Universidad Zaragoza
1542

Trabajo Fin de Grado

Planificación de trayectorias de equipos de robots en entornos desconocidos

Path planning of team of robots in unknown environments

Autor

Sergio Beltrán García

Director

Cristian Mahulea

Escuela de Ingeniería y Arquitectura

2021-2022

Resumen

Hoy día podemos decir que los robots se han convertido en un recurso valioso para las empresas tecnológicas, siendo los robots móviles uno de los tipos de robots más importantes. Una de las posibles utilidades de este tipo de robots es la recogida y entrega de materiales desde un punto a otro en un entorno dinámico. Para este trabajo suponemos que los robots no conocen a priori el entorno, es decir, no existe ninguna unidad central que pueda dar a estos robots la información real del entorno.

El enfoque principal de este trabajo de fin de grado consiste en implementar un programa en el lenguaje de programación Matlab capaz de simular el movimiento de un equipo de robots, situados en un mapa dividido en celdas y, a priori, desconocido para estos, desde unos estados iniciales hasta unos estados finales en los que se cumplan las metas definidas según una fórmula booleana.

Ya que los mapas son desconocidos a priori por los robots supondremos que estos son capaces de reconocer de alguna forma el entorno que les rodea, ya sea, por ejemplo, gracias a sensores de proximidad o cámaras, entre otros; sin embargo, cabe destacar que el reconocimiento de este entorno se hará de manera probabilista por cada uno de los robots, pudiendo esta información del entorno recogida por los robots no ser totalmente fiel a la información del entorno real. Además, también es importante mencionar que estos robots son capaces de comunicarse entre ellos para compartir la información probabilista del entorno que hayan podido obtener, por ejemplo, con un dispositivo Wifi. Estos aspectos son importantes de cara a un futuro donde se quisiera utilizar esta simulación en un entorno real con robots físicos.

Dentro de los objetivos de desarrollo sostenible (ODS) este trabajo de fin de grado contribuye al objetivo número 9, concretamente con el punto 9.5 el cual tiene como una de sus metas el aumento de la investigación científica.

Abstract

Today we can say that robots have become a valuable resource for technology companies, with mobile robots being one of the most important types of robots. One of the possible utilities of this type of robots is the collection and delivery of materials from one point to another in a dynamic environment. For this work we assume that the robots have no a priori knowledge of the environment, i.e., there is no central unit that can give these robots the actual information of the environment.

The main focus of this end-of-degree project is to implement a program in the Matlab programming language capable of simulating the movement of a team of robots, located on a map divided into cells and, a priori, unknown to them, from initial states to final states in which the goals defined according to a boolean formula are met.

Since the maps are a priori unknown to the robots, we will assume that they are able to somehow recognize the environment around them, either thanks to proximity sensors or cameras, among others; however, it should be noted that the recognition of this environment will be done probabilistically by each of the robots, and this information on the environment collected by the robots may not be completely faithful to the information on the real environment. In addition, it is also important to mention that these robots are capable of communicating with each other to share the probabilistic information of the environment that they have been able to obtain, for example with a Wifi device. These aspects are important for a future where we would like to use this simulation in a real environment with physical robots.

Within the Sustainable Development Goals (SDGs) this thesis contributes to goal number 9, specifically with point 9.5 which has as one of its goals the increase of scientific research.

Índice general

Resumen **I**

Abstract **II**

Índice general **III**

Índice de figuras **VI**

Índice de tablas **IX**

1. Introducción **1**

1.1. Motivación y contexto 1

1.2. Objetivos 2

1.3. Alcance del proyecto 3

1.4. Estructura de la memoria 4

2. Definición del problema **5**

2.1. Entorno 5

2.2. Robots 6

2.3. Redes de Petri 8

2.4. Especificaciones sobre el estado final 10

2.5. Path planning probabilista 11
2.6. Especificación del problema en pseudocódigo 13

3. Herramientas utilizadas **14**
3.1. Matlab 14
3.2. RMTool 14
3.3. Bibliotecas 15
3.3.1. Optimization Toolbox 15
3.3.2. CPlex (opcional) 15
3.3.3. Mapping Toolbox (opcional) 15

4. Estimación distribuida **16**
4.1. Algoritmos de estimación utilizados 18
4.1.1. Algoritmo independiente 19
4.1.2. Algoritmo centralizado 19
4.1.3. Algoritmo de consenso 20

