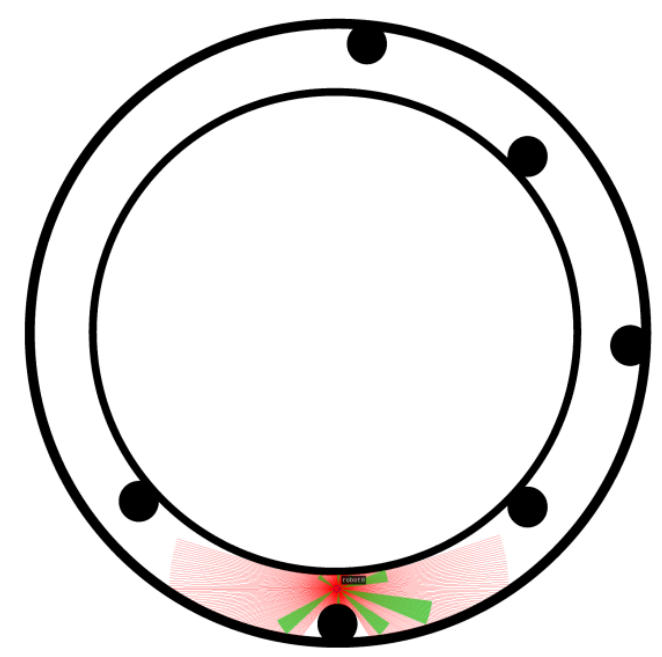
Miniproyecto: Control de robot móvil



**SISTEMAS DE CONTROL INTELIGENTE**

**05/1/2020**

**Autores:** Álvaro de las Heras Fernández

Álvaro Maestre Santa

## Índice

[Índice 2](#_Toc29754985)

[1. Introducción 3](#_Toc29754986)

[2. Desarrollo de la práctica 7](#_Toc29754987)

[2.1. Parte I 7](#_Toc29754988)

[Parte II 15](#_Toc29754989)

[3. Conclusión 19](#_Toc29754990)

[4. Bibliografía 19](#_Toc29754991)

## 

## 1. Introducción

El objetivo de esta práctica consiste en diseñar un control de velocidad lineal y angular de un robot móvil para que éste sea capaz de recorrer un circuito cerrado, en el que pueden aparecer obstáculos estáticos, en el menor tiempo posible.

En primer lugar, vamos a hablar de los diferentes circuitos, los cuales, nuestro robot debe ser capaz de realizarlo de forma autónoma.

1. **Circuito sin obstáculos:** Este circuito es el más **básico** de todos, ya que no tiene ningún obstáculo, como podemos ver en la figura 1.

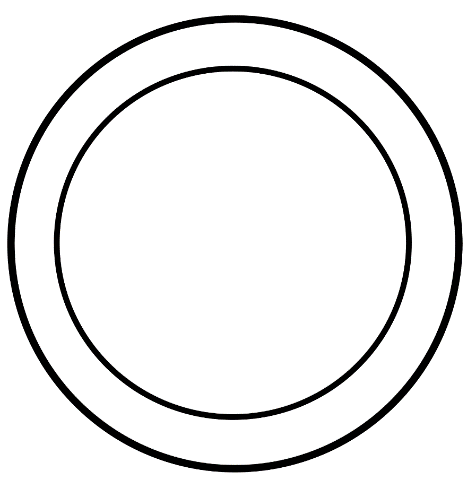


Fig. 1. Circuito sin obstáculos.

1. **Circuito con obstáculos 1:** En este circuito podemos ver **dos obstáculos** estáticos, este circuito tiene una dificultad media, debido a que tiene un obstáculo a la izquierda primero y seguidamente uno a la derecha, como se puede ver en la figura 2.

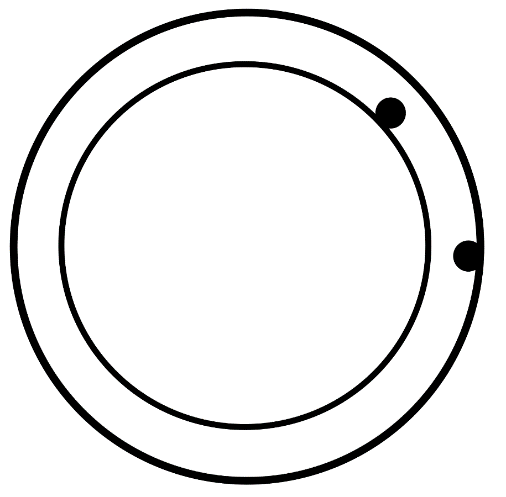


Fig. 2. Circuito con obstáculos 1.

1. **Circuito con obstáculos 2:** En este circuito podemos ver muchos más obstáculos estáticos que en el circuito con obstáculos 1, este circuito es el más difícil de todos, como se puede mostrar en la figura 3.

