Sistemas de Control Inteligente

Aparcamiento Inteligente

Pablo García García
Valeria Villamares Félix

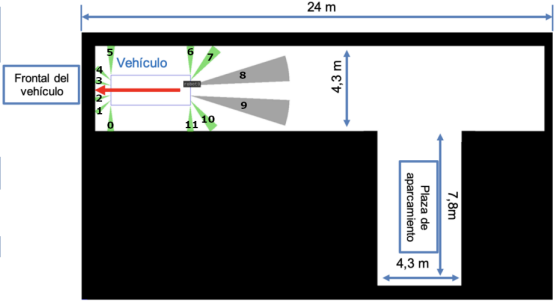
Universidad de Alcalá

14 de enero de 2024

Índice

Definición del problema

Se cuenta con un carril de $24 \times 4{,}3$ metros, y ligeramente hacia la mitad de este, en perpendicular, se encuentra una plaza de aparcamiento de $7{,}8 \times 4{,}3$ metros



Deberá realizarse un aparcamiento en batería mediante técnicas de **Lógica Borrosa** y **Redes Neuronales**.

Condición de parada

Cuando el coche se encuentre en la zona de aparcamiento deberá **frenar** mediante la ayuda de un bloque de Simulink.

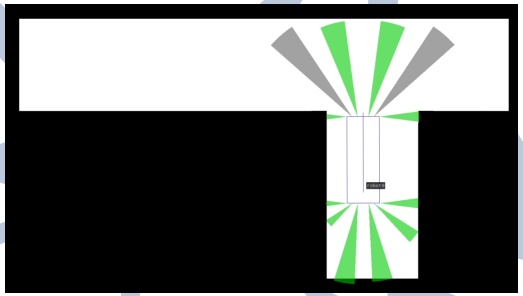


Figura: Posición de parada

Figura: Bloque de condición de parada

Condición de parada

Código

```
1   function [flag_parada, filter_speed] = condicion_parada(raw_speed,
        sensor8, sensor9)
2
3    condicionS8 = sensor8 < 0.6;
4    condicionS9 = sensor9 < 0.6;
5
6    if condicionS8 && condicionS9
7     filter_speed = 0.0;
8     flag_parada = 1;
9    else
10    filter_speed = raw_speed;
11    flag_parada = 0;
12
13  end
```

Controlador borroso

Variables

El controlador trabaja con las **variables**:

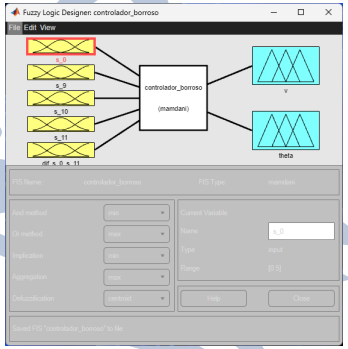


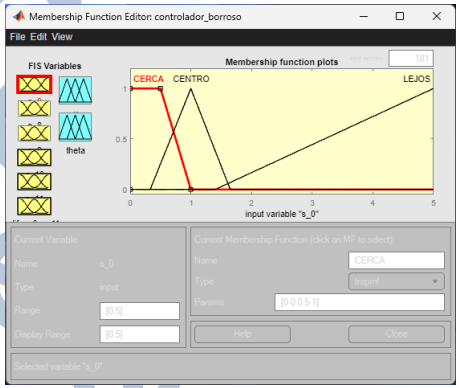
Figura: Entradas y salidas del controlador

- S_0
- S_9
- S_{10}
- S_{11}
- $S_0 - S_{11}$
- V
- θ

Controlador borroso

Variables S_i

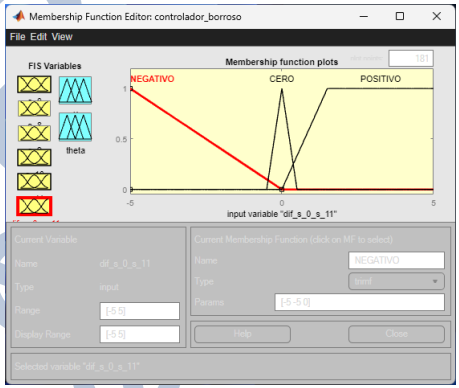
Tres conjuntos borrosos: `CERCA`, `CENTRO`, `LEJOS`



Controlador borroso

Variable $S_0 - S_{11}$

Tres conjuntos borrosos: `CERO`, `POSITIVO`, `NEGATIVO`



Controlador borroso

Variable *V*

Cinco conjuntos borrosos: `CERO`, `POS`, `MPOS`, `NEG`, y `MNEG`.

