USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA
ESCOLA POLITÉCNICA SUPERIOR DE ENXEÑARÍA

Memoria práctica 1
Agentes Inteligentes

Adrián Losada Álvarez

Tabla de contenido

1. Introducción

En este documento se explicará de forma detallada el proceso para la realización de la primera práctica de la materia agentes inteligentes.

El objetivo de esta se basa en el juego infantil *"Hide and Seek"*, en español, el escondite. Por si aún no se conocen las normas básicas de este juego, será descrito a continuación:
- Una persona (robot en nuestro caso) tendrá el papel de *Seeker*, es decir, su objetivo será encontrar al *Hider*, que deberá esconderse de él en todo momento.
- El juego termina cuando el *Seeker* encuentra al *Hider*.

Dejando atrás la parte del juego comenzaremos a hablar de la resolución de la práctica propuesta, para ello se diseñó una maquina finita de estados (FSM) para el movimiento de cada robot. Una máquina de estados es una herramienta conceptual para modelar comportamientos reactivos mediante estados y transiciones, éstas se generan en respuesta a eventos de entrada externos e internos y sirven como condición para cambiar de un estado a otro. Las implementaremos en ROS utilizando Python como lenguaje de programación dada su sencillez.

Uno de los principales problemas de esta práctica es la filtración de datos que recibe el robot *seeker* para lograr diferenciar entre una pared y el robot *hider*. Para superar este problema hasta un cierto límite nos podemos ayudar de la estadística, más en concretamente de la desviación típica *(se explicará en detalle más adelante).*

Las entradas y salidas de la máquina de estados dependerán de cada robot:

Seeker
Entradas
- Valor mínimo detectado por las mediciones del LiDAR.
- Desviación típica de las 10 medidas alrededor del mínimo detectado.
Salidas
- Estilos de movimiento programados (*'wandering', 'seeking', 'success'*).

Hider
Entradas
- Callback del *seeker*.
Salidas
- Estilos de movimiento programados (*'wandering', 'stop'*).

En cuanto al entorno a desarrollar la práctica se nos proporciona un mapa inicial el cual optamos por cerrar por completo para evitar que los robots se alejaran demasiado y a partir de éste lo modificamos con distintos obstáculos para observar los posibles resultados.

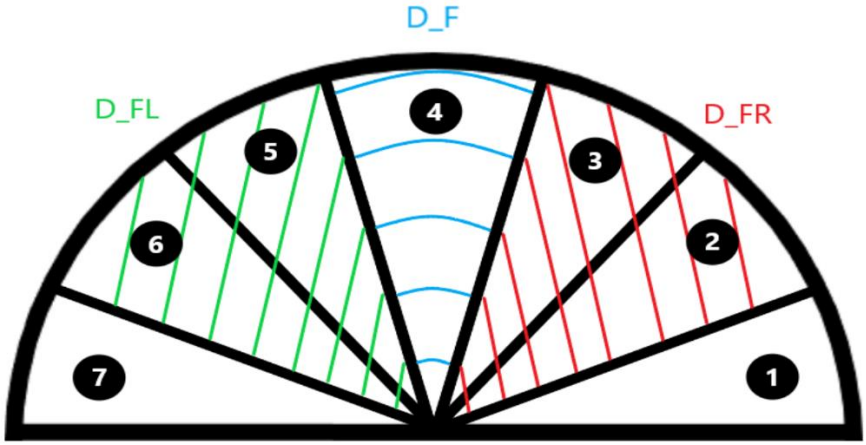
Para finalizar, sacaremos algunas conclusiones a partir de los resultados obtenidos, así como algunas posibles mejoras a implementar.

2. Aclaraciones previas

Antes de comenzar con el análisis de los datos, conviene aclarar que los robots están programados para evitar colisiones con los elementos del mapa, aún así pueden producirse algunas colisiones entre los robots debido a fluctuaciones en las mediciones de los LiDAR.