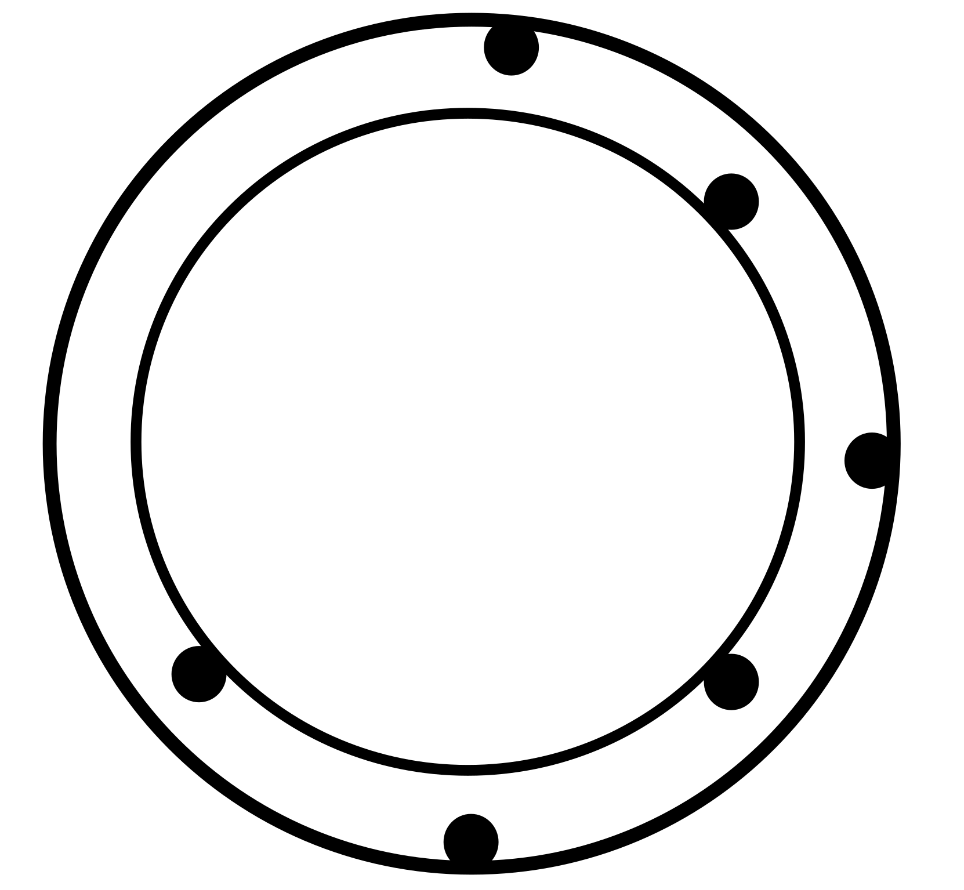


Fig. 3. Circuito con obstáculos 2.

Una vez explicado los circuitos vamos a explicar cómo está formado nuestro robot. En primer lugar, hay que tener en cuenta las dimensiones y las velocidades máximas que puede alcanzar, para ello, tenemos los siguientes datos:

* **Longitud:** 33 cm.
* **Anchura:** 28 cm.
* **Altura:** 15 cm.
* **Velocidad lineal máx.:** 1 m/s.
* **Velocidad angular máx.:** 1 rad/s.

Una vez mostrado estos datos, vamos a hablar de los sensores que tiene nuestro robot.

* **Sensores de posicionamiento** absoluto con respecto al centro geométrico del círculo que forma el circuito.
* **8 sensores de ultrasonidos**, dispuestos de tal forma que rodean al robot, para así detectar los obstáculos, estos sensores tienen un rango de detección que va desde los **0.1 metros** hasta los **5 metros**.

Ahora vamos a explicar cómo hemos configurado la máquina virtual.

Esta práctica esta creada en **ROS**, que es un middleware robótico, es decir, una colección de frameworks para el desarrollo de software de robots. Para ello, nos ha sido proporcionado para esta práctica una máquina virtual creada con el programa VirtualBox, con ROS instalado. Ahora bien, aunque se nos haya proporcionado la máquina virtual, tuvimos que configurarla para que esta se pudiese conectar con Matlab y viceversa, y es esto, lo que vamos a explicar ahora. En primer lugar, vamos a ver el escritorio de Ubuntu, como bien se muestra en la figura 4.



Fig. 4. Escritorio Ubuntu.

Posteriormente abrimos el terminal de Ubuntu y escribimos *ifconfig.* Ya que debíamos de saber qué dirección IP tenía nuestra máquina virtual para poder establecer la conexión con Matlab. Una vez sabida la dirección IP, modificamos el fichero *bashrc*, para ello pusimos el siguiente comando ***sudo gedit ~/.bashrc****,* al ejecutar este comando, se nos abrió un editor de texto, y manipulamos la información del fichero *bashrc*, lo único que hicimos fue añadir unas líneas al final del archivo, con la dirección IP consultada previamente.

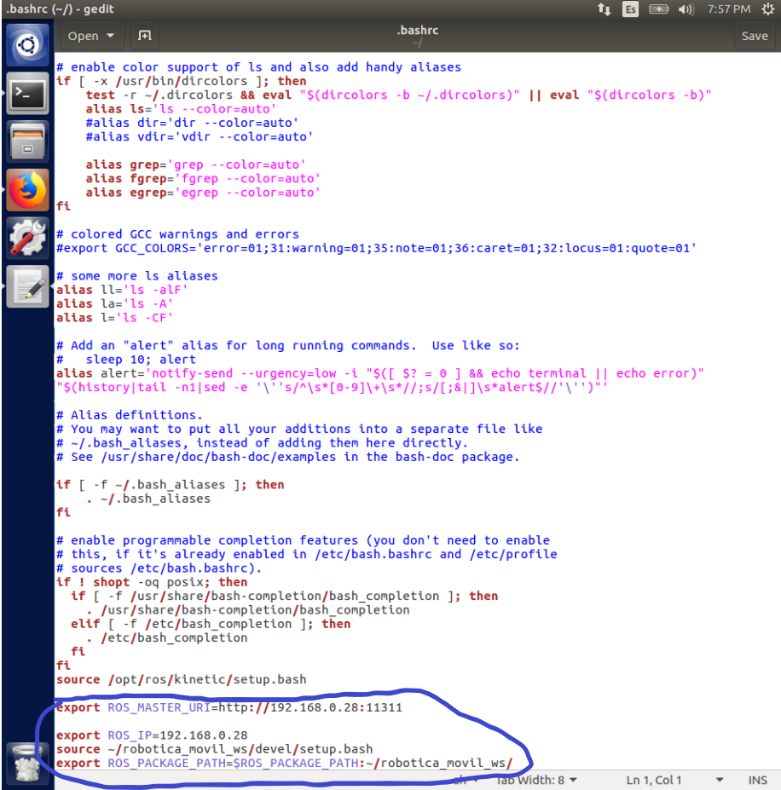
****

Fig. 5. Archivo Bashrc.

Una vez configurado esto, pasamos a configurar Matlab, para ello descargamos los archivos que nos proporcionaron los profesores, y modificamos las direcciones IPs, para que se pudiese conectar con ROS. En primer lugar, nos fuimos al fichero, ControlManualRobot.m, como podemos ver en la figura 6, y modificamos ROS\_MASTER\_IP, que sería la IP de la máquina virtual, y ROS\_IP, que sería la IP de Windows, para ello, abrimos el terminal de Windows y escribimos el comando ipconfig.