5. Mejoras implementadas **21**
5.1. Introducción de datos por parte del usuario 21
5.2. Creación del entorno de forma manual 22
5.3. Animar el movimiento de los robots 23
5.4. Movimiento a zonas sin explorar 23
5.4.1. Saber a qué zona del mapa ir 24
5.4.2. Obtener la dirección a moverse 25
5.5. Identificar cuando se ha cumplido la misión 27

6. Evaluación **29**

6.1. Radio de comunicación 29
6.2. Movimientos a zonas sin explorar 31
6.3. Algoritmos de estimación 32
6.3.1. Primera prueba 32
6.3.2. Segunda prueba 33
6.3.3. Tercera prueba 35

7. Conclusiones **36**
7.1. Líneas de futuro 36
7.1.1. Mejora en la estimación de posiciones de los robots 36
7.1.2. De simulación a realidad 37

A. Guía de instalación **38**
A.1. Matlab 39
A.2. RMTool y CPlex 39
A.3. Optimization Toolbox y Mapping Toolbox 39

B. Robot Motion Toolbox **41**
B.1. Interfaz 41
B.2. Creación del entorno 42
B.2.1. Definición de las dimensiones 42
B.2.2. Inserción de las regiones y robots 43
B.3. Path Planning 47
B.3.1. Definición de la fórmula booleana 47
B.3.2. Ejecución del Path Planning 47

Bibliografía **50**

Índice de figuras

1.1. Recorridos realizados por los robots en un entorno para cumplir la misión definida por la fórmula booleana 'y_1 & y_2 & y_3 & y_4', siendo y_1 las celdas de color amarillo, y_2 las de color azul, y_3 las de color rosa e y_4 las de color verde. También se muestran los mapas de creencias de cada robot con la información probabilista del tipo de regiones de interés y la estimación de las posiciones de los otros robots en el estado final. 2

2.1. Entorno de dimensiones 3x3 con dos robots, dos regiones de interés y un obstáculo. El robot 1 se encuentra en la celda 1 y el robot 2 en la celda 3. La región de interés y_1 corresponde a la celda 7 de color amarillo y la región de interés y_2 con la celda 9 de color azul. 6

2.2. Iteración i del programa donde dos robots aún no se han podido comunicar al ser su radio de comunicación 5 y estar a una distancia superior a este valor. Los mapas de creencias y la posición estimada son muy diferentes respecto a la información del entorno real. 7

2.3. Iteración i+1 del programa donde dos robots pueden comunicar su información probabilista del entorno. Podemos observar como el robot 1 obtiene información de la parte derecha del entorno la cual obtuvo el robot 2 y como el robot 2 obtiene información de la parte izquierda del entorno obtenida por el robot 1. Los mapas de creencias y la posición estimada por los robots son mejores. 7

2.4. Red de Petri del entorno de la figura 2.1. 8

2.5. Problema de optimización MILP a resolver [1]. 11

4.1. Visualización de las probabilidades de estimar el tipo de región de cada celda de forma correcta por parte de los sensores, donde la celda con el símbolo '**o**' corresponde a la posición del robot. 17

5.1. Iteración en la cual el robot debe moverse de la celda 2 a la celda más cercana sin explorar en un entorno 4x6 según su información probabilista del tipo de regiones, la cual se puede observar en su mapa de creencias. . . 25

5.2. Mapa de creencias del robot para la iteración de la figura 5.1, donde las celdas con la letra 'E' y la celda de inicio se consideran celdas ya exploradas y las celdas con la letra 'D' se consideran las celdas destino a visitar. La celda destino alcanzada con el menor número de movimientos se considera la celda a la que el robot tendrá que ir. 26

5.3. Grafo en forma de árbol generado por el algoritmo de recorrido en anchura para el ejemplo de la figura 5.2. 26

5.4. Resultados devueltos por el algoritmo de recorrido en anchura para el ejemplo de la figura 5.2. 27

5.5. *Path planning* finaliza al llegar a las 50 iteraciones. 27

5.6. *Path planning* finaliza cuando los robots cumplen con la misión. 27

6.1. Estado final de la ejecución del *Path planning* en un entorno 10x20 con 3 robots donde no se ha podido cumplir la misión de ir a las celdas de color amarillo, azul y rosa según la fórmula booleana 'y_1 & y_2 & y_3' debido a la falta de comunicación por parte de los robots. 30

