Contenido

[INTRODUCCION 1](#_Toc29555441)

[PARTE 1: Diseño manual de un control borroso de tipo MAMDANI. 2](#_Toc29555442)

[PARTE 1.1: Circuito Sin Obstáculos 3](#_Toc29555443)

[PARTE 1.2: Circuito con Obstáculos 1 5](#_Toc29555444)

[PARTE 1.3: Circuito con Obstáculos 2 8](#_Toc29555445)

[PARTE 1.4: Conclusiones Mamdani 10](#_Toc29555446)

[PARTE 2: Diseño automático de un controlador neuroborroso de tipo SUGENO. 11](#_Toc29555447)

[PARTE 2.1: Circuito Sin Obstáculos 11](#_Toc29555448)

[PARTE 2.2: Circuito con Obstáculos 1 11](#_Toc29555449)

[PARTE 2.3: Circuito con Obstáculos 2 11](#_Toc29555450)

[PARTE 2.4: Conclusiones Sugeno 11](#_Toc29555451)

[PARTE 3: Conclusiones generales 11](#_Toc29555452)

# INTRODUCCION

El objetivo de la práctica es el diseño del control de velocidad (lineal y angular) de un robot móvil para que éste recorra un circuito cerrado (delimitado por paredes), en el que pueden aparecer obstáculos estáticos, en el menor tiempo posible.

El robot cuenta con una serie de ***sensores de ultrasonidos*** distribuidos alrededor de este que son capaces de detectar objetos a una distancia en un rango [0.1- 5]m.

Imagen que contiene dispositivo

Descripción generada automáticamente

**Figura 1:** Distribución de los sensores

El comportamiento del robot y su interacción con el entorno se simula en la ***plataforma ROS-STDR***. Robot Operating System o ***ROS***, es una plataforma de desarrollo robótico que permite crear aplicaciones con múltiples sensores y actuadores de forma flexible. Es una colección de herramientas, bibliotecas y convenciones que tiene como objetivo simplificar la tarea de crear un comportamiento de robot complejo y robusto en una amplia variedad de plataformas robóticas.

A través de una Máquina Virtual con el sistema ROS Kinetic ya preinstalado podremos controlar nuestro robot desde Matlab.

Lo primero que hemos hecho ha sido configurar el entorno de trabajo necesario para manejar todo lo relativo al robot en el directorio ***robótica\_movil\_ws****.*

Además debemos añadir el espacio de trabajo al path por defecto añadiendo estas 2 lineas al final del archivo ***.bashrc***:

source ~/robotica\_movil\_ws/devel/setup.bash

export ROS\_PACKAGE\_PATH=$ROS\_PACKAGE\_PATH:~/robotica\_movil\_ws/

Y también añadiremos las IPs necesarias, extraídas con el comando ***ifconfig***, al final del mismo archivo, en nuestro caso serán las siguientes:

export ROS\_MASTER\_URI=http://192.168.1.26:11311

export ROS\_IP=:192.168.1.26

Una vez realizado todo este proceso de configuración ya podremos iniciar el simulador con el siguiente comando:

roslaunch stdr\_launchers PracticaFinal.launch

A lo largo de la práctica necesitaremos cambiar el mapa que recorrerá nuestro robot entre 3 distintos que se incluyen entre los archivos del Miniproyecto: ***EntornoSinObstaculos.png, EntornoConObstaculos.png y EntornoConObstaculos2.png***. Para ello simplemente basta con modificar el archivo de descripción del mapa ***EntornoPracticaFinal.yaml*** y especificar que archivo .png deseamos utilizar al iniciar el simulador.

Imagen que contiene captura de pantalla

Descripción generada automáticamente

**Figura 2:** Archivo de descripción del mapa EntornoPracticaFinal.yaml

# PARTE 1: Diseño manual de un control borroso de tipo MAMDANI.

En esta primera parte de la práctica se nos pide que nuestro ***amigobot*** recorra 3 circuitos utilizando un Controlador Borroso de ***tipo Mamdani***.