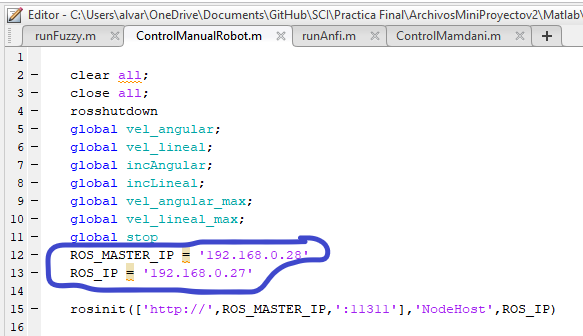


Fig 6. ControlManualRobot.m Matlab.

Una vez que hicimos esto, abrimos el archivo test\_controler\_generic.slx y modificamos los subscribe, poniendo la dirección IP de la máquina virtual.

Por último, volvimos a la máquina virtual e instalamos ROS Launcher, a través del terminal, y siguiendo los pasos que nos proporciona la documentación dada por los profesores. Para ejecutar el programa ROS, pusimos el siguiente comando ***roslaunch stdr\_launchers PracticaFinal.launch***, al ejecutar este comando, nos aparece lo siguiente:

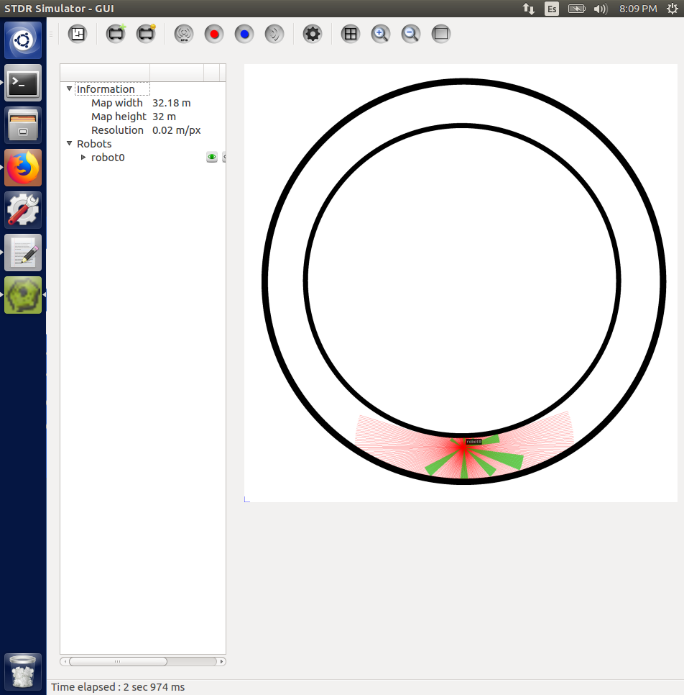
****

Fig. 7. Programa ROS.

Al abrir el programa, procedimos a crear nuestro robot, siguiente los pasos que nos dan los profesores y por último cargamos el robot en la imagen, dando como resultado la figura 7.

## 2. Desarrollo de la práctica

### 2.1. Parte I

#### **2.1.1. Planteamiento**

El objetivo de esta parte es diseñar un controlador de lógica borrosa de tipo Mamdani, para superar un circuito de obstáculos con forma de corona circular, en el menor tiempo posible. Como ayuda para conseguirlo el robot dispone de un total de 8 sonares situados en distintos ángulos cubriendo casi todas las zonas.

Con este objetivo en mente lo primero de todo es estudiar la estrategia a seguir. Al ser un circuito tan grande con forma circular la opción más eficaz es que haga el recorrido lo más pegado al interior, de tal forma que sea el menor recorrido. Como se puede comprobar calculando los perímetros de las trayectorias:

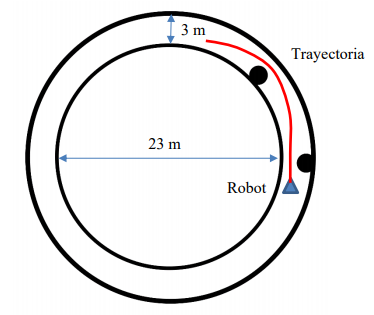


Fig. 8 Dimensiones del circuito.

Obviando el tamaño del robot y las paredes se puede observar como el recorrido exterior supone un aumento de 9 metros de distancia, que se traduciría en unos 9 segundos de tiempo extra (1 m/s). Por tanto el robot avanzará pegado al interior.

Al avanzar por el interior del circuito se han de tener en cuenta todos los obstáculos que se puede encontrar en su camino tanto en el anillo interior como en el exterior, pudiendo tener varios seguidos. Por eso hicimos un esquema con todos los tipos de obstáculos posibles que se podía encontrar el robot, mientras avanzaba por el interior. En el caso de los obstáculos del anillo exterior observamos que no afectaban al robot, siendo relevantes los internos, que si obligaban a modificar la trayectoria del robot.



Fig. 9 Posibles obstáculos a enfrentarse en el circuito.

Por este motivo utilizamos los 3 situados en el lado izquierdo del robot correspondientes a los ángulos de 90º, 41º y 15º, como se observa en la figura.

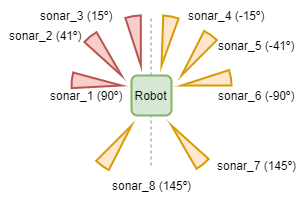


Fig. 10 Sonares del robot y los elegidos en rojo.

El sónar de 90º se encarga de comprobar que el robot avanza de forma paralela a la pared, al menos hasta que llega a un obstáculo. El obstáculo se detecta con el sónar de 15º porque detecta lo que tiene enfrente el robot. El sónar de 41º se usa cuando se rebasa el obstáculo para no perder las referencias en el momento de viraje del robot, de tal forma que siempre apunta este sónar al obstáculo cuando se rebasa.

#### 

Fig. 11 Comportamiento del robot al superar un obstáculo interno.

Al realizar este planteamiento hemos conseguido simplificar el problema centrándonos en los aspectos clave de éste, haciendo que use únicamente 3 sónares y unas reglas más sencillas basadas en que este cerca de la pared interna.