- **Manejo de datos LiDAR**

Seccionaremos el arco de 180º del escáner del robot en 7 segmentos, de los cuales agruparemos de la siguiente manera:



En donde D_FL, D_F y D_FR serán las distancias mínimas obtenidas de los segmentos: frontal izquierdo (segmentos: 6+5), frontal (segmento 4) y frontal derecho (segmentos 3+2) respectivamente.
Estos datos serán utilizados en las funciones creadas para controlar el movimiento de los robots.

- **Código**

Para alcanzar un movimiento fluido y a la vez seguro, la condición de avance y de giro no dependen una de la otra, es decir, si la distancia frontal recibida del LiDAR es menor a un cierto umbral el robot se detendrá y girará sobre si mismo, mientras que si se dispone de distancia frontal suficiente girará mientras avanza, a su vez el robot no hará giros si no es necesario, continuará recto. La lógica está explicada con más detalle en los comentarios del código. Toda la lógica de los movimientos está en el script '*movementsClass.py*'.

Se trata de un código general para todos los mapas, por lo que pueden surgir algunos problemas en aquellos que tengan algún tipo de elemento en específico.

En cuanto a la posición de inicio de los robots, se les ha girado 180º a ambos para que empiecen de espaldas en lugar de ir uno contra el otro al iniciar el movimiento.

Por último, cabe aclarar que los valores de los umbrales tanto de DIST_MIN como de NEARBY_STD_D para las salidas de las máquinas de estados y las velocidades de avance y giro han sido seleccionadas por métodos experimentales mediante el método de prueba y error.

3. Maquina finita de estados

A continuación, veremos cómo hemos implementado las máquinas finitas de estados para cada robot en detalle.

Tablas de entradas y salidas:

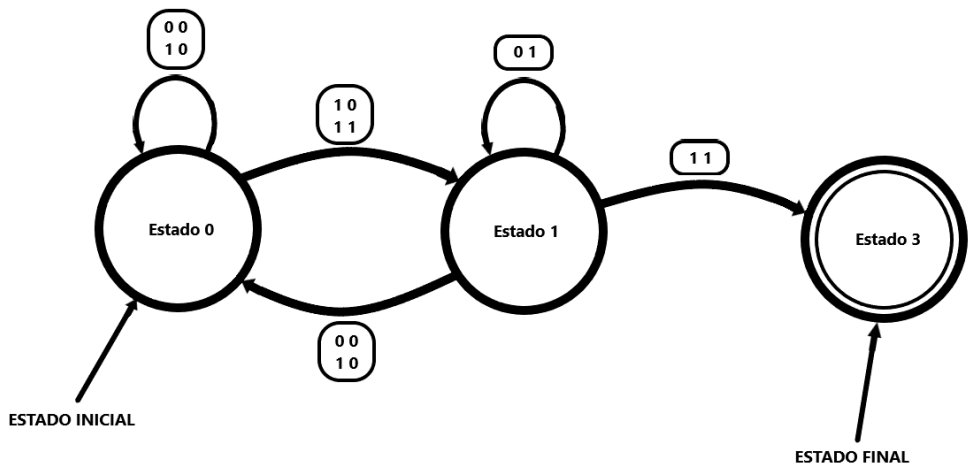
Seeker	
Entradas	**Salidas**
Desviación típica	*'wandering'*
Distancia mínima	*'seeking'*
	'success'

Hider	
Entradas	**Salidas**
Callback seeker	*'wandering'*
	'stop'

- SEEKER

1. Diagrama de estados

La máquina desarrollada para el robot *Seeker* cuenta con tres estados, dispuestos de la siguiente forma:



En el diagrama se determinan las condiciones de la siguiente manera:

Distancia mínima (MIN_SCAN)	Desviación típica (NEARBY_STD_D)
0	0
0	1
1	0
1	1

Distancia mínima -> 0: MIN_SCAN > Umbral (1.5)
1: MIN_SCAN <= Umbral

Desviación típica -> 0: NEARBY_STD_D => Umbral (0.08)
1: NEARBY_STD_D < Umbral

Como ya se mencionó anteriormente, el valor de la variable MIN_SCAN es el valor mínimo obtenido en los 180º del LiDAR. Mientras que NEARBY_STD_D se obtiene de calcular la

desviación estándar desde los 5 valores a la izquierda hasta los 5 valores a la derecha del punto más próximo detectado (MIN_SCAN), para así poder ver como varían los datos ya que si se trata de una pared el resultado de esta operación es un valor bajo mientras que, si el robot *Hider* se encuentra en el este rango, la operación dará unos valores mayores.
Este método para encontrar al robot da falsos positivos cuando nos encontramos con una esquina muy pronunciada, se verá con más detalle en el apartado 4. Análisis de resultado.

