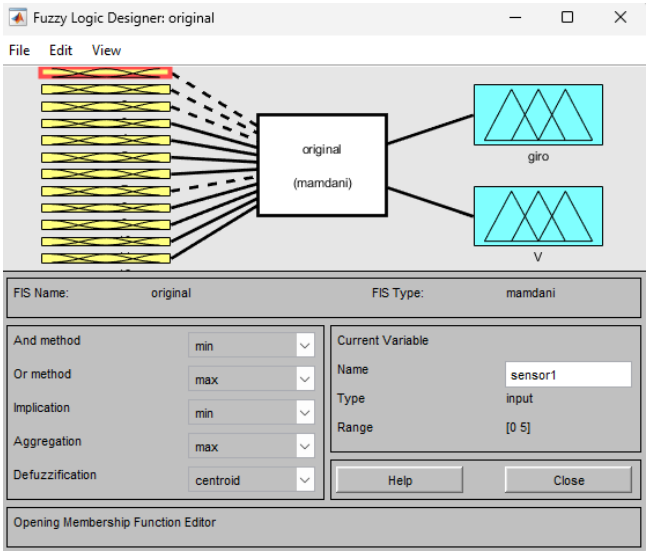
Parte I
Diseño manual de un control borroso de tipo MAMDANI.

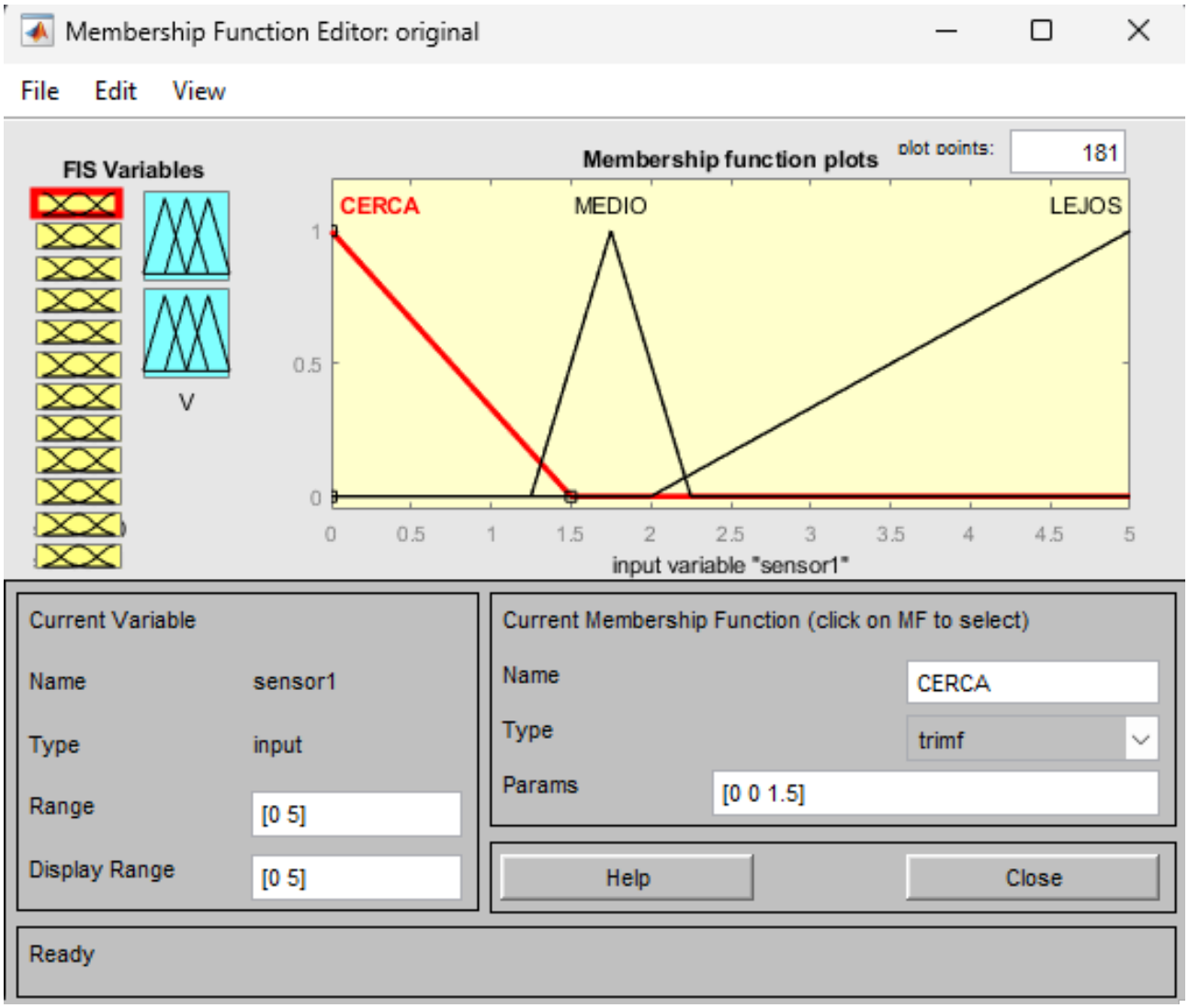
En el diseño del controlador borroso para la maniobra de estacionamiento, se empleó la herramienta Fuzzy de MATLAB, utilizando un sistema de inferencia Mamdani. El objetivo principal fue regular la velocidad lineal y el giro del volante del vehículo, basándose en las lecturas de sus 12 sensores de ultrasonidos.