Controlador borroso

Variable θ

Cinco conjuntos borrosos: `CERO`, `POS`, `MPOS`, `NEG`, y `MNEG`.

Controlador borroso

Diseño de las reglas

Si se piensa de una **forma algorítmica**...

1. Acelerar marcha atrás girando hacia la izquierda ligeramente hasta estar cerca de la pared
2. Al estar cerca de la pared, girar el volante a la derecha hasta tener el coche recto, y entonces dejar el volante recto, todo mientras se da marcha atrás
3. Seguir marcha atrás hasta llegar al hueco
4. Girar todo el volante a la izquierda dando marcha atrás hasta tener el coche recto
5. Girar el volante a la derecha hasta dejar las ruedas rectas y seguir dando marcha atrás
6. Parar al llegar cerca de la pared

Controlador borroso

Diseño de las reglas

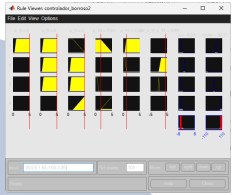
Se traducen en:

$$
\begin{array}{lll}
R_1: & \neg\texttt{CERCA}(S_0) \wedge \neg\texttt{CERCA}(S_{11}) & \longrightarrow \quad \texttt{MNEG}(V) \wedge \texttt{POS}(\theta) \\
R_2: & \texttt{POS}(S_0 - S_{11}) \wedge \texttt{CERCA}(S_{11}) & \longrightarrow \quad \texttt{NEG}(V) \wedge \texttt{MNEG}(\theta) \\
R_3: & \texttt{CERO}(S_0 - S_{11}) \wedge \texttt{CERCA}(S_{10}) & \longrightarrow \quad \texttt{MNEG}(V) \wedge \texttt{CERO}(\theta) \\
R_4: & \texttt{LEJOS}(S_9) \wedge \texttt{LEJOS}(S_{10}) & \longrightarrow \quad \texttt{MNEG}(V) \wedge \texttt{MPOS}(\theta)
\end{array}
$$

Controlador borroso

Diseño de las reglas

¡Tenemos **problemas**!, hemos pensado de forma **algorítmica**, no es forma de **estados "disjuntos"**. Ayuda con *View Rules*.



Solución: hacer más restrictivas reglas que se verifican en momentos indeseados.

$$\left.\begin{array}{lrcl} R_i: & (p \wedge q) & \longrightarrow & t \\ R_j: & (u \wedge v) & \longrightarrow & r \end{array}\right\} \quad \begin{array}{lrcl} R_i: & (p \wedge q \wedge \neg u \wedge \neg v) & \longrightarrow & t \\ R_j: & (u \wedge v) & \longrightarrow & r \end{array}$$

Controlador borroso

Diseño de las reglas

Experimentando y haciendo "*debugging*" con *View Rules*, se obtienen las reglas definitivas:

$$
\begin{array}{lll}
R_1: & (\neg \mathtt{CERCA}(S_0) \wedge \neg \mathtt{LEJOS}(S_9) \wedge \neg \mathtt{LEJOS}(S_{10}) \wedge \neg \mathtt{CERCA}(S_{11})) & \longrightarrow \quad \mathtt{MNEG}(V) \wedge \mathtt{POS}(\theta) \\
R_2: & (\mathtt{POS}(S_0 - S_{11}) \wedge \mathtt{CERCA}(S_9) \wedge \mathtt{CERCA}(S_{11})) & \longrightarrow \quad \mathtt{NEG}(V) \wedge \mathtt{MNEG}(\theta) \\
R_3: & (\mathtt{CERO}(S_0 - S_{11}) \wedge \neg \mathtt{CERCA}(S_9) \wedge \mathtt{CERCA}(S_{10})) & \longrightarrow \quad \mathtt{MNEG}(V) \wedge \mathtt{CERO}(\theta) \\
R_4: & (\mathtt{LEJOS}(S_9) \wedge \mathtt{LEJOS}(S_{10})) & \longrightarrow \quad \mathtt{MNEG}(V) \wedge \mathtt{MPOS}(\theta)
\end{array}
$$