Utilizaremos la función ***fuzzy*** para generar un ***controlador.fis*** que añadiremos a nuestro bloque de Controlador en Simulink y conectaremos con el ***bloque ROS Robot*** a través de bloques multiplexores y demultiplexores. En la Figura 3 podemos ver como quedaría el archivo Simulink **cont\_mamdani.slx** sin ningún sensor conectado de los 8 disponibles, a lo largo de la práctica conectaremos aquellos que consideremos necesarios para una trayectoria exitosa:

Imagen que contiene cielo, captura de pantalla

Descripción generada automáticamente

**Figura 3:** Esquema de Simulink cont\_mamdani.slx

Por último, para terminar de configurar todo antes de comenzar las simulaciones debemos conectar los bloques Suscriptores del bloque ROS Robot a nuestra máquina virtual configurando la dirección red:

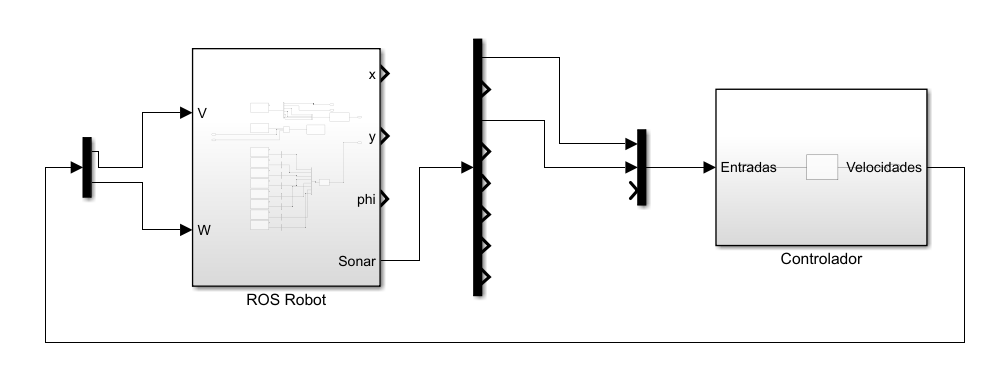
**Imagen que contiene captura de pantalla

Descripción generada automáticamente**

**Figura 4:** Configuración de red de bloques suscriptores

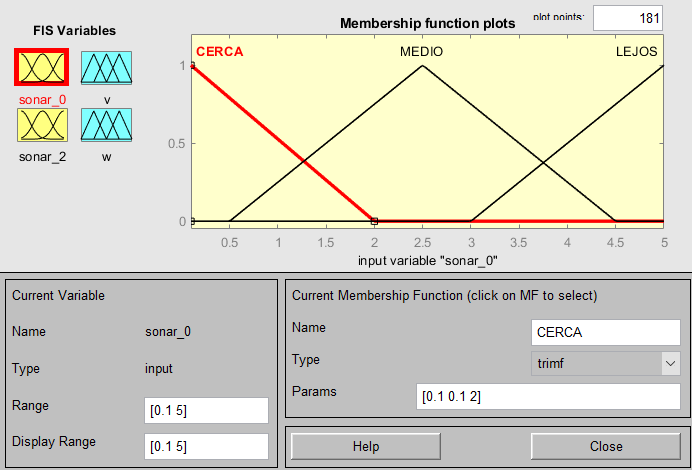
## PARTE 1.1: Circuito Sin Obstáculos

Lo primero que tendremos que hacer es realizar un controlador borroso de tipo Mamdani que ayude a nuestro robot a superar el circuito con el mapa sin obstáculos.