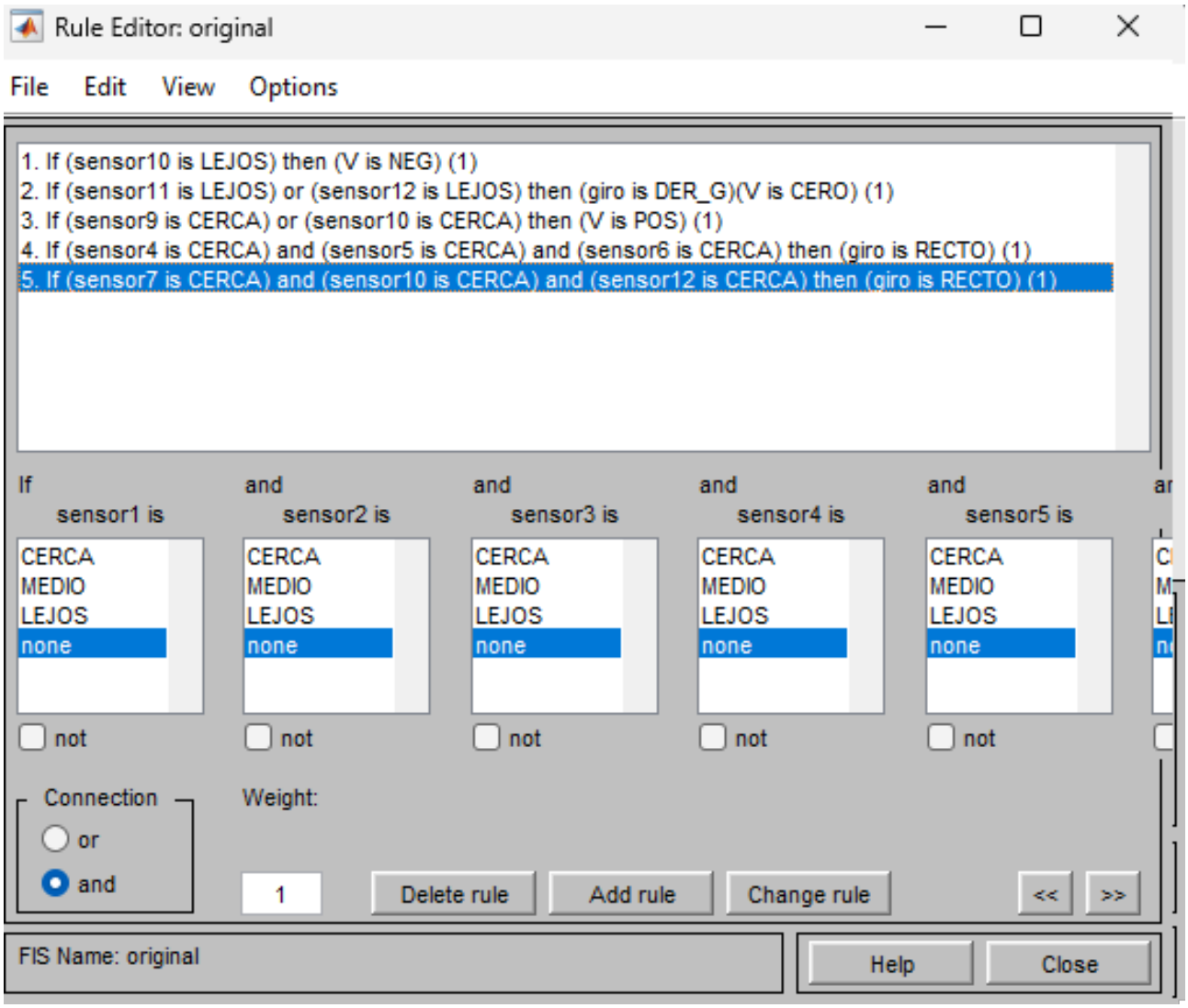
Para cada sensor, se definieron tres funciones de membresía (CERCA, MEDIO, LEJOS) con rangos de entrada de [0, 5]. Estas funciones capturan la información borrosa proveniente de los sensores y facilitan la toma de decisiones en el sistema borroso. La selección de tres funciones de membresía por sensor permite modelar diferentes grados de proximidad de manera más precisa.