#### **2.1.2. Mamdani**

El controlador que se emplea en esta parte es el Mamdani, de ahí la necesidad de plantear correctamente el funcionamiento del robot ante los distintos obstáculos. Se basa en un conjunto de reglas, que se definen de forma sencilla con las MF (funciones de pertenencia) de las entradas y salidas. Básicamente la estructura consta de borrosificadores en la entrada, de tal forma que convierten el valor real que devuelven los sensores en un valor de pertenencia a partir de una función de pertenencia, (p.ej. una triangular de altura con alto, mediano y bajo). A partir de estas funciones se establecen unas reglas que se activarán en mayor o menor medida según los grados de pertenencia de las entradas, proyectando esto en su salida, estas reglas se pueden unir con distintos operadores como AND o OR, para tener una lógica más compleja. Por último es en esta salida dónde se obtiene un valor mediante la agregación de todos los resultados, dando como resultado una figura compuesta de las distintas áreas que cada regla ha creado para unos valores de entrada determinados. Sin embargo, aún quedaría obtener el valor resultado para ello se emplea un desborrosificador, que se encargará de dar un valor en función del área de esa figura compuesta, entre los métodos más destacados están COA[[1]](#footnote-1), COS[[2]](#footnote-2), etc.

Por tanto el algoritmo Mamdani se basa en una estructura compuesta de borrosificador a la entrada, reglas entre las entradas y salidas y un desborrisificador a la salida.

#### **2.1.3. Controlador**

En la configuración del Mamdani en general se puede ver como sigue la estructura anteriormente planteada con borrosificadores a la entrada, en este caso 3 correspondientes al sonar\_1, sonar\_2, y sonar\_3. Estos tienen su entrada acotada entre los valores que los sonares pueden detectar entre 0.1 y 5 metros. Cada sónar posee 3 funciones de pertenencia correspondientes a cercano, mediano y lejano, que son las distancias a las que se encuentra el obstáculo aproximadamente. Los valores de pertenencia de las funciones los hemos fijado mediante prueba y error hasta tener unos que funcionarán correctamente.

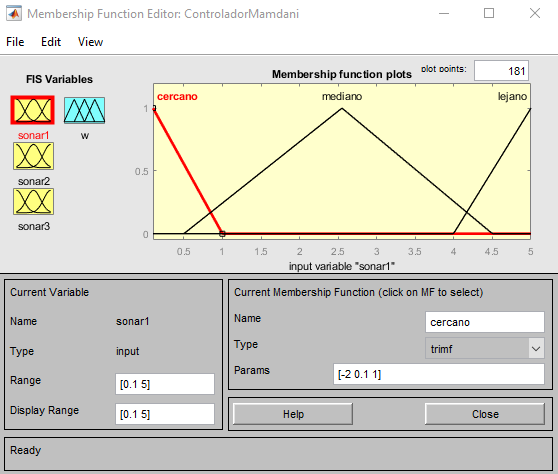


Fig. 12 Sonar\_1, su rango y sus MFs.

También se disponen de las reglas que son las encargadas del control del robot, que hemos decidido tratar individualmente para que mediante la agregación se obtuviera la trayectoria. En este caso calculamos únicamente la velocidad angular, encargada de los giros, teniendo una velocidad lineal constante, porque al ser tan grande el circuito no influía si se ponía al máximo (1 m/s).

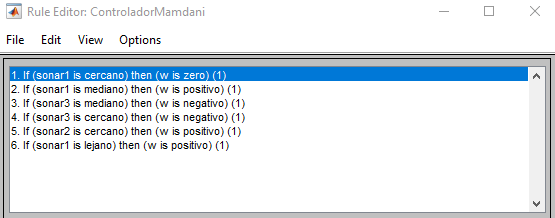


Fig. 13 Reglas del controlador.

Las reglas se basan en que cuando la velocidad angular es 0 el robot tiende a salirse del circuito, lo que implica que se ha de corregir este desvío con velocidad angular positivo, de ahí la regla 2, siendo cero cuando se encuentra lo suficientemente cerca. Con esta única regla somos capaces de recorrer la circunferencia interior sin obstáculos. Sin embargo, si se añaden obstáculos esto ya no se cumple, porque el robot no es capaz de detectar nada delante suyo, motivo por el que se añade el sonar\_3 (15º) y las respectivas reglas 3 y 4, con estas reglas se obliga a girar hacia el exterior, con velocidad angular negativa o cercana a 0.

Aún con estos cambios el robot a veces era incapaz de rebasar el obstáculo al quedarse ambos sonares sin captar ningún dato, lo que hizo que tuviéramos que añadir el sonar\_2 y su regla 5 para que hiciera que la velocidad angular fuera positiva y superará el obstáculo.

Al estar de forma independiente las reglas conseguimos un resultado que trazaba una trayectoria muy suavizada, sin necesidad de realizar todas las combinaciones posibles entre los sonares.

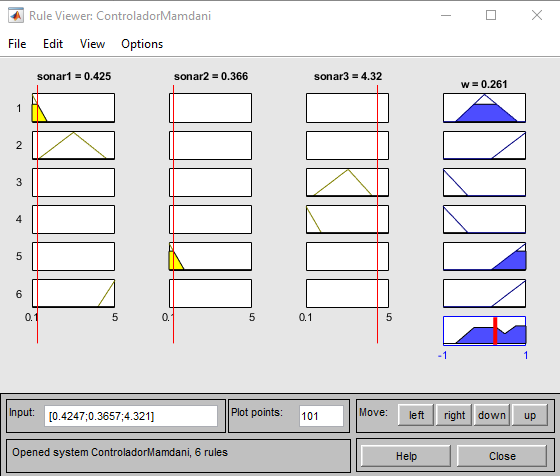


Fig. 14 Vista de los valores de las reglas.

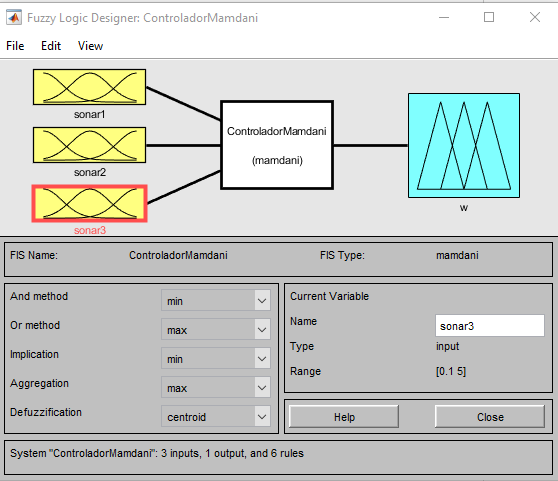


Fig. 15 Resaltada la configuración del controlador.