Para ello, lo primero que haremos será determinar cuántos sensores vamos a utilizar para este caso. Nuestra decisión fue utilizar dos sensores (sonar\_0 y sonar\_2), para mantenernos cerca de la pared interior. La arquitectura sería la siguiente:  


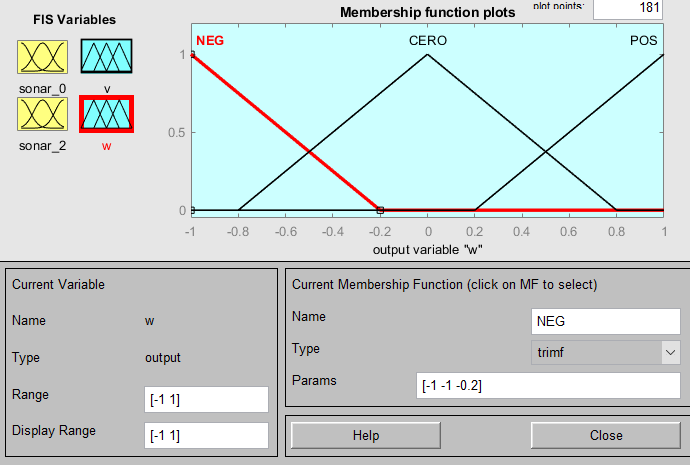
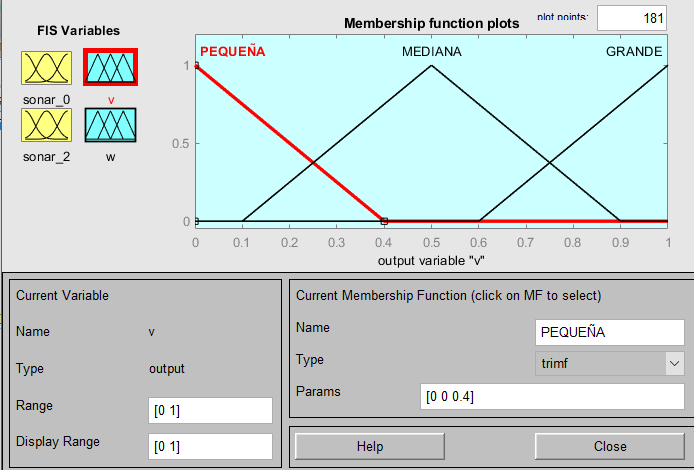
**Figura 5:** Arquitectura del sistema circuito sin obstáculos

Una vez tuvimos clara la arquitectura, nos pusimos con el controlador borroso. Teniendo en cuenta las medidas del circuito establecimos los mismos límites para el sonar\_0 y el sonar\_2:



**Figura 6:** Límites y funciones de pertenencia controlador sin obstáculos

Por otro lado, para las velocidades lineal y angular:

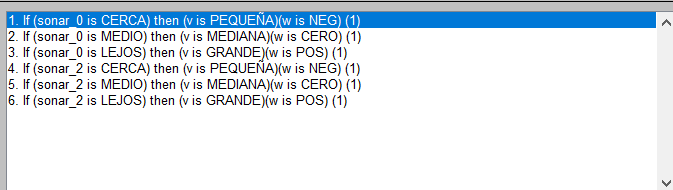


**Figura 7:** Límites y funciones de pertenencia W y V

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Funciones de pertenencia** | | | | |
| **Variable** | **Rango** | **Nombre de función** | **Forma** | **Parámetros** |
| **Sonar\_0** | [0.1, 5] | **CERCA** | Trimf | [0.1 0.1 2] |
| **MEDIO** | Trimf | [0.5 2.5 4.5] |
| **LEJOS** | Trimf | [3 5 5] |
| **Sonar\_2** | [0.1, 5] | **CERCA** | Trimf | [0.1 0.1 2] |
| **MEDIO** | Trimf | [0.5 2.5 4.5] |
| **LEJOS** | Trimf | [3 5 5] |
| **V** | [0, 1] | **PEQUEÑA** | Trimf | [0 0 0.4] |
| **MEDIANA** | Trimf | [0.1 0.5 0.9] |
| **GRANDE** | Trimf | [0.6 1 1] |
| **W** | [-1, 1] | **NEG** | Trimf | [-1 -1 -0.2] |
| **CERO** | Trimf | [-0.8 0 0.8] |
| **POS** | Trimf | [0.2 1 1] |

**Tabla 1:** Funciones de pertenencia de las variables

Y, por último, las reglas que modelarán el comportamiento del robot serán las siguientes:



**Figura 8:** Reglas controlador sin obstáculos

Una vez hemos terminado de configurar nuestro controlador borroso, lo guardaremos en el archivo **ContSinObstaculos.fis**.