2. Estados

Explicación detallada de cada estado:

- **Estado 0:**

  Se trata del estado '*wandering*' (estado inicial) en el cual el robot deambulará por el mapa evitando obstáculos y buscando al *Hider*, la lógica es la siguiente: el robot avanzará si el valor de la distancia mínima del arco frontal (*D_F*) es mayor al umbral definido (*THRESHOLD* = 2.0), de no ser así el robot no avanzará, para la condición de giro si la distancia mínima del arco frontal izquierdo (*D_FL*) es mayor que la del arco frontal derecho (*D_FR*) éste girará a la izquierda, si es al revés, girará a la derecha y si son iguales, no girará.

- **Estado 1:**

  Es el estado 'seeking' en el que el *Seeker* ha detectado al otro robot en su rango de visión y procede a perseguirlo, la lógica es la siguiente: el robot avanza siempre a una velocidad constante y la condición de giro es la opuesta a la del estado 0 ('*wandering*') ahora el robot girará hacia el lado que detecte la menor distancia, es decir, si se detecta una distancia más pequeña en el lado izquierdo (*D_FL*), girará a la izquierda, en caso contrario, a la derecha.

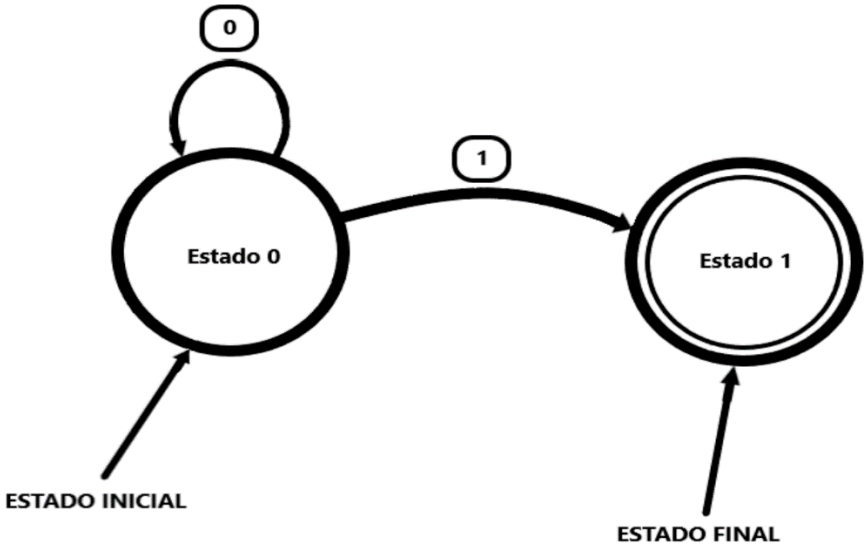
- **Estado 2:**

  Es el estado '*success*' y significa que el juego a terminado ya que el *Hider* a sido encontrado por lo tanto el robot *Seeker* comenzará a celebrarlo girando sobre sí mismo en sentido antihorario, a su vez se le comunicará al otro robot que ha sido encontrado.

- HIDER

1. Diagrama de estados

La máquina desarrollada para el robot *Hider* cuenta con tres estados, dispuestos de la siguiente forma:



En el diagrama se determinan las condiciones de la siguiente manera:

Callback seeker (data)
0
1

Callback seeker -> 0: data == False
1: data == True

Este diagrama de estados es muy simple ya que solamente cambia del estado inicial al final si el robot *Seeker* se lo indica mediante el topic *'/game'*.