En la figura 15 se puede observar la configuración de reglas y del desborrosificador, en el caso de las reglas los métodos para cortar las funciones son de máximos o mínimos (podrían haber sido producto y suma), aunque en este caso solo empleamos la de implicación que escoge el valor mínimo y la de agregación que escoge el máximo.

Como hemos visto el método de desborrosificado es el de centroide, que se corresponde con el COA, que se trata de un método que funciona generalmente bien en el que se calcula el centro de la figura creada, siendo éste el punto de equilibrio de la figura. Lo calculará usando la figura resultante de las reglas, en este caso solo tenemos una para la velocidad angular que estará acotada entre -1 y 1.

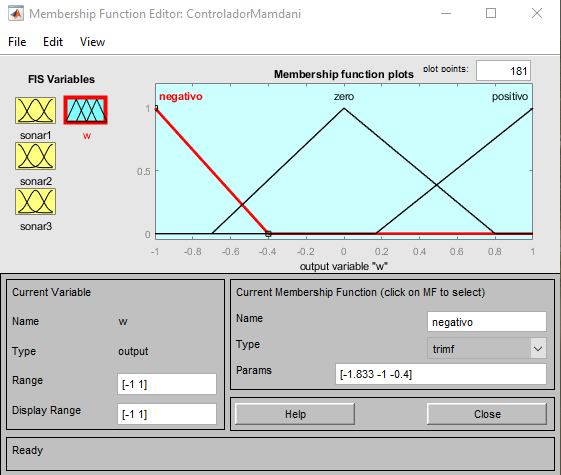


Fig. 16 Velocidad angular de salida, su rango y sus MFs.

#### **2.1.4. Resultados**

Tras realizar múltiples pruebas para afinar el controlador conseguimos obtener un robot que era capaz de realizar el circuito sin y con obstáculos, de una forma fluida y rápida, como se puede ver en las capturas. En estos casos fue capaz de dar la vuelta al circuito en un tiempo de 1 minuto y 20 segundos (en los distintos circuitos) aproximadamente, que medimos con un cronómetro.

Por tanto el aplicar un controlador Mamdani como solución al problema es viable, ofreciendo un controlador capaz de superar el problema en un buen tiempo, esto es debido a que se tenía un conocimiento bastante completo del problema y su funcionamiento, además de no ser excesivamente complejo éste.

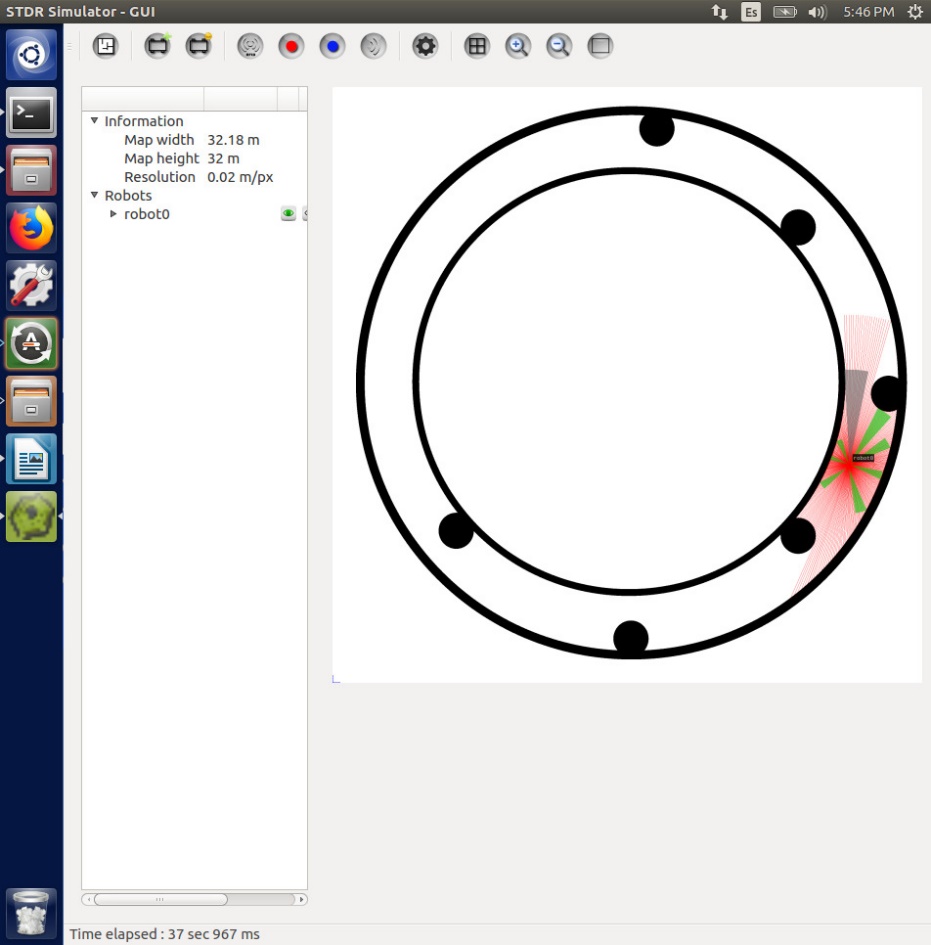
****

Fig. 17 Robot avanzando por zona sin obstáculos interiores.