Por último, cuando tuvimos todo conectado y configurado correctamente, procedimos a probar nuestro controlador con ROS ejecutando el siguiente código **Ej\_Mamdani.m**, especificando en nuestro esquema Simulink que el controlador a utilizar es el recién creado **ContSinObstaculos.fis**:

Imagen que contiene captura de pantalla

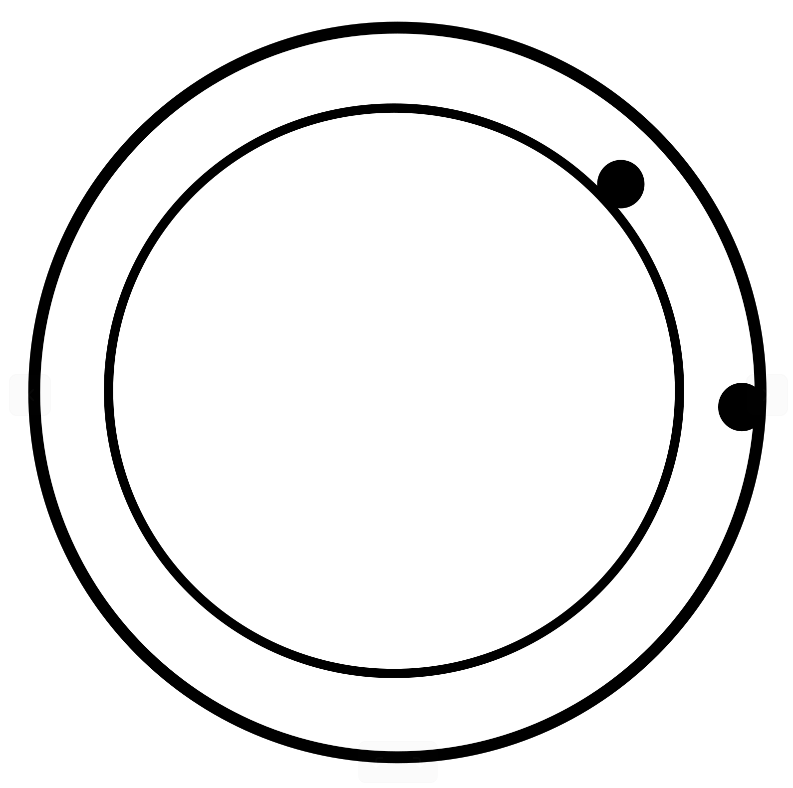
Descripción generada automáticamente

**Figura 9**: Código .m para la ejecución de nuestro controlador Mamdani.

Los resultados obtenidos fueron satisfactorios, ya que nuestro robot daba una vuelta completa al circuito sin chocarse con nada. Estos resultados pueden observarse en el archivo de vídeo adjuntado en el entregable del proyecto.

