




17 DE OCTUBRE DE 2021

# PRÁCTICA 1

AGENTES INTELIGENTES

NICOLÁS ESPIÑEIRA GARCÍA Y JOSE MIGUEL BURÉS AMATRIAIN

GRADO ROBÓTICA  
USC



1. Introducción .....	2
2. Diseño de maquina finita de estados .....	2
Configuraciones previas .....	2
Recto Giros de 90º .....	2
Sigue Derecha Giros de 90º.....	3
Versión final .....	4
3. Implementación de maquina finita de estados.....	5
4. Análisis de los resultados .....	6
5. Conclusión .....	7

## 1. Introducción

En esta práctica se abordará el desarrollo de un comportamiento basado en máquinas finitas de estado para la resolución de laberintos, utilizando como plataforma robótica un robot dado el cual dispone de un escáner laser de 180°.

En los primeros apartados se explicará el desarrollo de la máquina finita de estados que se utiliza, así como los modelos anteriores que se desecharon por no ser lo suficientemente robustos o por contener estados redundantes.

Se explicará también la implementación de dicho modelo en ROS y como se han utilizado los sensores y las distintas herramientas que nos brinda el robot para optimizar su comportamiento tanto en la simulación como en un supuesto entorno real.

En los apartados posteriores se analizará los resultados del modelo de estados tanto en el laberinto proporcionado por el profesor en la práctica como en los distintos laberintos diseñados para añadirle cierto grado de dificultad y comprobar la robustez del comportamiento en todo tipo de situaciones adversas para el robot.

Por último, se dará una conclusión de la práctica analizando de manera crítica la máquina finita de estados, mostrando sus puntos positivos y negativos e intentado dar soluciones para estos puntos negativos.

## 2. Diseño de máquina finita de estados

### Configuraciones previas

Antes de llegar a una configuración definitiva se probaron varias máquinas finitas de estado variando los estados que se toman y la información sensorial que provoca los cambios de estados:

#### Recto Giros de 90°

En esta máquina se tomaban como información sensorial las medidas de la derecha, de delante y de la izquierda del robot, llamaremos a estas medidas  $dr$ ,  $dl$  y  $iz$  respetivamente. En caso de que haya una pared (distancia tomada es menor que 2 metros) el valor correspondiente será igual a 1 en caso contrario igual a 0. Se resume entonces la información sensorial en una tripla binaria =  $(dl, dr, iz)$

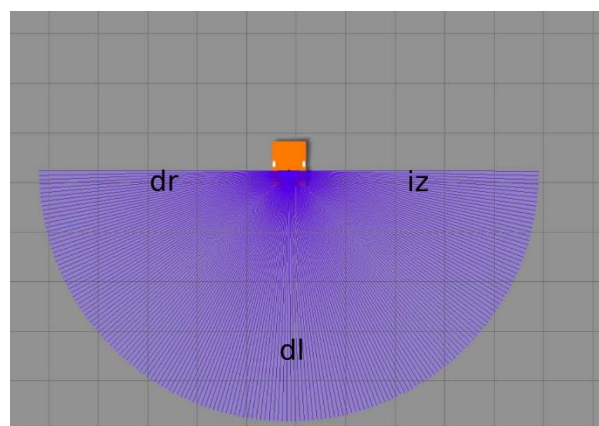


Ilustración 1 - Información sensorial

Los estados de esta máquina serán tres, *recto* (el robot continua recto), *derecha* (el robot gira 90° a la derecha) e *izquierda* (el robot gira 90° a la izquierda). El estado inicial es *recto* en caso de que se encuentre con un obstáculo en frente comprobará si puede ir hacia la derecha, si puede realizará el giro, en caso contrario realizará el mismo procedimiento para

la izquierda lo hará hacia la izquierda. En caso de no haber obstáculos se vuelve al estado inicial y el robot continua recto, en caso de haberlos a ambos lados el robot no es capaz de continuar.