6.2. Estado final de otra ejecución sobre el mismo entorno donde sí se cumple la misión de ir a las celdas de color amarillo, azul y rosa según la fórmula booleana 'y_1 & y_2 & y_3' gracias a cambiar el radio de comunicación a 25 para que los robots puedan comunicarse. 31

6.3. Ejecución donde los robots se mueven de forma aleatoria si el problema MILP no encuentra solución. 32

6.4. Ejecución donde los robots se mueven a las zonas sin explorar más cercanas si el problema MILP no encuentra solución. 32

6.5. Entorno 10x20 utilizado para comparar la media y desviación típica de 50 ejecuciones para cada algoritmo de estimación con un solo robot cuya misión es ir a una celda de color rosa según la fórmula booleana 'y_3'. 33

6.6. Entorno 10x20 utilizado para comparar la media y desviación típica de 50 ejecuciones para cada algoritmo de estimación con tres robots cuya misión es ir a una celda de color amarillo, otra azul y otra rosa según la fórmula booleana 'y_1 & y_2 & y_3'. 34

A.1. Pasos para buscar las bibliotecas. 40

A.2. Ventana donde poder instalar la biblioteca Optimization Toolbox. 40

A.3. Ventana donde poder instalar la biblioteca Mapping Toolbox. 40

B.1. Interfaz de RMTool. 41

B.2. Desplegable a clicar para poder modificar los valores de las dimensiones de un entorno. 42

B.3. Ventana donde introducir los valores de las dimensiones de un entorno. . . 42

B.4. Interfaz de RMTool tras cambiar las dimensiones del entorno a 5x7. 43

B.5. Botones 'Boolean formula' y 'Unknown environment' activados. 43

B.6. Ventana donde introducir los valores del número de robots y el número de tipos de regiones de interés. 44

B.7. Ventana donde elegir la forma de generar el entorno. 44

B.8. Ventana donde introducir el número de regiones de cada tipo. 44

B.9. Interfaz de RMTool tras generar un entorno desconocido de forma aleatoria. 45

B.10. Antes de hacer clic sobre la celda a cambiar a región de tipo 1. 46

B.11. Tras hacer clic la celda pasa a ser una región de tipo 1. 46

B.12. Antes de hacer clic sobre la celda donde colocar el primer robot. 46

B.13. Tras hacer clic la celda pasa a contener el robot 1. 46

B.14. Interfaz de RMTool tras generar un entorno desconocido manualmente. . . 47

B.15. Ventana donde introducir el tipo de algoritmo a utilizar en el *Path Planning*, el radio de comunicación de los robots, las probabilidades tau y tau obs y el número máximo de iteraciones del *Path Planning*. 48

B.16. Ejecución del *Path planning* con fórmula booleana y_1 & y_2 & y_3 & $!y_4$. . . . 48

Índice de tablas

2.1. Opciones posibles para poder definir una fórmula booleana que deban cumplir los robots en su estado final. 10

4.1. Probabilidad de que un robot acierte al estimar el tipo de región de una celda según la distancia entre este y la celda a estimar. 16

5.1. Datos que pueden ser modificados por parte del usuario. 22

6.1. Resultados obtenidos tras 50 ejecuciones del programa para cada algoritmo de estimación en un entorno de un solo robot con una misión booleana formada por una meta (y_3). 33

6.2. Resultados obtenidos tras 50 ejecuciones del programa para cada algoritmo de estimación en un entorno de tres robots con una misión booleana formada por tres metas (y_1 & y_2 & y_3). 34

6.3. Resultados de las 50 ejecuciones del programa utilizando el algoritmo de consenso cambiando el radio de comunicación por los valores 5, 10 y 25. El entorno y misión que se han utilizado para esta prueba son los mismos que los de la figura 6.6. 35

Capítulo 1

Introducción

Este capítulo expone la motivación y contexto del trabajo, además de sus objetivos y el alcance de este. Tras esto se presenta la estructura que sigue esta memoria, indicando y explicando de qué tratan los capítulos en los que está dividida.