2. Estados

Explicación detallada de cada estado:

- **Estado 0:**

  Se trata del estado '*wandering*' (estado inicial) en el cual el robot deambulará por el mapa evitando obstáculos y buscando al *Hider*, la lógica es la siguiente: el robot avanzará si el valor de la distancia mínima del arco frontal (*D_F*) es mayor al umbral definido (*THRESHOLD* = 2.0), de no ser así el robot no avanzará, para la condición de giro si la distancia mínima del arco frontal izquierdo (*D_FL*) es mayor que la del arco frontal derecho (*D_FR*) éste girará a la izquierda, si es al revés, girará a la derecha y si son iguales, no girará.

- **Estado 1:**

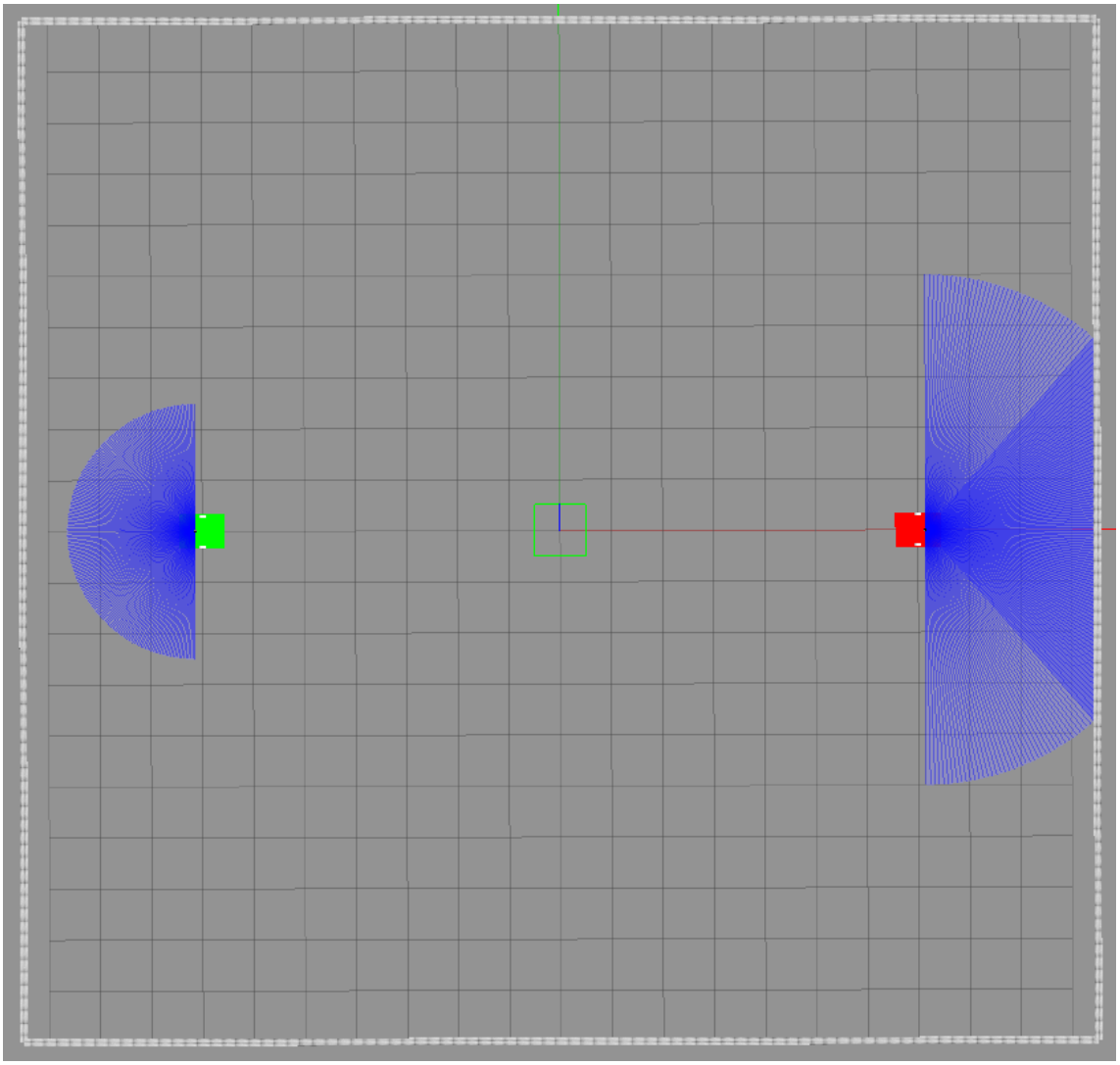
  Es el estado '*stop*' significa que el *Seeker* lo ha encontrado y debe parar su movimiento.