Por lo que obtenemos la siguiente máquina de estados finita (los cambios de estado vienen provocados por los valores de la tripla sensorial, en caso de que algún valor no sea relevante se marca con una X):

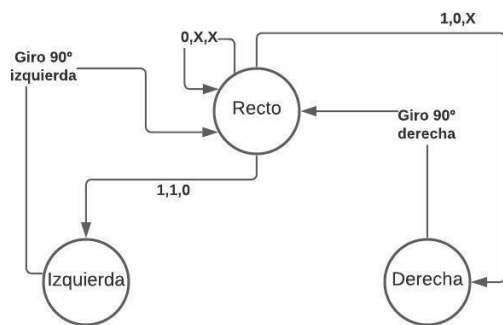


Ilustración 2 - Esquema MFE V1

Esta máquina tiene como ventaja su fácil implementación, sin embargo, tiene dos grandes inconvenientes: el primero, en caso de encontrarse obstáculos a derecha e izquierda el robot se para. El segundo, en muchos casos no resuelve el laberinto, ya que solo va a girar a la derecha si tiene un obstáculo delante, lo que deja muchos caminos por explorar por lo que muchas veces no resuelve el laberinto.

[Sigue Derecha Giros de 90°](#)

Teniendo en cuenta que una de las maneras de resolver un laberinto es seguir la pared pegada a la derecha se intenta implementar un comportamiento que siguiera la pared de la derecha.

En esta máquina se toma como información sensorial las medidas de la derecha y de delante del robot (ya no se toma la izquierda puesto que la prioridad va a ser siempre ir a la derecha), llamaremos a estas medidas *dr* y *dl* respetivamente. En caso de que haya una pared el valor correspondiente será igual a 1, en caso contrario igual a 0. Se resume entonces, la información sensorial en una tupla binaria = (*dr*, *dl*).

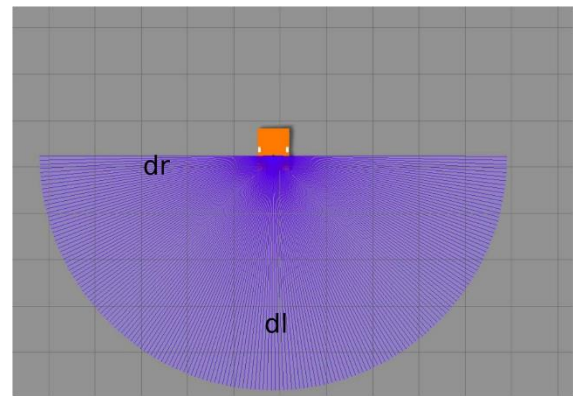


Ilustración 3 - Información sensorial

Tenemos los mismos estados que la anterior máquina: *recto* (el robot continúa recto), *derecha* (el robot gira 90° a la derecha) e *izquierda* (el robot gira 90° a la izquierda). En esta máquina se supone que el robot va a tener una pared al lado desde el comienzo. El estado inicial es *recto* mientras no exista la pared de la derecha o no haya obstáculos delante. En caso de dejar de haber pared en la derecha el robot gira 90° grados a la derecha, de esta manera vuelve a tener una pared a la derecha y puede continuar recto. En caso de seguir habiendo pared a la derecha y existir también pared de frente el robot gira 90° a la izquierda, de esta manera la pared que antes era de frente ahora es la de la derecha y puede continuar siguiendo su derecha.

Obtenemos la siguiente máquina de estados finita:

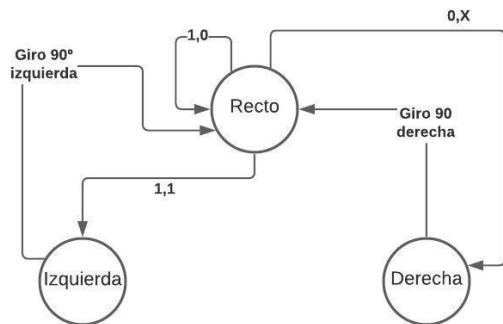


Ilustración 4 - Esquema MFE V2

Esta máquina tiene como ventajas su fácil implementación y que debería ser capaz de resolver casi todos los laberintos. Sin embargo, presenta errores: en primer lugar, en caso de no empezar pegado a la derecha no es capaz de avanzar, en otro lugar, si se encuentra con un mapa como el de la imagen inferior no sigue la derecha por lo que no cumple la premisa principal.