En cuanto a las salidas del controlador, se definieron dos variables: giro y velocidad. Para la variable de giro, se establecieron cinco funciones de membresía (IZQ_G, IZQ, RECTO, DER, DER_G) con un rango de salida de [-90, 90]. Mientras tanto, la variable de velocidad tiene tres funciones de membresía (NEG, CERO, POS) con un rango de salida de [-6, 6].

Se implementaron cinco reglas borrosas en el sistema de inferencia, las cuales relacionan las condiciones borrosas de los sensores con las acciones de control. Cada regla especifica el comportamiento del vehículo en función de la distancia detectada por los sensores. Por ejemplo, si los sensores 10, 11 y 12 indican que el obstáculo está LEJOS, la regla 1 establece que el vehículo debe girar a la IZQUIERDA_GRANDE y retroceder con una velocidad NEGATIVA.




Membership Function Editor: original
File
Edit
View
FIS Variables
V
Membership function plots
plot points:
181
CERCA
MEDIO
LEJOS
1
0.5
0
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
input variable "sensor1"
Current Variable
Name
sensor1
Type
input
Range
[0 5]
Display Range
[0 5]
Current Membership Function (click on MF to select)
Name
CERCA
Type
trimf
Params
[0 0 1.5]
Help
Close
Ready



En resumen, el diseño del controlador borroso se basa en la definición cuidadosa de funciones de membresía, reglas y métodos de inferencia. A través de la lógica difusa, el controlador interpreta eficazmente las lecturas borrosas de los sensores y genera comandos precisos de velocidad y giro para el vehículo, optimizando su desempeño en la maniobra de estacionamiento. El ajuste y refinamiento de estos componentes son cruciales para adaptar el controlador a diferentes situaciones y mejorar su rendimiento en la tarea específica planteada.

Parte II
Diseño automático de un controlador neuronal.