4. Análisis de resultados

A continuación, se probarán los comportamientos de los robots en los distintos mapas implementados.

- box.world

Mapa inicial, pero cerrando el mapa para evitar que los robots se alejen demasiado.



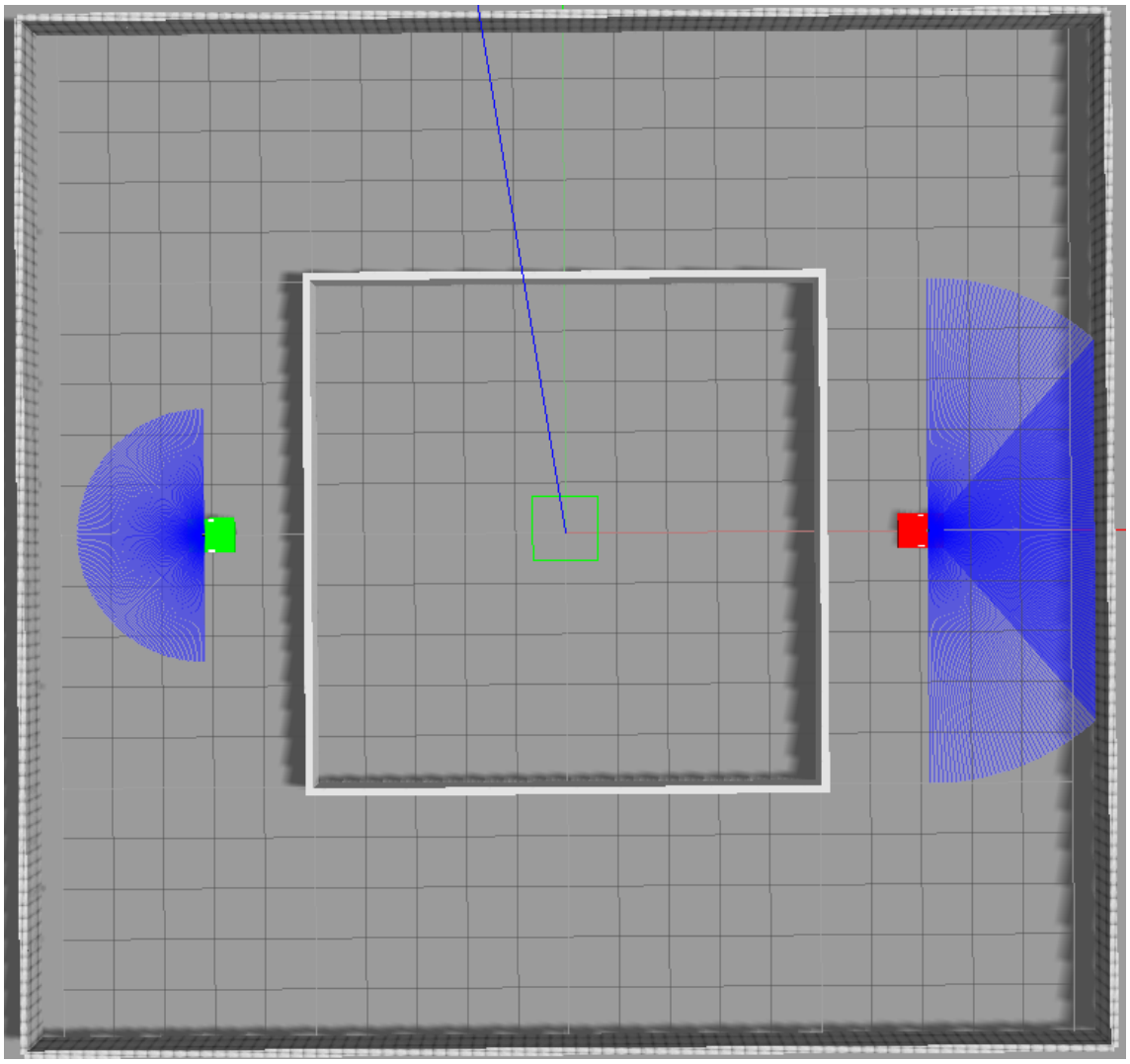
Al no tener ningún tipo de obstáculo el robot *Seeker* no tiene problemas a la hora de navegar por el mapa y encontrar al *Hider* nunca.