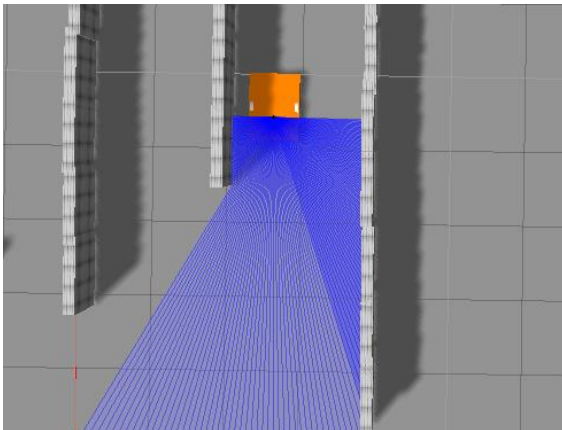


Ilustración 5 - Mapa de error

### Versión final

La versión final se basa en gran parte en la anterior máquina de estados, realizando pequeños cambios. La información sensorial es la misma que la

anterior máquina: la tulpia binaria =  $(dr, dl)$

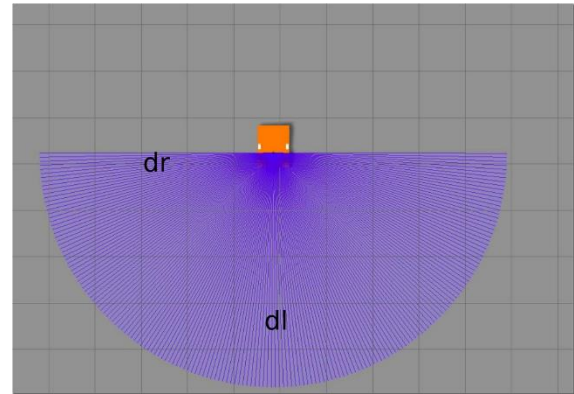


Ilustración 6 - Información sensorial

Los estados de esta máquina serán tres, *recto* (el robot continua recto), *derecha* (el robot gira a la derecha mientras avanza) e *izquierda* (el robot gira a la izquierda). Los estados son similares a la anterior máquina, con la diferencia de que ya no se utilizan giros de 90°. El estado inicial es girar a la derecha, es por ello que la primera comprobación es comprobar si hay pared a la derecha en caso de no haberla girará a la derecha mientras avanza (la magnitud del giro a la derecha se explica en la parte de la implementación), si, por lo contrario, hay pared a la derecha irá recto. Continuará yendo recto mientras no haya posibilidad de ir a la derecha ni obstáculos delante, si hay un obstáculo delante y otro en frente girará hacia la izquierda hasta que el de frente desaparezca (acaba ocurriendo siempre, puesto que si tiene un obstáculo delante y otro a la derecha está en una esquina).

Se obtiene la siguiente máquina finita de estados:

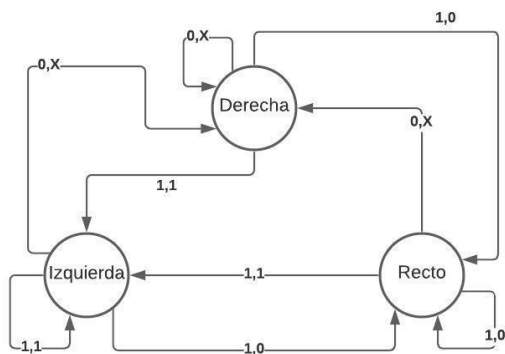


Ilustración 7 - Esquema MFE final

Al eliminar los giros de 90º desaparece el problema del mapa expuesto arriba, simultáneamente también se obtiene un comportamiento más afín con la realidad puesto que se toman medidas todo el rato y no se deja al robot realizando una acción sin feedback (giro de 90º). Por otra parte, al programar que cuando el estado es *derecha* el robot también avanza solventamos el problema de no estar situado al lado de una pared desde el inicio.

### 3. Implementación de maquina finita de estados

Para implementar la máquina finita de estados se ha optado por utilizar la librería Smach, la cual utiliza la programación orientada a objetos para permitir crear máquinas de estados finitas dentro de ROS.