1. Obtener los datos de entrenamiento

Para esta parte de la práctica en primera instancia se intentó recopilar los datos a partir del control manual, pero tras comprobar la falta de consistencia y la complejidad de hacerlo de este modo se decidió pasar a la siguiente opción proporcionada en el enunciado, 'b) Realizar un recorrido prefijado '

El código para la generación de los datos de entrenamiento está hecho a partir del proporcionado en el aula virtual de la asignatura, cambiando tras el mensaje de inicialización y la inicialización de la matriz training_data. El primer cambio que aparece es un bucle que verifica la llegada de los mensajes de los sensores

```
% Verificación de disponibilidad de mensajes de todos los sensores antes de avanzar
while isempty(sonar_0.LatestMessage) || isempty(sonar_1.LatestMessage) || isempty(sonar_2.LatestMessage) ||
↪  ...
      isempty(sonar_3.LatestMessage) || isempty(sonar_4.LatestMessage) || isempty(sonar_5.LatestMessage) ||
      ↪  ...
      isempty(sonar_6.LatestMessage) || isempty(sonar_7.LatestMessage) || isempty(sonar_8.LatestMessage) ||
      ↪  ...
      isempty(sonar_9.LatestMessage) || isempty(sonar_10.LatestMessage) || isempty(sonar_11.LatestMessage)
    % Espera hasta que todos los mensajes de sonar sean recibidos
    waitfor(r);
end
```

Para entrenar la red de forma que pueda aparcar desde distintas posiciones de inicio, se han almacenado datos de diferentes aparcamientos exitosos con posiciones iniciales distintas. La siguiente parte del código son los valores de cada avance realizado por el vehículo en cada simulación. La elección de qué mapa utilizar está determinada por qué secciones se comentan o descomentan en el código.

```
% Mapa 1
% distancias = [0.5, 1.5, 2, 1.5, 3.2, 2.6, 1, 3.5]; velocidades = [-15, -15, -15, -15, -15, -10, -10, -15];
↪  angulos = [0, 70, 0, -70, 0, 90, 30, 0];
% Mapa 2
% distancias = [0.5, 1.5, 2, 1.5, 2.8, 2.6, 1, 3.5]; velocidades = [-15, -15, -15, -15, -15, -10, -10, -15];
↪  angulos = [0, 10, 0, -10, 0, 90, 30, 0];
% Mapa 3
% distancias = [0.5, 1.5, 2, 1.5, 3.5, 2.6, 1, 3.5]; velocidades = [-15, -15, -15, -15, -15, -10, -10, -15];
↪  angulos = [0, 60, 0, -60, 0, 90, 30, 0];
% Mapa 4
%distancias = [0.5, 1.5, 2, 1.5, 3.2, 2.6, 1, 3.5]; velocidades = [-15, -15, -15, -15, -15, -10, -10, -15];
↪  angulos = [0, 70, 0, -70, 0, 90, 30, 0];
% Mapa 5
%distancias = [0.5, 1.5, 2, 1.5, 2.8, 2.6, 1, 3.5]; velocidades = [-15, -15, -15, -15, -15, -10, -10, -15];
↪  angulos = [0, 30, 0, -30, 0, 90, 30, 0];
% Mapa 6
% distancias = [0.5, 1.5, 2, 1.5, 3.32, 2.6, 1, 3.5]; velocidades = [-15, -15, -15, -15, -15, -10, -10, -15];
↪  angulos = [0, 60, 0, -60, 0, 90, 30, 0];
% Mapa 7
% distancias = [0.5, 1.5, 2, 1.5, 2.8, 2.6, 1, 3.5]; velocidades = [-15, -15, -15, -15, -15, -10, -10, -15];
↪  angulos = [0, 30, 0, -30, 0, 90, 30, 0];
% Mapa 8
```

```
% distancias = [0.5, 2, 2, 1.5, 3.2, 2.6, 1, 3.5]; velocidades = [-15, -15, -15, -15, -15, -10, -10, -15];
↪ angulos = [0, 70, 0, -75, 0, 90, 30, 0];
% Mapa 9
% distancias = [0.5, 1.5, 2, 1.5, 2.8, 2.6, 1, 3.5]; velocidades = [-15, -15, -15, -15, -15, -10, -10, -15];
↪ angulos = [0, 10, 0, -10, 0, 90, 30, 0];
% Mapa 10
%distancias = [0.5, 1.5, 2, 1.5, 2.925, 2.6, 1, 3.5]; velocidades = [-15, -15, -15, -15, -15, -10, -10, -15];
↪ angulos = [0, 50, 0, -50, 0, 90, 30, 0];
% Mapa original
% distancias = [0.5, 1.5, 2, 1.5, 3.5, 2.6, 1, 3.5]; velocidades = [-15, -15, -15, -15, -15, -10, -10, -15];
↪ angulos = [0, 60, 0, -60, 0, 90, 30, 0];
% Mapa definitivo
 distancias = [0.5, 1.5, 2, 1.5, 3.15, 2.6, 1, 3.5]; velocidades = [-15, -15, -15, -15, -15, -10, -10, -15];
 ↪ angulos = [0, 10, 0, -10, 0, 90, 30, 0];