Resultados:

Prueba	Resultado	Tiempo (seg)
1	Éxito	56,99431753158569
2	Éxito	61,74492335319519
3	Éxito	60,98682117462158
4	Éxito	58,18912651178915
5	Éxito	60,23187124517152
Media	Éxito	59,629411963273

- big_square.world

Ligera modificación del mapa inicial creando un “pasillo” con algunas esquinas.



El robot *Seeker* presenta algunos problemas a la hora de moverse por el entorno dependiendo de cómo se aproxime a la esquinas, el tiempo de juego es bastante variable ya que a veces los robots avanzan en la misma dirección lo que prolonga la captura del robot *Hider*

Resultados:

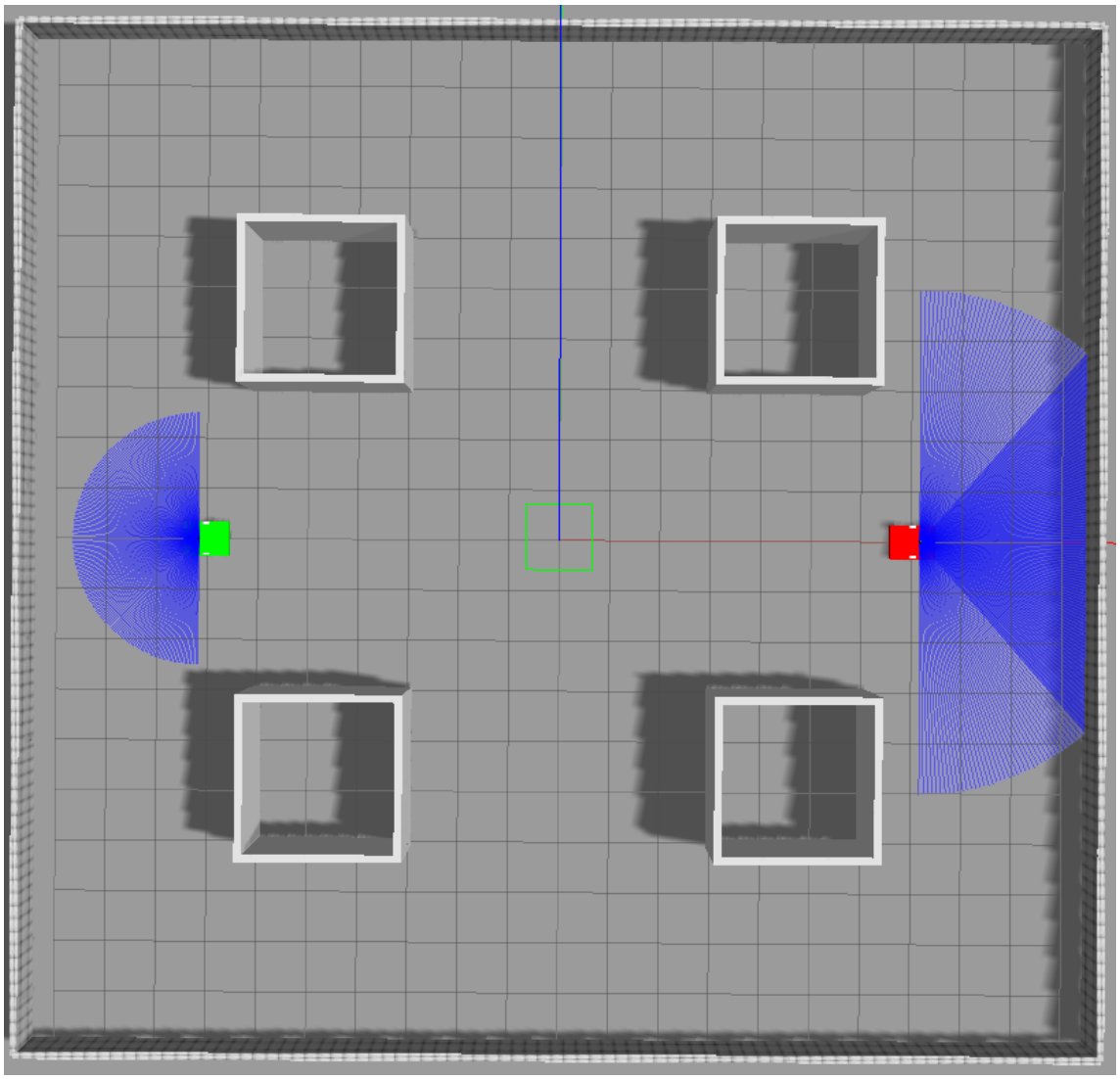
Prueba	Resultado	Tiempo (seg)
1	Fracaso	40,2154325234532
2	Fracaso	55,9816217851291
3	Éxito	144,329123248613
4	Éxito	133,8127818618162
5	Éxito	87,0247548716891
Media	Éxito	121,72221999404

