Técnicas de los Sistemas Inteligentes

Práctica 1 - Entrega 2: Navegación local con y sin mapa

José Pimentel Mesones Lothar Soto Palma Francisco David Charte Luque
José Carlos Entrena Jiménez

Resumen

Las técnicas basadas en navegación local pretenden guiar a un agente a lo largo de un mapa de forma que la trayectoria que sigue no está predefinida, es decir, se toman decisiones locales. En esta práctica se ha completado una implementación de un agente reactivo con navegación mediante campos de potencial con un mayor ángulo de visión, campos atractivos y repulsivos y detección de mínimos locales. Asimismo, se ha añadido cierta funcionalidad al agente, cono una gestión de la huida de mínimos locales mediante una selección heurística de objetivos secundarios atendiendo a los mapas de coste.

Índice

1. Tareas para mejorar el comportamiento reactivo del robot respecto al uso de campos de potencial (navegación local sin mapa)

1.a. Conseguir aumentar el campo de visión

El campo de visión del robot simulado es de 270º, ¿cómo hacer para que el escaneo laser sea mayor que el actualmente implementado?

En el archivo *myPlannerLite.h* hemos cambiado el valor de las constantes `MIN_SCAN_ANGLE_RAD` y `MAX_SCAN_ANGLE_RAD` a `-135.0/180*M_PI` y `135.0/180*M_PI` respectivamente, para aumentar el ángulo de visión a un total de 270º.

1.b. Conseguir que no se demore tanto tiempo en detenerse cuando esté lo suficientemente cerca del objetivo

¿Por qué tarda tanto en detenerse cuando está próximo al objetivo? ¿Cómo solucionarlo?

Cuando calculamos la componente atractiva y nos encontramos dentro del campo de atracción (representado por la componente *spread*), el módulo de la velocidad es inversamente proporcional a la distancia al objetivo, por lo que cuanto más nos acercamos a él, más lento se mueve el robot.

Para solucionar este comportamiento, nuestra idea ha sido hacer constante el módulo de la componente atractiva de la velocidad cuando la distancia al objetivo es menor que la mitad del *spread* del objetivo. Así, cuando estamos muy próximos al objetivo, la velocidad es constante y no se va reduciendo conforme nos acercamos, evitando que tarde tanto tiempo alcanzar el objetivo y detenerse.

Tomamos esta velocidad constante como aquella obtenida cuando estamos a una distancia del objetivo de *spread*/2.

1.c. Conseguir que el robot no tenga "comportamiento suicida", es decir, cuando está cerca de un obstáculo acelera y choca con él

¿Cuál es la causa de este comportamiento suicida? ¿Cómo evitarlo?

Al encontrarnos muy próximos a un obstáculo, el cálculo de la componente repulsiva dará como resultado una velocidad lineal muy alta, consecuencia de querer alejarnos cuanto antes de dicho obstáculo. Sin embargo, como tratamos la velocidad lineal y la velocidad angular por separado, al aumentar la velocidad lineal el robot no tiene tiempo de girar antes de chocarse con el obstáculo, dando la sensación de que el robot se lanza hacia él.

Como solución, hemos decidido hacer 0 la velocidad lineal del robot cuando se encuentra con un obstáculo y tiene que girar para evitarlo. De esta forma, permitimos al robot tomar la dirección deseada antes de volver a trasladarse, evitando así lanzarnos hacia el obstáculo. Cuando esto ocurra, tendremos al robot en un estado de giro, que al estar activado inhabilitará el paso de la velocidad lineal calculada, y en su lugar se pasará 0 como la componente `linear` del mensaje.

1.d. Conseguir que, una vez que el robot se queda atrapado en una esquina (en un mínimo local), trate de salir de esta situación

¿Cómo conseguirlo sin utilizar memoria (es decir, información de un mapa)?

En el archivo `myClientLite.cpp` se ha implementado una funcionalidad adicional en la función gestora de eventos de *feedback* que lleva un registro de la última posición y es capaz de analizar si el robot está quieto (o prácticamente quieto, mediante una tolerancia) y, tras transcurrir un tiempo razonable (que permite que el robot pivote sobre sí mismo si es necesario), asume que se ha encontrado un mínimo local.