```

Estos valores se aplican a la simulación igual que en el código proporcionado a los alumnos, con la diferencia de que se realiza en un bucle para jecutar los avances para cada conjunto de distancias, velocidades y ángulos en el mapa seleccionadoguardar los datos de todos los avances.

```
% Realizar avances
for i = 1:length(distancias)
    distancia = distancias(i);
    vel_lineal_ackerman_kmh = velocidades(i);
    steering_wheel_angle = angulos(i);
    avanzar_ackerman;
end
```

Tras la simulación se ha añadido una comprobación del éxito del aparcamiento, la cual se realiza manualmente, introduciendo el resultado con el teclado (una 's 'almacena los datos al final de la matriz training_data.

```
% Guardar datos de entrenamiento solo si el usuario desea
respuesta = input('¿Desea guardar la simulación? (Sí: s / No: n): ', 's');

if strcmpi(respuesta, 's')
    save datos_training_definitivo training_data;
    disp('Simulación guardada.');
else
    disp('Simulación no guardada.');
    % Puedes agregar aquí el código para realizar acciones adicionales si la simulación no se guarda
end
```

Con este código se almacena en el archivo "datos_training_definitivo"la matriz training_data con los datos recogidos de los 12 sónares, las velocidades y los ángulos del volante

2. Generar la red con la configuración deseada y preparar los datos de entrenamiento.

En el archivo 'control_neuronal.m ', se establece la configuración de la red neuronal encargada de controlar el vehículo. Para lograr una red con la capacidad óptima para ejecutar la maniobra deseada, se procede a entrenar la red en un ciclo específico solo si su rendimiento ha caído por debajo de un umbral predefinido.
En primer lugar, se inicializan las variables que determinan el número de ciclos y el umbral de rendimiento del entrenamiento, junto con una red neuronal inicial que cuenta con tres capas ocultas, cada una compuesta por 6, 7 y 6 neuronas, respectivamente.

```
% Número de ciclos de entrenamiento
num_epochs = 4;
max_performance_threshold = 50;  % Ajusta este umbral según tus necesidades

% Inicializar la red neuronal
net = feedforwardnet([6, 7, 6]);
```

Después comienza el bucle del entrenamiento, que se ejecuta una vez por cada ciclco de entrenamiento. Al principio del bucle, se desordenan las filas de la matriz training_data con la que se entrena la red. Esto ayuda a mejorar la generalización del modelo al evitar que la red memorice el orden de los datos.

```
for epoch = 1:num_epochs
    % Desordenar las filas de los datos de entrenamiento
    num_samples = size(training_data, 1);
    permuted_index = randperm(num_samples);
    training_data_permuted = training_data(permuted_index, :);
```

Entonces se crea una variable a partir de training_data permutado con los datos que se van a usar, en nuestro caso todos los registros de todos los sónares como entradas y la velocidad y ángulo del volante como salidas. Los valores registrados como infinito se cambian a 5.0 en la línea 20 del código para que no afecten al entrenamiento.

```
    % Crear una variable 'inputs' que tome las columnas 1 a 12
    inputs = training_data_permuted(:, 1:12)';

    % Crear una variable 'outputs' que tome las columnas 18 y 19
    outputs = training_data_permuted(:, [18, 19])';

    inputs(isinf(inputs)) = 5.0;
    inputs = double(inputs);
    outputs = double(outputs);
```