1.1. Motivación y contexto

Un problema a abordar por los robots móviles trata el cómo planificar los caminos por los que estos robots tienen que ir para cumplir su misión. El primer paso para saber qué caminos tienen que seguir un conjunto de robots consiste en hacer que estos conozcan el entorno que les rodea, ya sea, por ejemplo, gracias a una unidad central que les suministre la información de dicho entorno para que estos sepan hacia qué zona moverse para cumplir la misión, evitando posibles colisiones con obstáculos.

Sin embargo, para nuestro caso esta unidad central no existe, es decir, los robots tienen que moverse hasta cumplir su misión partiendo desde un entorno desconocido para estos. Debido a que para cumplir con la misión se necesita esta información del entorno para saber hacia dónde tiene que ir cada robot asumiremos que estos tienen algún tipo de sensor que les permita reconocer el entorno de manera probabilista, asumiendo que estos sensores ofrecen una información más exacta del entorno a cortas distancias. De esta forma la información del entorno obtenida por cada robot puede no ser un reflejo de la realidad, sin embargo, a medida que estos robots avanzan por el entorno la información de este es cada vez más exacta, debido a que cada vez que estos robots avancen la información probabilista es actualizada a partir de la información probabilista previa.

En la figura 1.1 se muestra el ejemplo de una ejecución del programa implementado, donde se puede observar el recorrido de los robots para cumplir con la misión de llegar hasta las regiones de interés y_1, y_2, y_3 e y_4 (celdas de color amarillo, azul, rosa y verde,

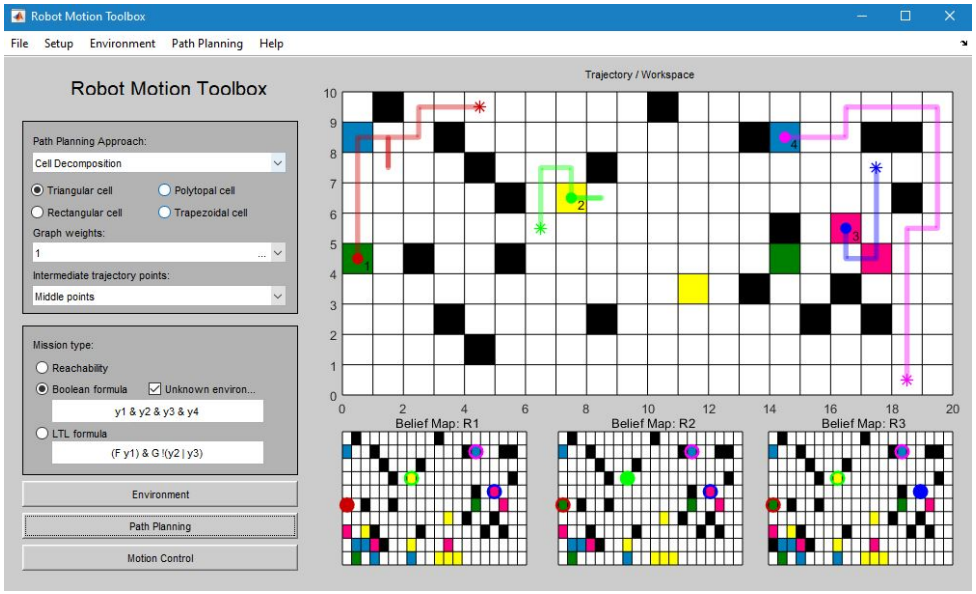

Figura 1.1: Recorridos realizados por los robots en un entorno para cumplir la misión definida por la fórmula booleana 'y_1 & y_2 & y_3 & y_4', siendo y_1 las celdas de color amarillo, y_2 las de color azul, y_3 las de color rosa e y_4 las de color verde. También se muestran los mapas de creencias de cada robot con la información probabilista del tipo de regiones de interés y la estimación de las posiciones de los otros robots en el estado final.

respectivamente). Esta misión está definida por la fórmula booleana en forma normal conjuntiva 'y_1&y_2&y_3&y_4', la cual indica que al final de la ejecución queremos que al menos un robot se encuentre en una celda de color amarillo, otro en una de color azul, otro en una de color rosa y otro en una de color verde. Además, también se observan los mapas de creencias o *belief maps* de los tres primeros robots, siendo estos mapas una representación de la información probabilista obtenida del entorno por cada robot, la cual es actualizada en tiempo real a la hora de ejecutar el *Path planning*.