El láser del robot trabaja con 180 valores, y cubre 180º por lo que cada valor cubre aproximadamente un grado, se decide que el valor de *dr* va a venir determinado por los 45º situados a la derecha del robot, en caso de alguno de esos valores ser menor que 1,4 se

considera que hay pared, por lo que  $dr = 1$ . De la misma manera *dl* viene condicionado por los 20º situados en frente del robot, y si alguno de estos valores es menor que 1.85 entonces habrá pared en delante y  $dl = 1$ . Asimismo si todos los valores son mayores se considera que no hay pared. En el programa *dr* y *dl* se almacenan en una lista *medidas*.

Smach permite crear una clase para cada estado. En cada clase se determina que se debe ejecutar mientras se está en el estado y cuáles son las condiciones de transición. Para el estado *recto* se ha fijado una velocidad lineal baja y una velocidad angular nula (el robot avanza hacia delante). Para el estado *izquierda* se fija una velocidad angular baja y una velocidad lineal nula (el robot da vueltas hasta que se cambia de estado). Para el estado *derecha* se fija una velocidad lineal baja y la velocidad angular (potencia de giro) viene determinada por el grado a la que encuentra la medida más próxima, es decir, en caso de que la pared más próxima esté localizada en 45º el giro será poco pronunciado, mientras que si la pared más próxima se localiza en 0º el giro será mucho más potente.

Las condiciones de transición se escriben igual que en el esquema de la máquina finita de estados.

El programa seguirá funcionando siempre que exista alguna medida más baja que 10 (mientras se localicen paredes).

Se utiliza la librería *time* para medir el tiempo que el robot tarda en resolver el laberinto.

#### 4. Análisis de los resultados

Para evaluar el comportamiento de nuestro robot hemos diseñado 2 laberintos más, a parte del ya proporcionado por el profesor. Uno de ellos sería un laberinto de “nivel medio” y el otro sería de “nivel difícil”.

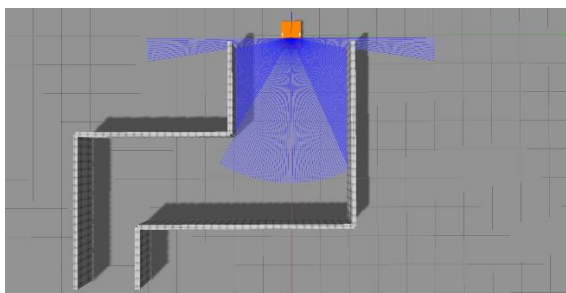


Ilustración 8 - Laberinto básico

En la ilustración 8 se muestra el laberinto básico proporcionado por el profesor, las dificultades que encuentra el modelo son prácticamente nulas exceptuando la parte final del laberinto que se complica por ser muy estrecha. Este laberinto podría haber sido resuelto por cualquiera de las máquinas finitas de estado que diseñamos ya que es muy sencillo.

El robot realiza el laberinto perfectamente, sin chocarse y sin cometer errores, cambiando de estado cuando es necesario, sin arriesgar mucho y acercarse de forma excesiva a las paredes. El robot hace el laberinto en una media de 43 segundos.

Tiempos laberinto básico (s)	
1-	43.31
2-	44.12
3-	43.53
4-	42.89
5-	43.92
Media-	43.55

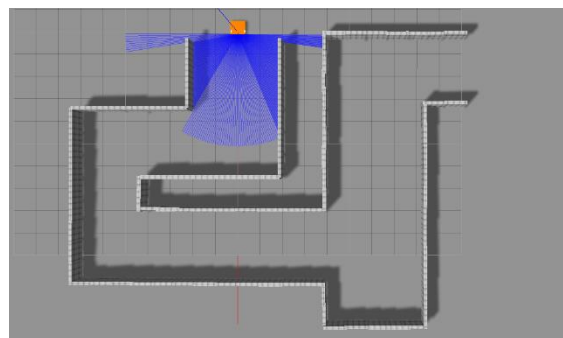


Ilustración 9- Laberinto nivel medio

En la ilustración 9 se muestra el laberinto diseñado de nivel medio, el comienzo es exactamente igual que el laberinto de nivel básico, pero se introduce un grado de dificultad mayor llegado un punto donde el camino se divide en dos posibles vías entre las cuales el robot deberá elegir, como nuestro robot prioriza el giro a la derecha irá por el camino sin salida y deberá modificar su trayectoria para volver al camino correcto.