La red a entrenar será una copia de la inicializada anteriormente, la cual la sustituirá en caso de cumplir con el umbral de entrenamiento. Se configura y entrena la red con los parámetros definidos anteriormente, se muestra por pantalla el rendimiento y, en caso de ser menor que el determinado al principio del código, se prosigue con el entrenamiento de la red. En caso contrario se descarta y se reinicia el bucle.

```
    while true
        % Configurar y entrenar la red temporal
        temp_net = configure(temp_net, inputs, outputs);
        temp_net = train(temp_net, inputs, outputs);

        % Evaluar el rendimiento en datos de prueba
        outputs_pred = temp_net(inputs);
        performance = perform(temp_net, outputs, outputs_pred);

        disp(['Rendimiento en datos de prueba (Epoch ', num2str(epoch), '): ', num2str(performance)]);

        % Condición para descartar el entrenamiento si el rendimiento es mayor a 80
        if performance > max_performance_threshold
            disp('Entrenamiento descartado debido a alto rendimiento.');
        else
            % Si el rendimiento es satisfactorio, salir del bucle
            net = temp_net;
            break;
        end
    end
end
```

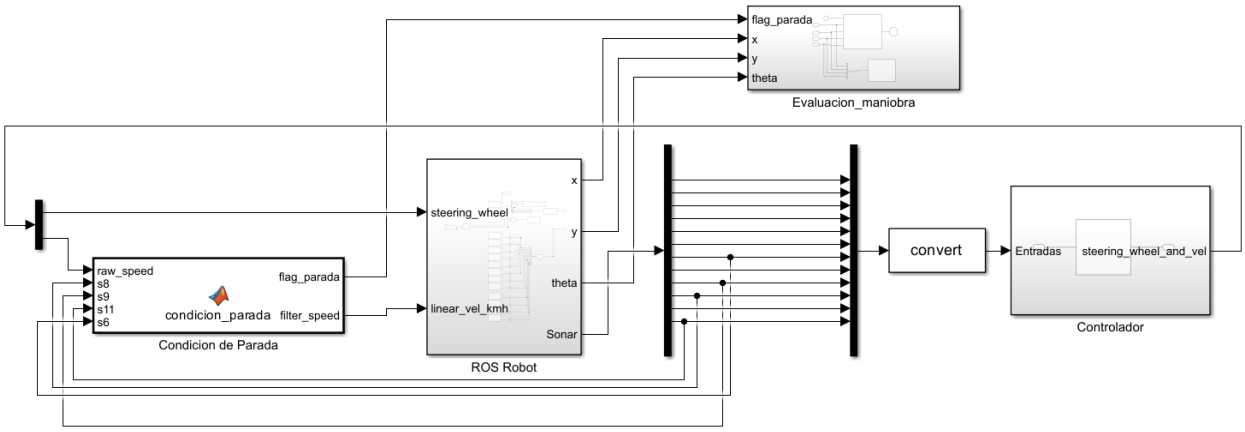
3. Generar el bloque de simulink

Al terminar de entrenarse la red se genera el bloque de simulink que controlará el vehículo en la simulación.

```
50  Ts = 100e-3;
51  gensim(net, Ts);
```

4. Modificar el diagrama de simulink

El diagrama de simulink que contiene la red neuronal está guardado con el nombre 'ackerman_ROS_controller_neuronal.slx'. En este diagrama se ha cambiado el controlador por el generado en el paso anterior y añadido el bloque 'Data Type Conversion'entre la salida del multiplexor de los ultrasonidos y la entrada de la red.



4.1. Modificar la condición de parada

Dado que la red neuronal no regula la distancia a la pared como el control borroso, se ha modificado la condición de parada para obtener una posición más cercana a la ideal según el enunciado. Para esto se han conectado los datos de los sónar traseros y laterales (de la parte de atrás) y ajustado la condición de parada en función de estos.