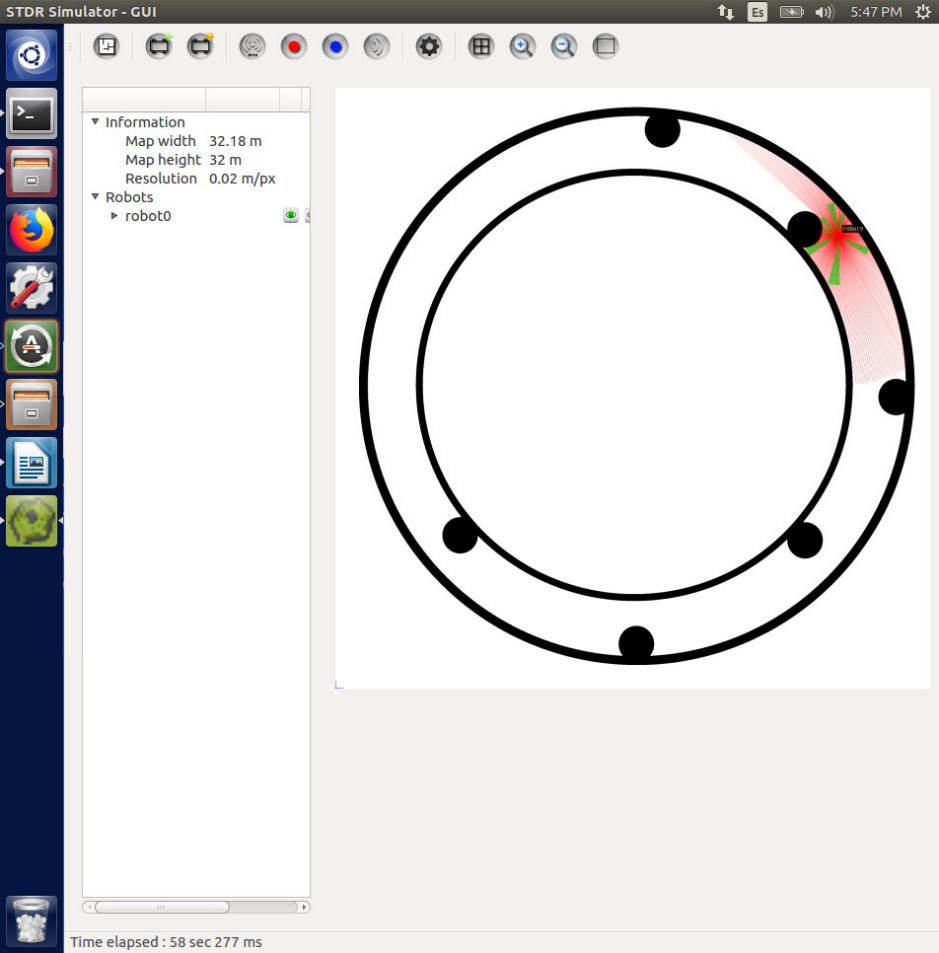
****

Fig. 18 Robot superando obstáculo interno.

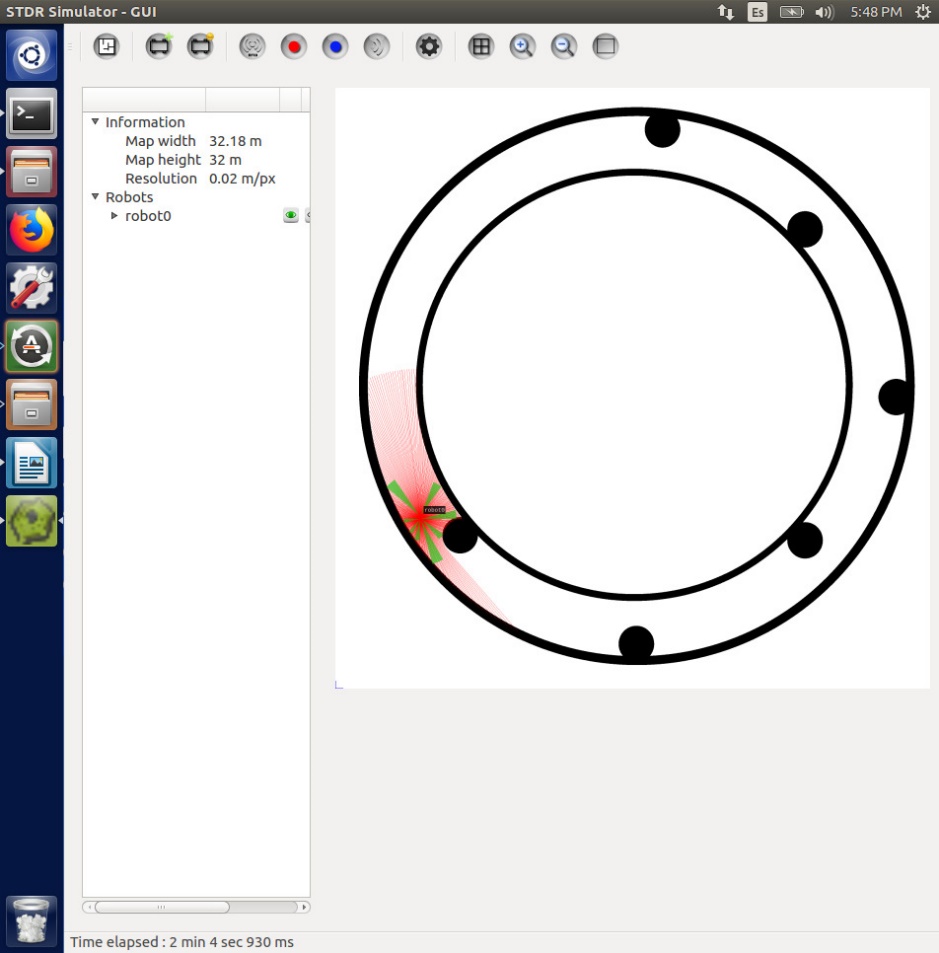


Fig. 19 Comportamiento del robot al superar el último obstáculo.

### Parte II

#### **2.2.1. Planteamiento**

El ejercicio ha consistido en la manipulación de valores de matrices, declarando una matriz

#### **2.2.2. Sugeno**

En esta parte el controlador que se emplea es el Sugeno. Este controlador se difiere con el Mamdani, en las reglas, ya que, en este caso, el consecuente de estas ya no es una etiqueta lingüística (más entendible), sino que es una función matemática para un momento dado. Esto hace que a la salida no requiera de un desborrosificador como si hace Mamdani.

La ventaja de usar Sugeno es que es mucho más matemático que Mamdani, lo que lo hace ideal para trabajar en computación, de ahí que se pueda combinar con redes neuronales. Sin embargo, es mucho más difícil de entender para las personas y por tanto de definir correctamente sus reglas y salidas.