1.2. Objetivos

El objetivo del trabajo es integrar en la herramienta RMTool un programa capaz de generar entornos divididos en celdas que sean desconocidos a priori por un equipo de robots y simular el movimiento de estos robots en dicho entorno de manera que cumplan con una fórmula booleana la cual indique el estado final de los robots.

Cabe destacar que este programa parte de una implementación previa capaz de simular el movimiento de un conjunto de robots en un entorno desconocido de forma que estos cumplan con la misión definida por una fórmula booleana. Sin embargo, esta implementación previa presenta algunas limitaciones explicadas en el capítulo 5. Es por esto que

dentro de los objetivos de este trabajo de fin de grado, aparte de implementar este programa en la herramienta RMTool, se tiene también en cuenta la realización de una serie de mejoras sobre la implementación mencionada anteriormente.

1.3. Alcance del proyecto

Aquí se comentan los diferentes pasos seguidos para la realización de este trabajo.

Requisitos previos: Para poder realizar este trabajo se necesitan las herramientas mencionadas en el capítulo 3, las cuales pueden ser instaladas siguiendo la guía de instalación del Anexo A. Además, se ha requerido de un conocimiento previo en el lenguaje propio de Matlab. Este conocimiento previo del lenguaje se obtuvo gracias a la asignatura de Aprendizaje Automático del grado en Ingeniería Informática, al ser Matlab el lenguaje de programación utilizado. Sin embargo, ha sido necesario investigar más sobre este lenguaje, concretamente en conocer cómo funcionan las interfaces gráficas en Matlab. Esto ha sido necesario debido a que en este trabajo se ha trabajado sobre una herramienta, RMTool, que utiliza una interfaz gráfica interactiva por parte del usuario.

Pasar la anterior implementación a RMTool: Se ha modificado el código y la interfaz de la herramienta RMTool con el fin de que la implementación previa pueda ser utilizada de forma interactiva por parte del usuario.

Mejorar la anterior implementación: Teniendo ya la anterior implementación integrada en RMTool se ha pasado a mejorarla, permitiendo (i) crear ahora entornos de forma manual haciendo que el usuario interactue con el entorno plasmado en RMTool, (ii) añadiendo la opción de que el usuario pueda modificar un conjunto de valores los cuales antes eran fijos y (iii) evitando que los robots se muevan de forma aleatoria al no encontrar una solución al problema de planificación debido a la falta de información exacta del entorno, entre otras más mejoras. Todas las mejoras implementadas se pueden encontrar descritas en el capítulo 5.

Evaluación y comparación: Con todas las mejoras ya implementadas se ha realizado una serie de simulaciones, modificando en cada una de estas algún parámetro del algoritmo, como por ejemplo el número de robots, el tipo de estimación utilizado, entre otros, con el fin de evaluar y comparar los resultados obtenidos.

1.4. Estructura de la memoria

En este apartado se describe la estructura de la memoria de este trabajo de fin de grado.

Capítulo 1. Introducción: Se expone la motivación, el contexto, los objetivos y el alcance del proyecto.

Capítulo 2. Definición del problema: Se define el problema a resolver. En este apartado se explican conceptos tales como la definición del entorno, cómo funcionan los robots, las redes de Petri, fórmulas booleanas y la planificación de caminos o *Path planning* para entornos desconocidos.

Capítulo 3. Herramientas: Se nombran y explican todas las herramientas utilizadas para la realización del trabajo.

Capítulo 4. Estimación distribuida: Se comentan y explican cómo funcionan los tres algoritmos de estimación distribuida (el algoritmo independiente, el algoritmo centralizado y el algoritmo de consenso) utilizados a la hora de actualizar la información probabilista del entorno en cada robot.

Capítulo 5. Mejoras: En este capítulo se comentan todas las mejoras respecto a la anterior implementación, las cuales fueron implementadas en la herramienta RMTool.

Capítulo 6. Evaluación: Se muestran y comparan los resultados de múltiples conjuntos de ejecuciones del programa.

Capítulo 7. Conclusiones: En este último capítulo se presentan las conclusiones del trabajo y sus líneas de futuro.

Capítulo 2

Definición del problema

En este capítulo se explica el problema de planificación de trayectorias para un equipo de robots en un entorno desconocido [1] [2] [3]. Este se ha dividido en secciones, donde cada una de ellas define diferentes conceptos clave del problema a tratar.