Nota: Como hay mayor número de intentos exitosos que fallidos se asume que la media de los resultados es exitosa.

Nota: Para la media de tiempo solo se toman los valores de aquellas pruebas con resultado exitoso.

- multiple_squares.world

En lugar de un cuadrado grande, ahora se implementan 4 cuadrados más pequeños y dispersos.



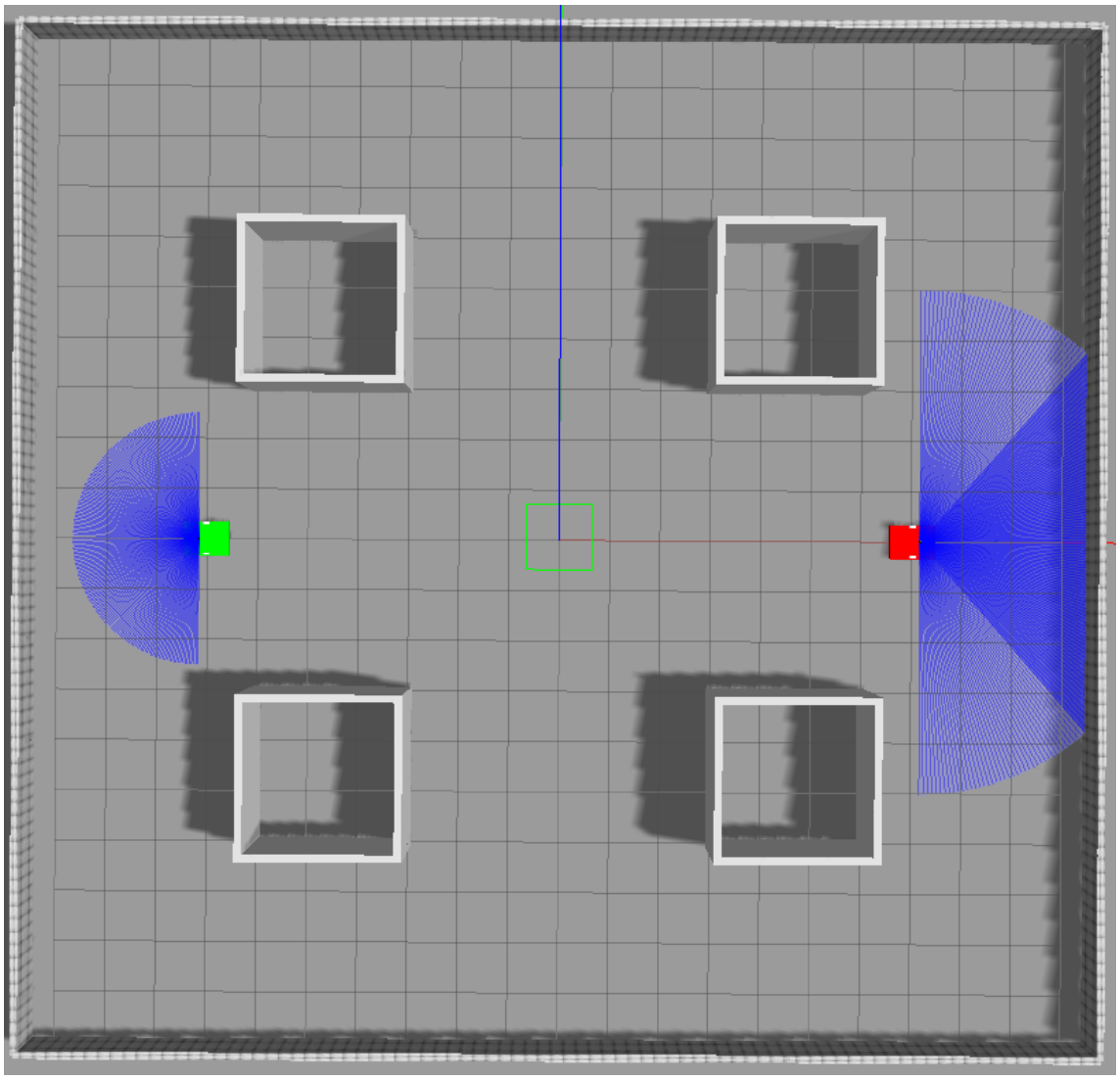
En este caso el robot *Seeker* es incapaz de no confundir una esquina de los cuadrados con el robot *Hider* ya que la probabilidad de que se encuentre con una de las esquinas de los cuadrados es muy alta