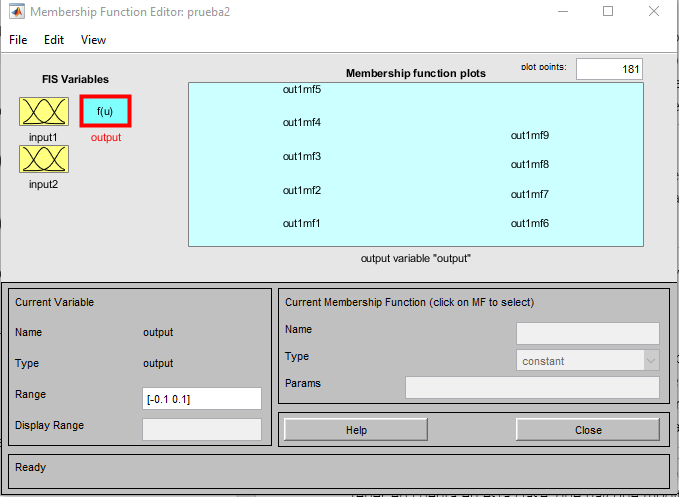
****

Fig. 20 Salida de las funciones de Sugeno.

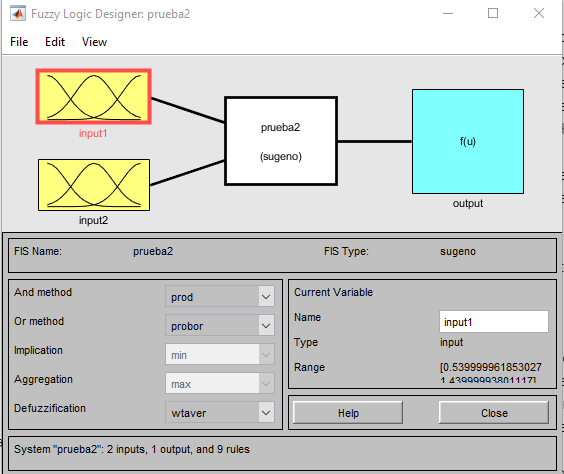


Fig. 21 Estructura del Sugeno creado.

Por tanto el algoritmo Sugeno, se basa en una estructura compuesta de borrosificador a la entrada y reglas con una función matemática a la salida, lo que lo diferencia del Mamdani.

#### **2.2.3. Neuroborroso**

Gracias a lo bien que funciona computacionalmente Sugeno se puede combinar con otras técnicas, de ahí que se use para control neuroborroso. Este tipo se caracteriza porque es una red neuronal la que se encarga de crear las reglas junto a sus funciones de salida a partir de un conjunto de datos para una salida, en este caso los sensores y la velocidad angular. Al ser una red neuronal, se han de tener en cuenta las característcas que poseen, especialmente aquellas que nos garanticen un buen entrenamiento. Es el motivo por el cual es necesario generar bastantes datos y no solo datos para el entrenamiento, sino que también para la validación. La validación es una parte muy importante en el proceso porque evita que salga una red sobreentrenada, es decir, que se dedique a memorizar el circuito, porque si se cambiará se chocaría. Para entrenarse se pueden emplear dos técnicas: la primera Backprogation y la segunda Híbrida (emplea también Backpropagation). Siendo más eficiente la híbrida según Matlab.

Estos entrenamientos, se utilizan junto a controladores diferentes como los basados en funciones de pertenencia de tipos como: trapmf (trapezoidal), disgmf (sigmoidal), gaussmf (gaussianas); controladores basados en sub-clustering, etc. Cada una de estas funciones, hará que nuestro controlador se comporte de una forma distinta, por lo tanto, es importante seleccionar una función que se adecue a nuestro problema y que aparte tenga un comportamiento normal el robot, y no tenga movimientos bruscos. Siendo estas las que se vayan modificando durante el entrenamiento con distintos pesos.

#### **2.2.4. Controlador**

En la configuración del Sugeno podemos apreciar como sigue la estructura planteada anteriormente con borrosificadores en la entrada, en este caso empleamos 2, correspondiente al sonar\_1 y sonar\_3. Para obtener los datos de entrada, para obtener la salida, hemos tenido que realizar el circuito manualmente a una velocidad constante, en este caso a 1 m/s, y coger el paquete training y almacenarlo en un paquete llamado datos\_validación.mat. Posteriormente lo cargamos en Matlab con la clase runAnfi, hay que tener en cuenta en esta clase, que hay que modificar el número de sensores que vamos a usar cada vez que queramos cambiar de sensores, ya que o si no, el robot no funcionará correctamente. Una vez cargado los datos, ya podemos crear nuestro controlador con el comando anfisedit, el cual, nos abrirá una ventana como en la figura 22. En esta ventana, cargamos los valores, en este caso hemos cargado valores para Training y para el Checking, para asegurarnos que no se aprenda de memoria el circuito (sobreentrene).

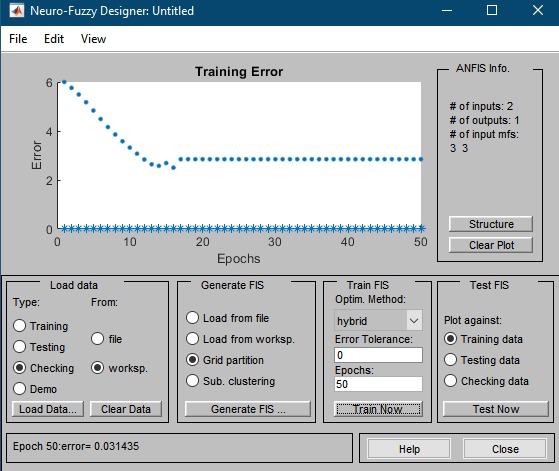


Fig. 22 Anfisedit resultados controlador.

Como se ve en la figura hay un momento en el que el error de validación deja de bajar y sube, siendo ese mínimo el que indique los valores para el FIS.

Por otra parte, al generar el FIS, hemos puesto la siguiente configuración, ver la figura 23.