Cuando detectamos que el robot está atascado, cancelamos el goal actual y enviamos un nuevo goal, para intentar escapar de este mínimo local. Calculamos este nuevo goal utilizando el costmap, intentando dirigirnos hacia la zona que tenga menos obstáculos. En caso de no conseguir alcanzar este segundo goal, cancelamos la acción del robot. Si llegamos hasta él, restauramos el goal inicial.

1.e. Conseguir suprimir las oscilaciones del robot

¿Cuál es la causa de las oscilaciones? ¿Cómo eliminarlas en la mayor medida posible?

Hemos detectado oscilaciones en el robot cuando está muy cerca del objetivo. Esto se debe a que, al intentar encarar el objetivo, el mínimo giro posible hace que nos pasemos y tengamos que girar hacia el otro lado, volviendo a darse la misma situación. Para evitarlo, hemos aumentado la tolerancia en el ángulo del robot, subiendo el valor de la componente `EPSILON_ANGULAR`.

Asimismo, se producen oscilaciones cuando el robot está bordeando una pared y el objetivo se encuentra al otro lado, ya que pretende girar hacia el objetivo pero para avanzar necesita hacerlo paralelamente, ya que la componente repulsiva le impide atravesar la pared. Para estos casos, se ha añadido una constante `MIN_LIN_SPEED_FOR_TURNS` que representa la velocidad lineal mínima que ha de tener el robot para permitirse girar. El valor de la constante se ha escogido experimentalmente, y se ha establecido a 0,05.

2. Tareas para mejorar el comportamiento del robot mediante técnicas de navegación local con mapa.

2.a. Contemplar el uso de distintos mapas para la experimentación

Los mapas pueden generarse a partir de una imagen mediante el paquete mapserver. Ver explicación sobre manejo de mapas y mundos simulados en la documentación adjunta.

Incluímos capturas de pantalla con el uso de distintos mapas sobre los que hemos experimentado.

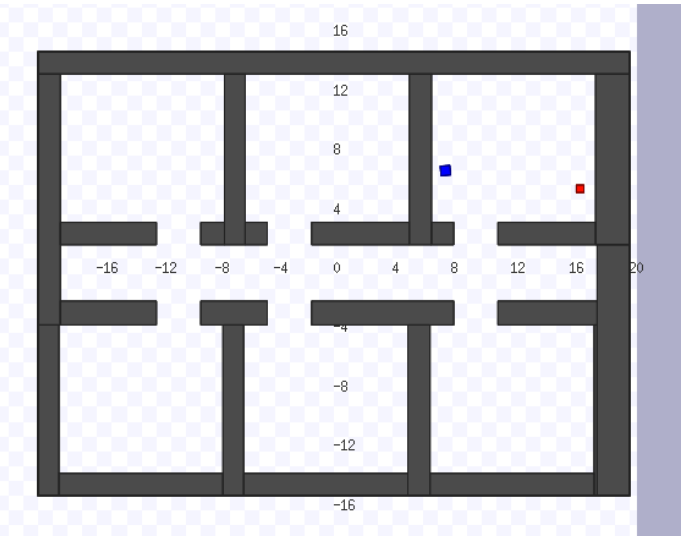

Figura 1: Mapa: Simple rooms

El comportamiento del robot en el mapa *Simple_rooms* se explica junto a la heurística en la sección 2.c. En el caso del mapa *autolab* el principal problema que se puede obtener es que un goal secudario no sea capaz de llegar a completarse con éxito puesto hay un pequeño pasillo a través del cual el robot no es capaz de pasar o pasa muy lentamente dependiendo de la asignación del goal secudario con la heurística como observamos en la figura 3.

En esta prueba se estableció el goal en el punto (-1.9,11.1) de forma que el robot se acerca inicialmente hasta el punto encontrandose con una pared y posteriormente aplicando la heurística programada como se ve en 4 y 5. Despues de aplicarse el robot buscará una nueva ruta asignandose goal secundarios como en 6, una de ellas obliga a pasar por el pasillo anteriormente mencionado, como este nos es capaz dependiendo de donde se coloque el nuevo goal el robot se atasca como ocurria en la figura 3.