El robot resuelve el laberinto muy fácilmente, sin choques, sin puntos conflictivos y fluctuando entre estados a la perfección en una media de 128 segundos. Resuelve también la bifurcación con mucha facilidad, escogiendo el camino de la derecha para detectar que no puede avanzar por esa vía y volver a la trayectoria correcta.

Tiempos laberinto medio (s)	
1-	130.27
2-	126.39
3-	128.03
4-	128.59
5-	129.17
Media-	128.49



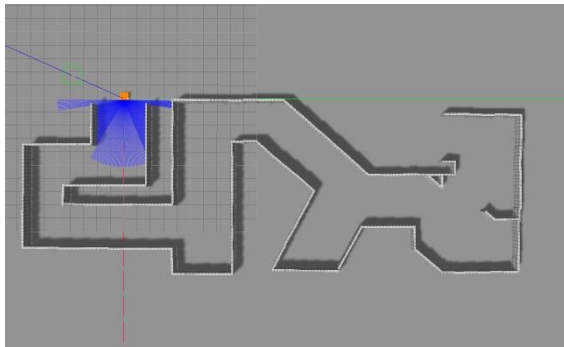


Ilustración 10 - Laberinto de nivel difícil

En la ilustración 10 se muestra el laberinto de nivel difícil, que comienza igual que el laberinto de nivel medio, pero en este la dificultad es mayor ya que se introducen paredes con ángulos que no son rectos y obstáculos en ambas paredes que el robot tendrá que sortear. El objetivo de este diseño es ver el comportamiento del robot frente a trayectorias diferentes a ángulos de  $90^\circ$  y su capacidad de sortear obstáculos y volver a su trayectoria habitual.

El robot hace el laberinto de nivel difícil en una media de 290 segundos con relativa facilidad y sin muchas complicaciones exceptuando un punto crítico en el cual hace muchas maniobras en algunas simulaciones excesivas.

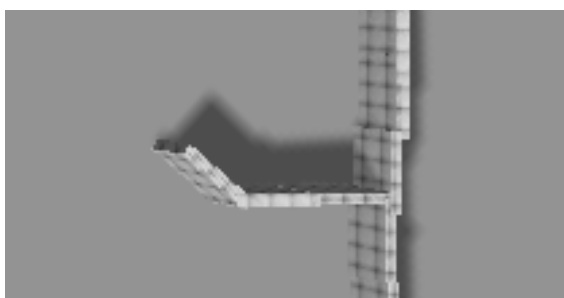


Ilustración 11 - Punto crítico en el laberinto difícil

El punto crítico, que simula un obstáculo pegado a una pared, se da en un punto en el que el robot pasa al estado de giro a la derecha al detectar la pared que está rotada  $45^\circ$  y comienza a girar a la

derecha para luego detectar la pared muy cerca y pasar al estado girar a la izquierda y viceversa. Aunque este error se produce en un bajo porcentaje de las simulaciones es un factor que hay que tener en cuenta a la hora de intentar optimizar nuestro comportamiento.

Tiempos laberinto difícil (s)	
1-	286.45
2-	294.50
3-	288.63
4-	293.62
5-	290.77
Media-	290.79

## 5. Conclusión

Como conclusión, destacar el hecho de que se ha priorizado un comportamiento robusto frente a la velocidad de resolución de laberintos, así como evitando los giros de  $90^\circ$  y tomando medidas constantes variando de estado varias veces por segundo permite que el robot se adapte a todo tipo de giros y de laberintos, pudiendo llevar su comportamiento incluso a la vida real manteniendo su robustez.

También mencionar, que debido a que no tomamos medidas en exceso del escáner láser optimizamos el funcionamiento tanto de la simulación como de un supuesto robot real, rebajando la presión sufrida por el procesador.