Controlador neuronal

Captura de datos

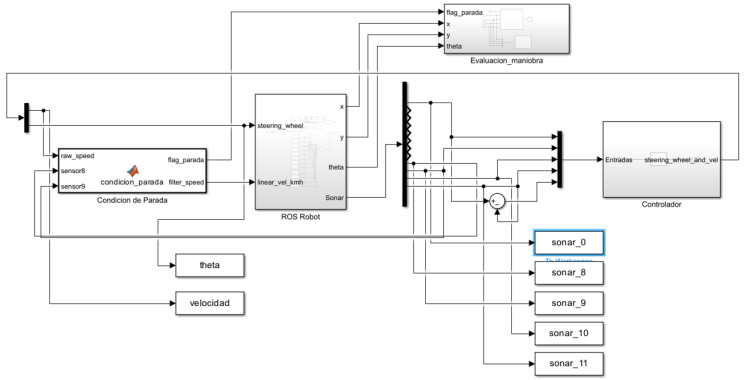
Se ha seguido los siguientes pasos para la **preparación de los datos de entrenamiento**:

1. Utilización del controlador borroso de la parte anterior de la práctica
2. Datos de entrenamiento mediante un script de captura
3. Unión de los datos de diferentes ejecuciones en un *dataset*
4. Almacenamiento de los datos en un archivo `.mat`
5. Configuración del modelo para asegurar los valores extraídos.

Controlador neuronal

Captura de datos

Se muestra el siguiente modelo con todas las configuraciones necesarias para la captura de datos



Controlador neuronal

Creación de la red neuronal

Con los datos almacenados en un archivo `.mat` se procede a la creación de la estructura de la red.

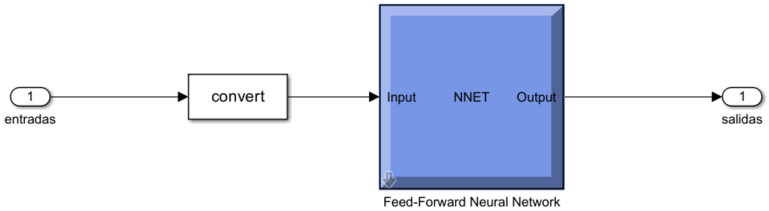
"El número de neuronas de la capa oculta no debe ser mayor al 15 % del número de vectores de entrenamiento."

- `neuronas = floor(size(X, 1) * 0.01)` (26 neuronas)
- `net = feedforwardnet(neuronas);`
- `net = configure(net, X', Y');`

Controlador neuronal

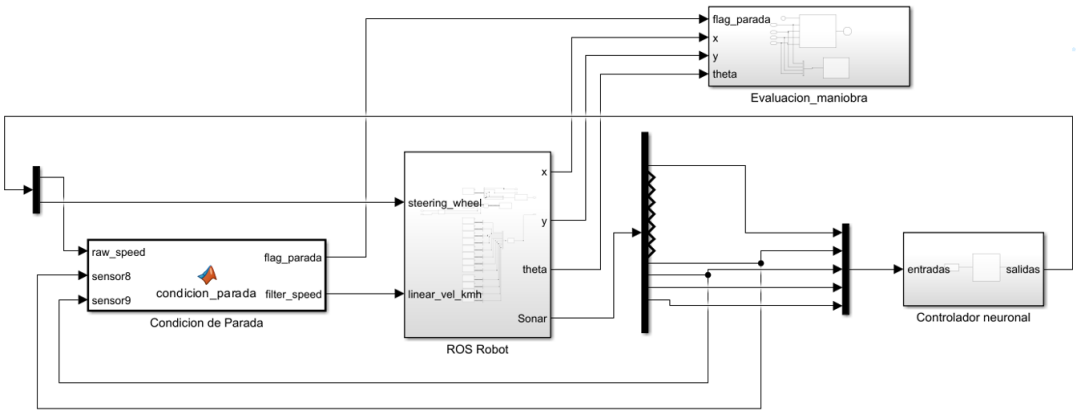
Creación del controlador neuronal

1. Se realiza el entrenamiento de la red con `trainlm`
2. Generación de bloque de Simulink (`gensim`) para probar su funcionamiento en el simulador
3. Se añade un bloque "`Data Type Conversion`" (en el modelo) entre la salida del multiplexor y la entrada de la red



Controlador neuronal

Modelo final



Controlador neuronal

Análisis de resultados

Durante el entrenamiento con Levenberg–Marquardt se obtienen buenos resultados, sin *overfitting*.