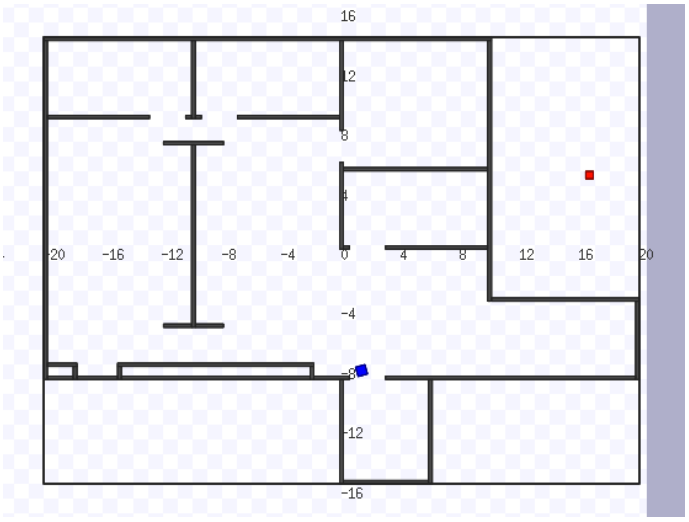

Figura 2: Mapa: autolab

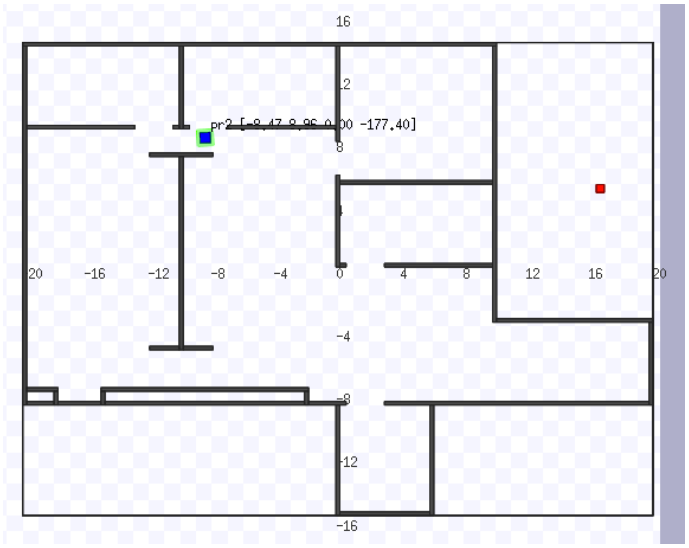

Figura 3: Atasco en goal secundario

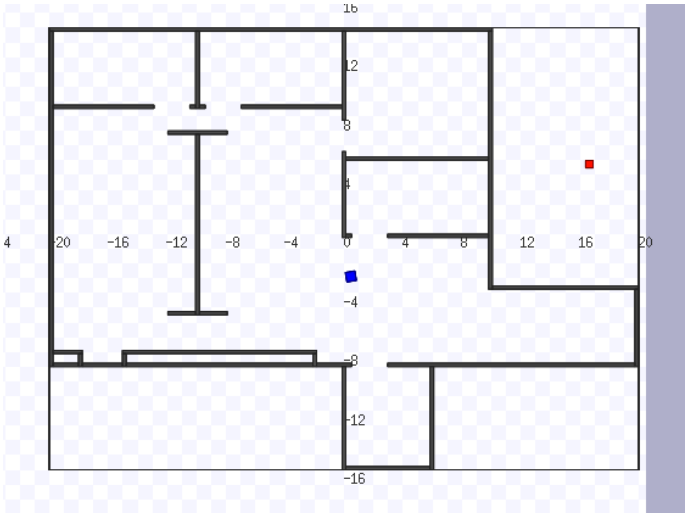

Figura 4: Ida hacia el goal principal

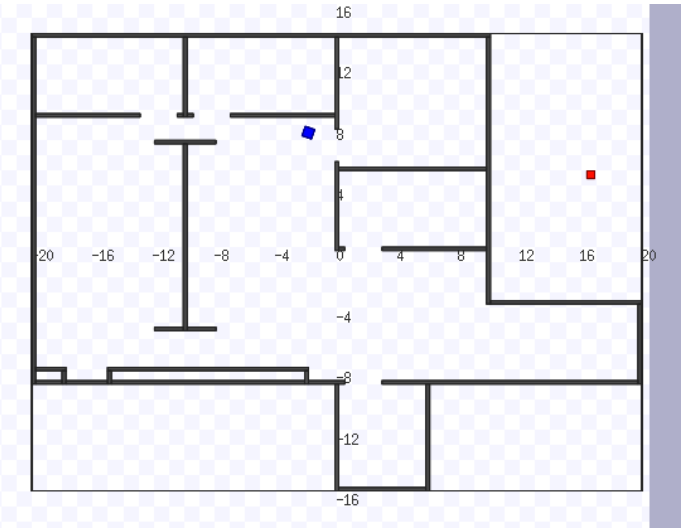

Figura 5: Goal principal no alcanzable

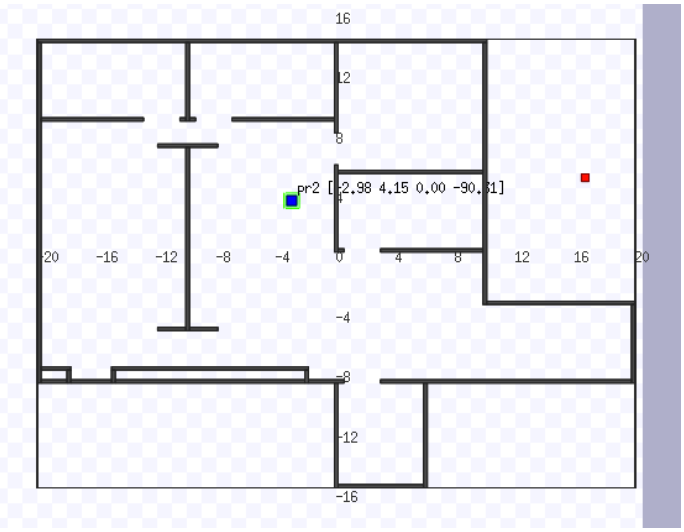

Figura 6: Aplicación de la heurística

2.b. Usar dos costmaps en el cliente

Uno local configurado para tener una ventana activa que se desplace con el robot y otro global para tener una información sobre el costmap del mapa completo.

La visualización y control de los costmaps se lleva a cabo en la herramienta Rviz, y estos nos permiten implementar nuevas políticas de asignación de goals.