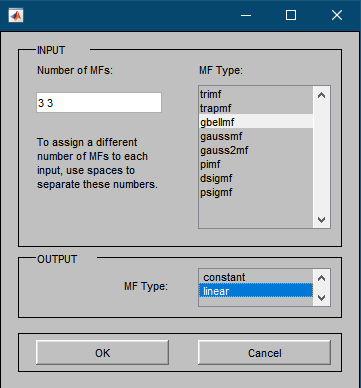


Fig. 23 Configuración para generar FIS.

Hemos elegido la función gbellmf, debido a que es la que mejor comportamiento a mostrado con nuestro robot, y por último hemos elegido que la salida sea lineal, al reducir considerablemente el error al entrenar y validar.

En este caso hemos usado pocos sensores e iteraciones debido a que era sencillo generar el controlador para este circuito, pero si fuera más complicado utilizaríamos más sensores, iteraciones y posiblemente más funciones de pertenencia, como probamos al intentar hacer el controlador para los obstáculos.

#### **2.2.5. Resultados**

Tras realizar numerosas pruebas para crear el controlador, hemos conseguido que funcione para el circuito sin obstáculos, ya que, con obstáculos, no hemos podido a pesar de los numerosos intentos que hemos realizado, probando distintas combinaciones de sensores, técnicas y datos. Podéis ver en la figura 4, como va haciendo el circuito a velocidad 1, para conseguir un tiempo bajo.

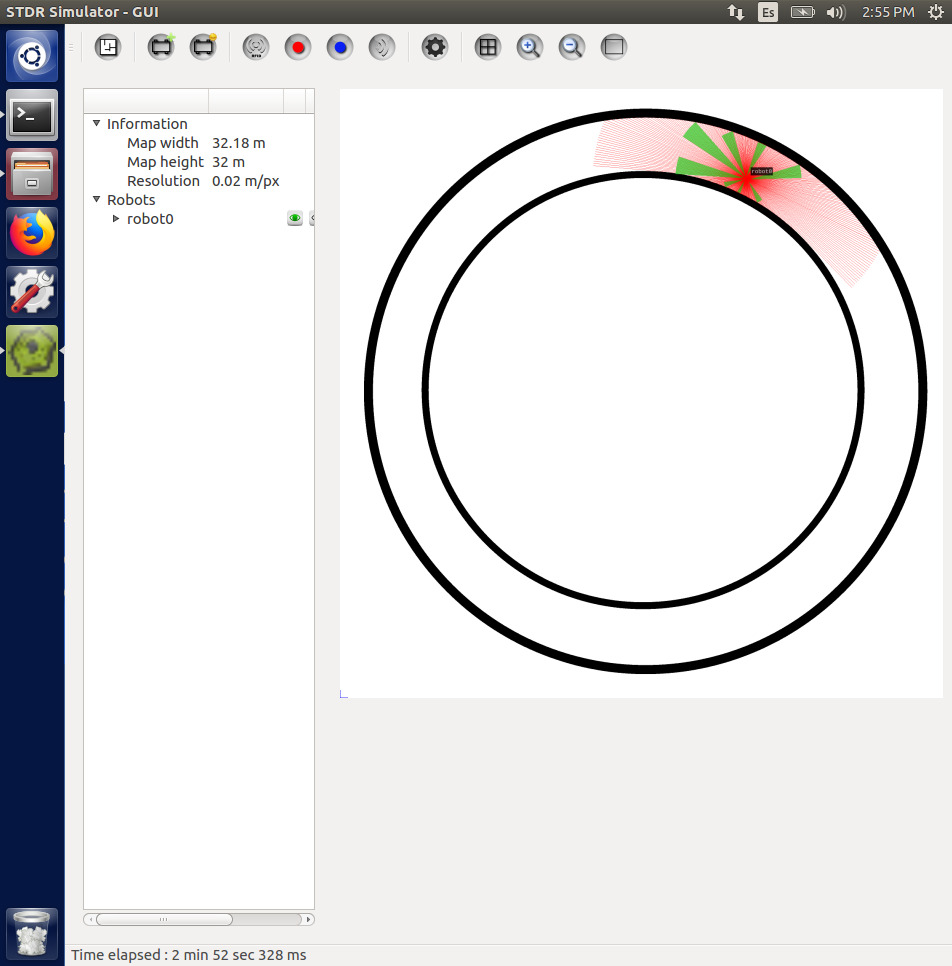


Fig. 24 Robot avanzando por circuito sin obstáculos.

## 3. Conclusión

En conclusión hemos podido ver el comportamiento de distintos controladores borrosos, el Mamdani y Sugeno con neuroborroso. Al hacer esto hemos visto como Mamdani se comportaba mejor en el circuito que el Sugeno con neuroborroso. Esto se debe principalmente a que los circuitos eran sencillos y se conocía el problema en detalle, mientras que para el neuroborroso solo se tenían hasta 1500 registros de trayectorias. Esto nos da la idea de que hay escenarios en los que según el problema sea mejor aplicar uno u otro, por ejemplo si tuviéramos muchos datos y no conociéramos el problema seguramente sería mejor aplicar un controlador neuroborroso.

Otro aspecto positivo que sacamos del proyecto es el uso de ROS para simular el comportamiento de robots, que permite emular distintos escenarios junto a distintos robots con comportamientos reales, que además puede conectarse con Matlab. Haciendo que sea una gran herramienta de trabajo para el diseño de robots y controladores.

## 4. Bibliografía

* <https://es.mathworks.com/help/fuzzy/types-of-fuzzy-inference-systems.html>
* <https://es.mathworks.com/help/fuzzy/types-of-fuzzy-inference-systems.html#mw_adc36736-5ae0-4e1e-bb51-db4c125a81aa>
* <https://es.mathworks.com/help/fuzzy/types-of-fuzzy-inference-systems.html#mw_d7634be8-9ef9-4dae-b503-a3643a5f994b>
* <https://es.mathworks.com/help/fuzzy/tune-fuzzy-inference-systems.html>
* <https://es.mathworks.com/help/fuzzy/tune-mamdani-fuzzy-inference-system.html>
* <https://es.mathworks.com/help/fuzzy/neuro-adaptive-learning-and-anfis.html>
* <https://es.mathworks.com/help/fuzzy/defuzzification-methods.html>

1. Center of area (Centro de areas) [↑](#footnote-ref-1)
2. Center of sums (Centro de sumas) [↑](#footnote-ref-2)