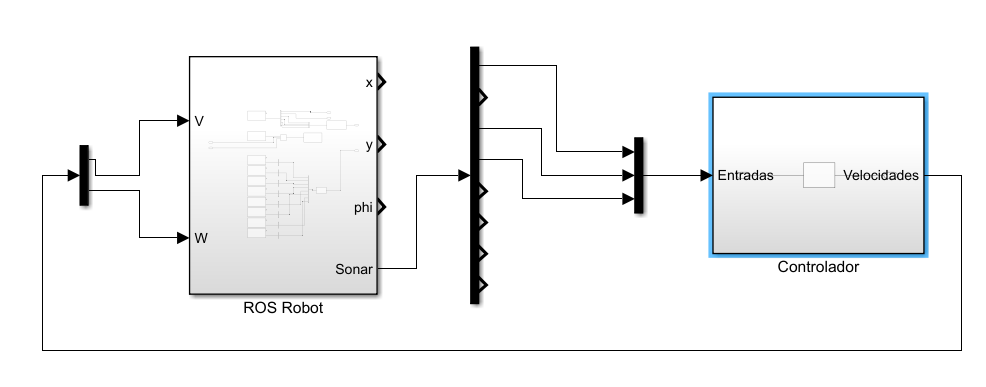
## PARTE 1.2: Circuito con Obstáculos 1

El siguiente circuito que tendrá que superar nuestro controlador es el “CircuitoConObstaculos1”, que podemos encontrar en la máquina virtual y tiene el siguiente aspecto:



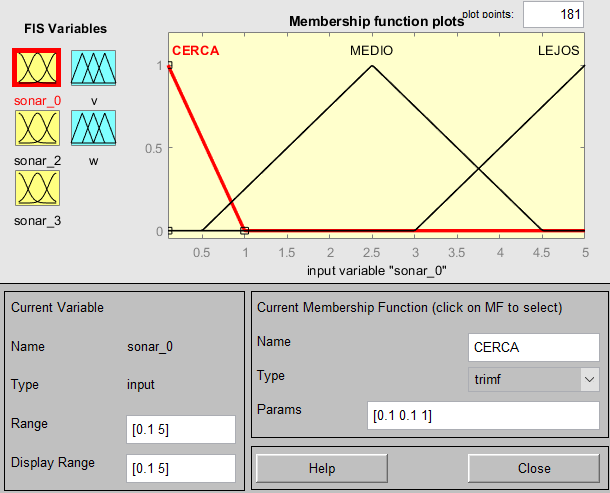
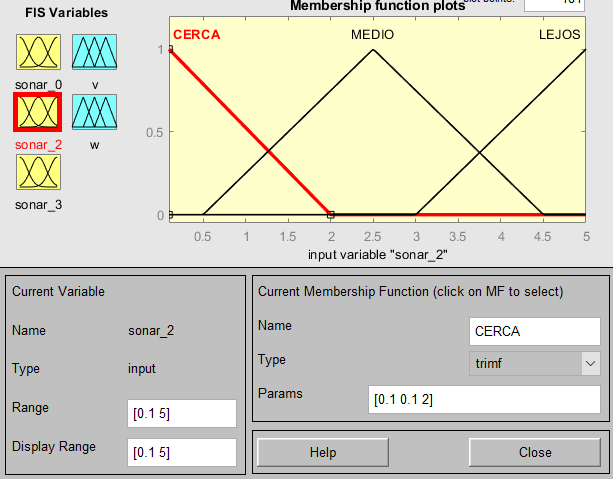
**Figura 10:** Circuito con obstáculos 1

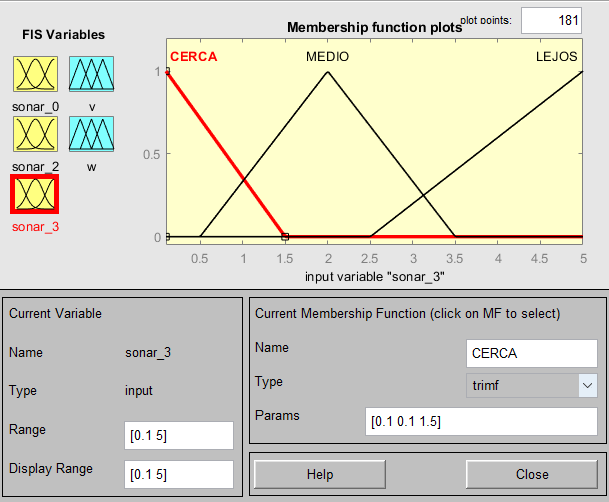
Como se puede observar, ahora nuestro robot tendrá que sortear dos obstáculos para dar una vuelta completa al circuito, por lo que deberemos gestionarlo en el controlador. La arquitectura cambiará, ahora utilizaremos 3 sensores para controlar que el robot no se choca con ningún obstáculo (sonar\_0, sonar\_2 y sonar\_3). Así, la nueva arquitectura queda de la siguiente manera:



**Figura 11:** Arquitectura del sistema para el circuito con obstáculos 1

El siguiente paso será modelar las funciones de pertenencia y límites de cada sonar, así como de la velocidad lineal y angular. Al introducir un nuevo sonar en el modelo, los límites de los sónares cambian, pero los de w y v quedan igual que en el modelo anterior. Así, los nuevos límites de los sónares serán:



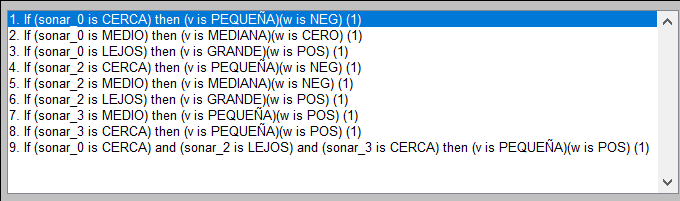


**Figura 12:** Funciones de pertenencia de los sónares para el circuito con obstáculos 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Funciones de pertenencia** | | | | |
| **Variable** | **Rango** | **Nombre de función** | **Forma** | **Parámetros** |
| **Sonar\_0** | [0.1, 5] | **CERCA** | Trimf | [0.1 0.1 1] |
| **MEDIO** | Trimf | [0.5 2.5 4.5] |
| **LEJOS** | Trimf | [3 5 5] |
| **Sonar\_2** | [0.1, 5] | **CERCA** | Trimf | [0.1 0.1 2] |
| **MEDIO** | Trimf | [0.5 2.5 4.5] |
| **LEJOS** | Trimf | [3 5 5] |
| **Sonar\_3** | [0.1, 5] | **CERCA** | Trimf | [0.1 0.1 1.5] |
| **MEDIO** | Trimf | [0.5 2 3.5] |
| **LEJOS** | Trimf | [2.5 5 5] |
| **V** | [0, 1] | **PEQUEÑA** | Trimf | [0 0 0.4] |
| **MEDIANA** | Trimf | [0.1 0.5 0.9] |
| **GRANDE** | Trimf | [0.6 1 1] |
| **W** | [-1, 1] | **NEG** | Trimf | [-1 -1 -0.2] |
| **CERO** | Trimf | [-0.8 0 0.8] |
| **POS** | Trimf | [0.2 1 1] |

**Tabla 2:** Funciones de pertenencia de las variables

Y, por último, las reglas que modelarán el comportamiento del robot serán las siguientes:



**Figura 13:** Reglas para el circuito con obstáculos 1

Una vez configurado completamente nuestro nuevo controlador borroso, lo guardaremos en el archivo **ContConObstaculos.fis**.

Por último, cuando tuvimos todo conectado y configurado correctamente, procedimos a probar nuestro controlador con ROS. Para ello utilizamos exactamente el mismo código **Ej\_Mamdani.m** mencionado en el apartado anterior y expuesto en la Figura 9. Además, fue necesario especificar en nuestro esquema Simulink que el controlador a utilizar esta vez es **ContConObstaculos.fis**.

Los resultados obtenidos fueron satisfactorios, ya que nuestro robot daba una vuelta completa al circuito sin chocarse con nada y evitando los obstáculos. Estos resultados pueden observarse en el archivo de vídeo adjuntado en el entregable del proyecto.