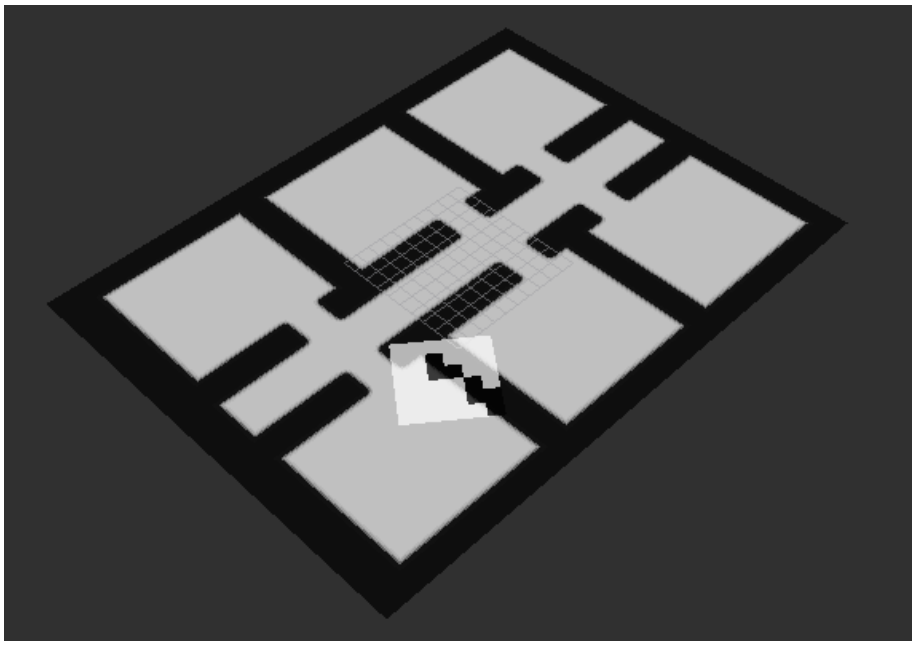

Figura 7: Visualización de los costmaps con Rviz

2.c. Usar el costmap local para poder encontrar una trayectoria local segura desde la pose actual

Se ha desarrollado una heurística sencilla orientada al descubrimiento de trayectorias alternativas hacia el objetivo. La heurística permite que el robot avance en una dirección que esté libre hasta alcanzar un objetivo secundario, y tras esto se puede reevaluar el objetivo secundario o restablecer el primario.

Cuando se realiza una llamada a la heurística, esta evalúa los vecinos libres alrededor de la posición del robot. Cada casilla vecina libre se utiliza como una posible dirección a tomar. Cuando hay más de una, utiliza los mapas de coste para calcular la dirección en la que se pueda alcanzar la distancia más lejana a la actual. Además, se cuenta con un parámetro de ajuste que indica a qué altura del recorrido debería el robot pararse y reevaluar el objetivo.

En la práctica, se consigue que el robot se aleje de la zona del mínimo local en línea recta (horizontal, vertical o diagonal) pero, puesto que el número de objetivos secundarios consecutivos y el parámetro de ajuste van cambiando, la heurística es potente y es capaz de descubrir “salidas” de una zona difícil.

A continuación ilustramos la heurística mediante un ejemplo. Se ha utilizado el mapa *simple_rooms* ubicando el objetivo en el interior de la segunda habitación superior y colocando al robot inicialmente en la tercera habitación. De esta forma, encuentra muy rápidamente un mínimo local al acercarse a la pared que separa ambas habitaciones, como observamos en la figura 8. Se llama entonces a la heurística que calcula un objetivo lejano en línea recta. Tras alcanzarlo, se restablecerá el objetivo inicial, y si de nuevo nos encontramos con un mínimo local, se calcularán entonces dos objetivos secundarios con un recorrido menor. Iterando este proceso, el robot llegará a salir de la habitación, como se muestra en la figura 9, y tratará de explorar otras zonas del mapa.

Si se restablece el objetivo inicial en otro punto, el robot podrá ser capaz de utilizar una nueva trayectoria gracias a los campos de potencial, que le permita llegar a su objetivo primario, como se observa en las figuras 10 y 11.