2.1. Entorno

El entorno sobre el cual se mueven los robots es continuo, pero está dividido en celdas cuadradas, también llamadas regiones. Cada una de estas celdas se identifica con un número entero positivo empezando desde el 1 y terminando en el número total de celdas. El orden para identificar estas celdas en el entorno va de izquierda a derecha y de abajo a arriba. Estas celdas o regiones pueden ser de uno de los siguientes tipos:

Región de interés: este tipo de región se refiere a las regiones que pueden ser utilizadas como meta de una misión definida por una fórmula booleana. En el entorno se pueden identificar por aquellas celdas que se encuentren coloreadas por un color que no sea ni negro (utilizado para las regiones de tipo obstáculo) ni blanco (utilizado para el espacio libre).

Región de tipo obstáculo: los obstáculos son un tipo de región los cuales los robots no pueden traspasar. En el entorno se muestran como celdas de color negro.

Espacio libre: este tipo de región se refiere a las celdas por las cuales los robots pueden avanzar sin problema. En el entorno aparecen como celdas de color blanco.

La figura 2.1 muestra un ejemplo de un entorno, donde se pueden observar en una ventana de texto los números de celdas para cada región de interés y obstáculos. En este caso existen dos robots situados en las celdas 1 y 3 del entorno, además existe un obstáculo a evitar en la celda 6. Por último, se encuentran las regiones de interés las

cuales se encuentran en las celdas 7 y 9, siendo la celda 7 la región de interés de tipo y_1 (amarillo) y la celda 9 la región de interés de tipo y_2 (azul).

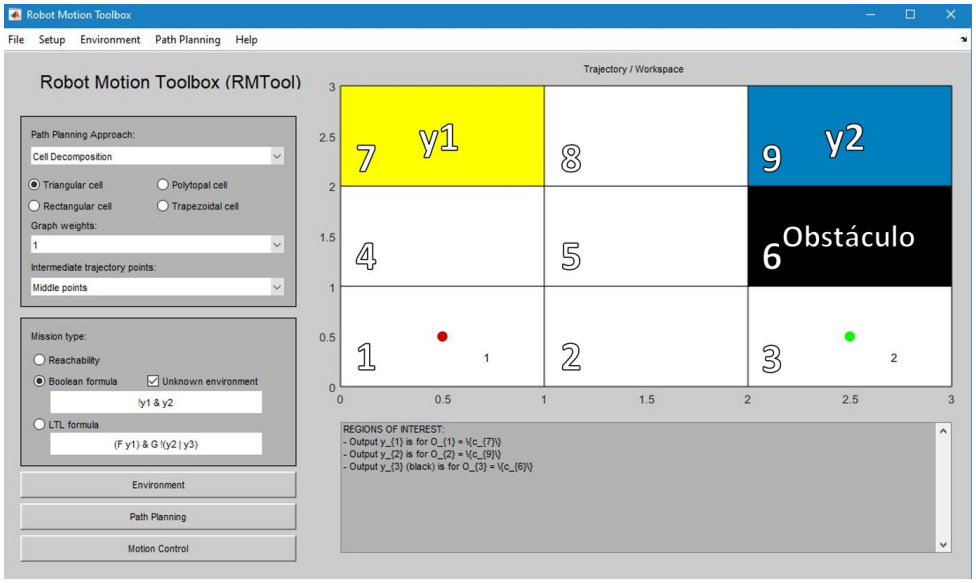

Figura 2.1: Entorno de dimensiones 3x3 con dos robots, dos regiones de interés y un obstáculo. El robot 1 se encuentra en la celda 1 y el robot 2 en la celda 3. La región de interés y_1 corresponde a la celda 7 de color amarillo y la región de interés y_2 con la celda 9 de color azul.

2.2. Robots

Los robots se encargan de moverse por el entorno con el fin de cumplir una misión definida por una fórmula booleana en forma normal conjuntiva definida sobre las regiones de interés. Un ejemplo de este tipo de fórmula es la ya mencionada anteriormente en el capítulo 1, siendo esta 'y_1 & y_2 & y_3 & y_4'.

El movimiento de los robots está definido por la 4-vecindad de la celda en la cual se encuentren, es decir, un robot puede moverse a las celdas que compartan un lado con la celda actual donde se encuentra el robot (arriba, abajo, izquierda o derecha).