## PARTE 1.3: Circuito con Obstáculos 2

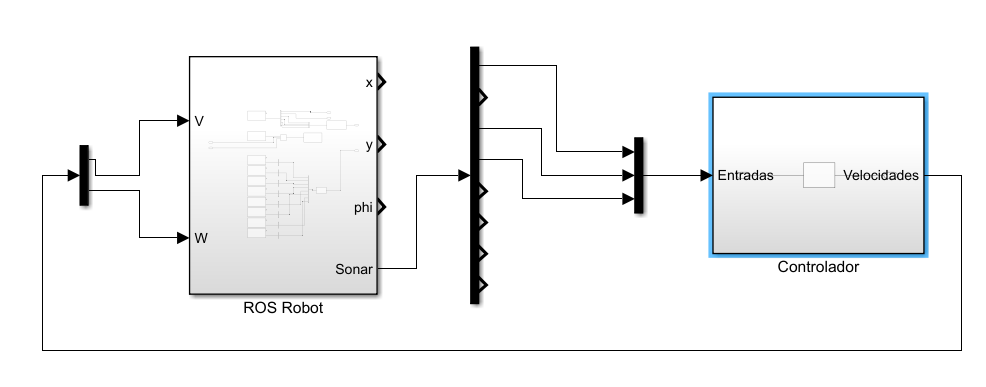
El último circuito que tendrá que superar nuestro controlador es el “CircuitoConObstaculos2”, que podemos encontrar en la máquina virtual y tiene el siguiente aspecto:

Imagen que contiene objeto

Descripción generada automáticamente

**Figura 14:** Circuito con obstáculos 2

En este caso nuestro amigobot deberá recorrer un circuito que cuenta con **6 obstáculos** distribuidos entre las paredes interior y exterior de este. Para realizar este circuito no ha sido necesario utilizar las lecturas de ningún sonar extra, por lo que las entradas de nuestro controlador seguirán siendo sonar\_0, sonar\_2 y sonar\_3 como se puede apreciar en la Figura 15:



**Figura 15:** Arquitectura del sistema para el circuito con obstáculos 2

Sin embargo, aunque los sónares utilizados son los mismos que en el apartado anterior, si que ha sido necesario modificar algunas funciones de pertenencia de los sónares para evitar choques con las paredes. Las funciones de pertenencia de las salidas V y W se mantienen exactamente igual. En la Figura 16 podemos ver las nuevas funciones de pertenencia de los sónares:

Imagen que contiene captura de pantalla

Descripción generada automáticamenteImagen que contiene captura de pantalla

Descripción generada automáticamente

Imagen que contiene captura de pantalla

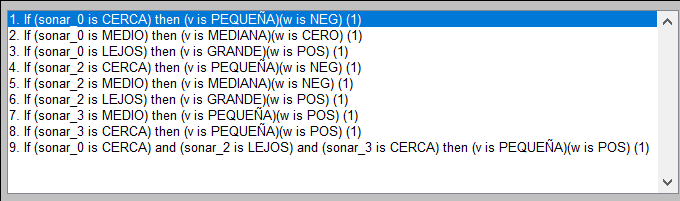
Descripción generada automáticamente

**Figura 16:** Funciones de pertenencia de los sónares para el circuito con obstáculos 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Funciones de pertenencia** | | | | |
| **Variable** | **Rango** | **Nombre de función** | **Forma** | **Parámetros** |
| **Sonar\_0** | [0.1, 5] | **CERCA** | Trimf | [0.1 0.1 1] |
| **MEDIO** | Trimf | [0.5 2.5 4.5] |
| **LEJOS** | Trimf | [3 5 5] |
| **Sonar\_2** | [0.1, 5] | **CERCA** | Trimf | [0.1 0.1 2] |
| **MEDIO** | Trimf | [0.5 2.5 4.5] |
| **LEJOS** | Trimf | [3 5 5] |
| **Sonar\_3** | [0.1, 5] | **CERCA** | Trimf | [0.1 0.1 2] |
| **MEDIO** | Trimf | [0.5 2 3.5] |
| **LEJOS** | Trimf | [2.5 5 5] |
| **V** | [0, 1] | **PEQUEÑA** | Trimf | [0 0 0.4] |
| **MEDIANA** | Trimf | [0.1 0.5 0.9] |
| **GRANDE** | Trimf | [0.6 1 1] |
| **W** | [-1, 1] | **NEG** | Trimf | [-1 -1 -0.2] |
| **CERO** | Trimf | [-0.8 0 0.8] |
| **POS** | Trimf | [0.2 1 1] |

**Tabla 3:** Funciones de pertenencia de las variables

Y, por último, las reglas que modelarán el comportamiento del robot serán las siguientes:



**Figura 16:** Reglas para el circuito con obstáculos 1

Una vez hemos terminado de configurar este último controlador borroso, lo guardaremos en el archivo **ContConObstaculos2.fis**.