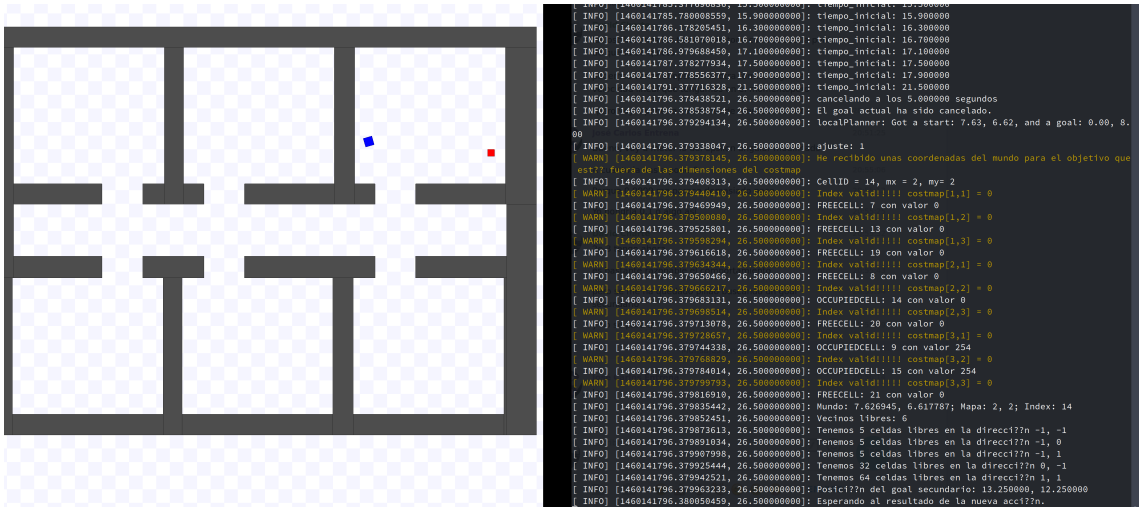

Figura 8: Cancelación de objetivo inicial y cálculo de un objetivo secundario

Figura 9: El robot sale de la habitación gracias a la heurística

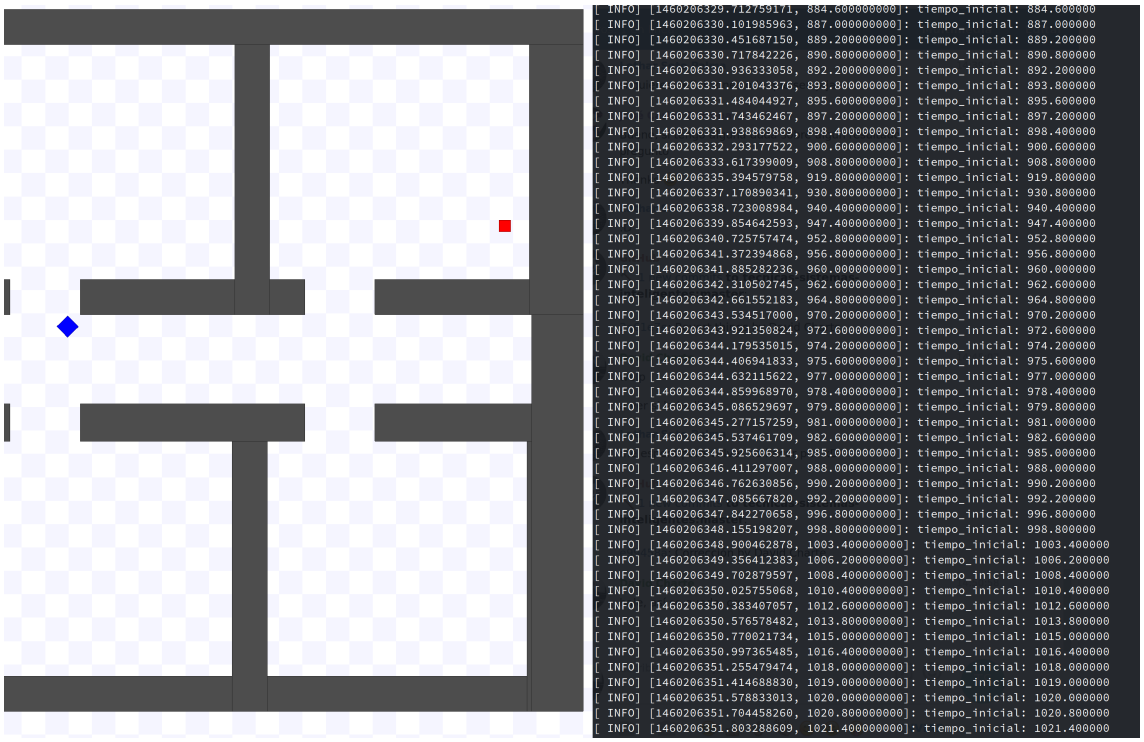

Figura 10: Se restablece el objetivo inicial tras varios secundarios

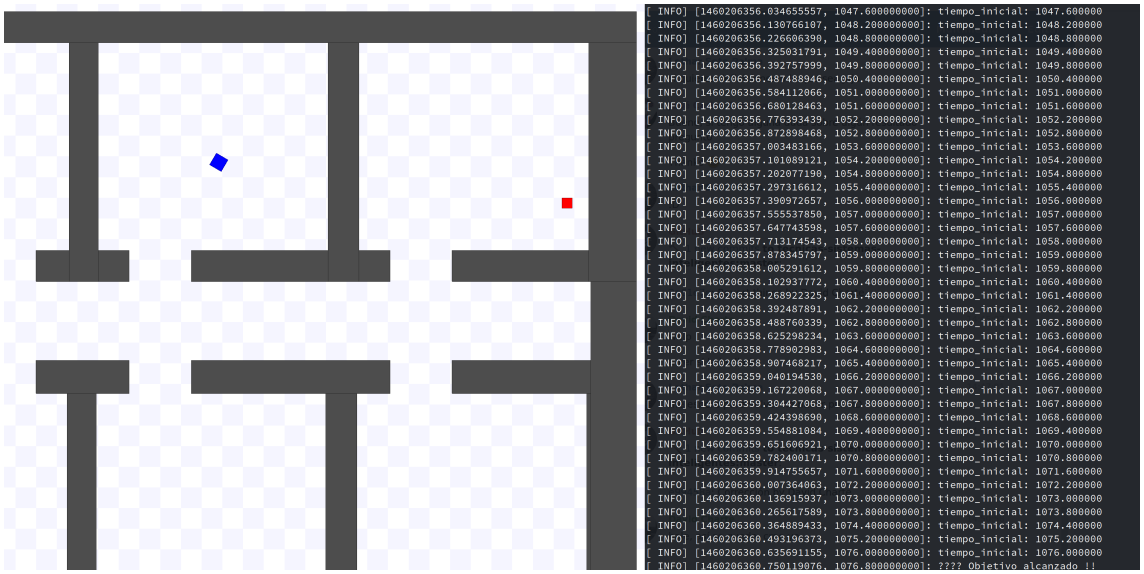

Figura 11: Se alcanza el objetivo inicial

2.d. Mejorar el comportamiento del cliente en los siguientes aspectos:

- Detectar que el robot está “atascado” (demasiado tiempo en una región sin avanzar al objetivo), usando información de “feedback”.
- En caso de atasco cancelar goal actual.
- Determinar un nuevo goal para sacarlo del atasco (usando el proceso de búsqueda local).
- Enviar el nuevo goal, detectar que se ha alcanzado, y volver a enviar el goal original.

Como se explica en la sección 1.d, la función gestora de eventos de *feedback* analiza el tiempo que transcurre sin que el robot se traslade. Cuando se decide que se está ante un mínimo local, se cancela el objetivo primario. Una vez se ha cancelado, el cliente recurre a la heurística de *huida* que trata de buscar una trayectoria alternativa como se desarrolla en 2.c. Se utiliza el método `waitForResult` del *action client* para esperar a que se alcance el objetivo secundario planeado. Si se alcanza sarisfactoriamente, se restablece el objetivo primario para intentar construir una nueva trayectoria hacia el mismo.

3. Consideraciones adicionales.

3.a. Uso de la práctica

Para hacer uso de los dos mapas mostrados se pueden utilizar archivos `.launch` predefinidos en el paquete. Los comandos a utilizar son los siguientes:

```
roslaunch e2subgrupo2_5_costmaps miscostmaps_fake_5cm.launch  # => Simple rooms
roslaunch e2subgrupo2_5_costmaps mimaze_fake_5cm.launch       # => Autolab
```

3.b. Condiciones de carrera

En la implementación del método `setTotalRepulsivo` en el archivo `myPlannerLite.cpp` se ha detectado un problema a la hora de acceder al vector que almacena los obstáculos que el robot está visualizando. Es posible que mientras se está calculando la componente repulsiva total, resultado de la suma de las componentes, se ejecute el callback `scanCallBack`, que cambia el vector de obstáculos, pudiendo modificarlo y derivar en un acceso a posiciones del vector que no están ocupadas, generando un *core* y abortando el programa. Para solucionar esto, en el callback no generamos los nuevos elementos directamente sobre dicho vector, sino que rellenamos un vector nuevo con los obstáculos detectados y usamos el método `swap`, que los intercambia instantáneamente, impidiendo que se realicen accesos incorrectos.