Resultados:

Prueba	Resultado	Tiempo (seg)
1	Fracaso	8,942986965179443
2	Fracaso	8,948646545410156
3	Fracaso	8,948767662048348
4	Fracaso	8,944819450378418
5	Fracaso	8,935220241546632
Media	Fracaso	0

- soft_walls.world

Por último, se creó un mapa con varias estructuras aleatorias con distintos ángulos en sus esquinas.



Al tratarse de un mapa con estructuras con esquinas “suaves” el robot *Seeker* es capaz de encontrar al robot *Hider* sin problemas.

Resultados:

Prueba	Resultado	Tiempo (seg)
1	Éxito	44,96626448631286
2	Éxito	49,74005174636841
3	Éxito	58,99505877494812
4	Éxito	35,25031065940857
5	Éxito	44,47800302505493
Media	Éxito	46,685937738419

5. Conclusión

Para finalizar, expondremos las conclusiones que hemos obtenido de los resultados anteriores.

Podemos observar que la máquina de estados diseñada cumple con su propósito, pero tiene algún que otro defecto, como son las esquinas, los bordes de las paredes, etc. Para mejorarla se podrían realizar algunos cambios, pero dado a su posible complejidad, no fueron implementados.

Problema	**Resolución Actual**	**Posible Solución**
Diferenciación esquina/robot	Desviación típica alrededor del mínimo	Comprobar si el objeto a detectar se mueve
Mejora del tiempo de captura	Movimiento aleatorio	No repetir caminos en un intervalo dado
Mejorar el escondite	Movimiento aleatorio	Localizador de rincón

Para el primer problema se podría implementar un script de tal manera que, si se detecta un objeto dentro de una distancia umbral, el robot *Seeker* se detuviera un tiempo dado para tomar un número de medidas y una vez terminado ese período de tiempo analizar la lista de valores obtenidos para comprobar si la distancia mínima ha cambiado o no.

Para el segundo, podríamos almacenar las posiciones por las que se desplaza el robot para después evitar repetir ese camino si ha pasado poco tiempo.

Por último, para el tercer problema se podría implementar un localizador de rincones para que el robot *Hider* se escondiera mejor, una posible solución sería detectar las distancias de su izquierda, frente y derecha, y si todas están por debajo de un umbral, que el robot se detenga en dicha posición.

Con dicha conclusión terminamos la primera práctica de Agentes Inteligentes sabiendo que la máquina finita de estados es una buena opción para plantear y planificar los comportamientos de los robots.