Por último, cuando tuvimos todo conectado y configurado correctamente, procedimos a probar nuestro controlador con ROS. De nuevo, el código utilizado para ejecutar la simulación es **Ej\_Mamdani.m** y especificaremos en Simulink que el controlador a utilizar es **ContConObstaculos2.fis**.

Los resultados obtenidos fueron satisfactorios, ya que nuestro robot daba una vuelta completa al circuito sin chocarse con nada y evitando los obstáculos. Estos resultados pueden observarse en el archivo de vídeo adjuntado en el entregable del proyecto.

## PARTE 1.4: Conclusiones Mamdani

Durante todo el proceso de desarrollo de Mamdani hemos generado un total de 3 controladores borrosos:

* **ContSinObstaculos:** Utilizado en el apartado 2.1 para realizar el circuito sin obstáculos.
* **ContConObstaculos:** Utilizado en el apartado 2.2 para realizar el circuito con obstáculos 1.
* **ContConObstaculos2:** Utilizado en el apartado 2.3 para realizar el circuito con obstáculos 2.

A diferencia de los otros 2 primeros controladores desarrollados, este último **si se puede utilizar en todos los circuitos**. Si intentásemos utilizar el **ContSinObs** en un circuito con obstáculos nuestro robot acabaría chocando. Y si intentásemos utilizar el **ContConObs1** en este último **CircuitoConObstaculos2** también acabaríamos colisionando nuestro amigobot.

Por lo tanto, el controlador definitivo que debemos utilizar es **ContConObs2**, ya que será capaz de realizar los 3 circuitos sin ningún tipo de problema.

# PARTE 2: Diseño automático de un controlador neuroborroso de tipo SUGENO.

Al igual que en la primera parte del Miniproyecto, en esta segunda se nos pide que nuestro **amigobot** recorra los 3 mismos circuitos anteriores, pero esta vez debemos desarrollar **controladores neuroborrosos de tipo Sugeno** en lugar de Mamdani.

Para ello, utilizaremos la función ***anfisedit***, en lugar de la función fuzzy utilizada anteriormente, para generar un **controlador.fis** que añadiremos a nuestro bloque Controlador en Simulink. Sin embargo, con la función anfisedit surge un problema, y es que solamente es capaz de generar un **controlador con una única salida**.

Por lo tanto, como necesitamos gestionar 2 salidas, V y W, debemos generar **2 controladores neuroborrosos** distintos (**ControladorVelocidadLineal y ControladorVelocidadAngular**) y conectarlos al bloque ROS Robot.

En la Figura 17 podemos ver como quedaría nuestro archivo Simulink **cont\_sugeno.slx** sin ningún sensor conectado de los 8 disponibles, a lo largo de la práctica conectaremos aquellos que consideremos necesarios para una trayectoria exitosa:

Imagen que contiene captura de pantalla

Descripción generada automáticamente

**Figura 17**: Esquema de Simulink cont\_sugeno.slx.

En cuanto a la configuración de red de los bloques subscriber de ROS Robot, al igual que en la Parte 1, debemos añadir la IP de la máquina virtual ROS.

## PARTE 2.1: Circuito Sin Obstáculos

## PARTE 2.2: Circuito con Obstáculos 1

## PARTE 2.3: Circuito con Obstáculos 2

## PARTE 2.4: Conclusiones Sugeno

# PARTE 3: Conclusiones generales