Suponemos que estos robots son capaces de estimar el entorno que les rodea gracias a algún tipo de sensor con el que se pueda obtener una información probabilista. Además, también suponemos que estos pueden comunicarse entre ellos gracias a algún dispositivo, por ejemplo, un dispositivo Wifi, pudiendo así compartir la información probabilista obtenida, siempre y cuando los robots se encuentren a una distancia menor o igual a un radio de comunicación indicado por el usuario.

Cada uno de los robots realiza en cada movimiento una estimación probabilista del

entorno, obteniendo de esta forma una estimación del tipo de regiones para cada celda del entorno y una estimación de las posiciones del resto de robots.

La estimación del tipo de regiones se realiza gracias a un algoritmo de estimación indicado por el usuario. Los algoritmos a elegir son el algoritmo independiente, el centralizado y el de consenso. El funcionamiento de cada uno de estos algoritmos se encuentra en el capítulo 4.

Por otro lado, las posiciones del resto de robots según el robot R dependen del radio de comunicación de los robots. Si el robot R se encuentra dentro del radio de comunicación con otros robots entonces las posiciones de estos otros robots para el robot R serán las posiciones reales. Por el contrario, si este robot R no se encuentra dentro de su radio de comunicación con otros robots entonces la posición de estos es una posición estimada.

En las siguientes figuras 2.2 y 2.3 se muestra un ejemplo de comunicación entre dos robots cuando el rango de comunicación es 5.

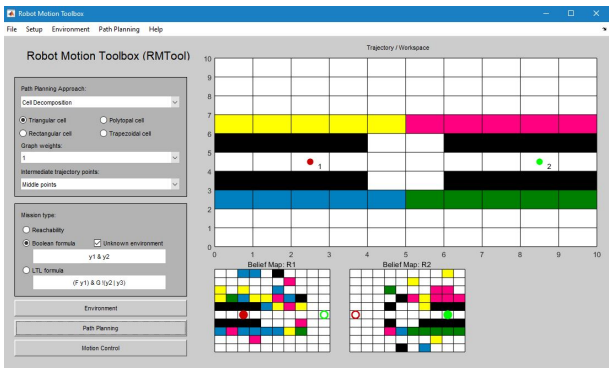

Figura 2.2: Iteración i del programa donde dos robots aún no se han podido comunicar al ser su radio de comunicación 5 y estar a una distancia superior a este valor. Los mapas de creencias y la posición estimada son muy diferentes respecto a la información del entorno real.

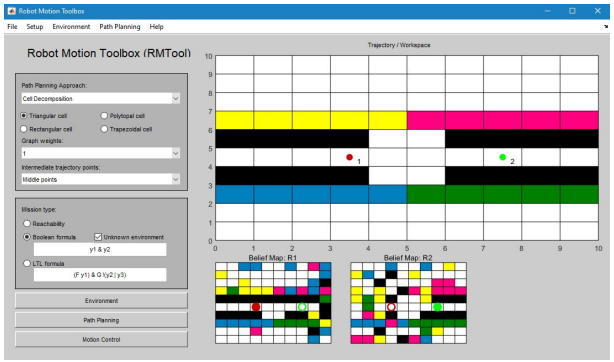

Figura 2.3: Iteración i+1 del programa donde dos robots pueden comunicar su información probabilista del entorno. Podemos observar como el robot 1 obtiene información de la parte derecha del entorno la cual obtuvo el robot 2 y como el robot 2 obtiene información de la parte izquierda del entorno obtenida por el robot 1. Los mapas de creencias y la posición estimada por los robots son mejores.

Como se puede observar la información probabilista del entorno del robot 1 se actualiza con la información probabilista del 2 y viceversa en la iteración `i+1`.

2.3. Redes de Petri

Para este trabajo las redes de Petri nos sirven para poder modelar el sistema discreto en el cual los robots deben moverse de forma distribuida para poder alcanzar la misión definida por una fórmula booleana.

Estas redes se componen de 4 elementos:

Lugares: son nodos que pueden tener asignados un número entero de marcas. Para el caso de nuestro problema un lugar modela una celda en la cual puede encontrase un robot. De forma gráfica se representan por un círculo.

Transiciones: se pueden disparar o no si están sensibilizadas. En caso de que se disparen se consumen marcas desde cada lugar de entrada a la transición y se producen marcas en los lugares de salida de la transición. Para este problema las transiciones modelan los movimientos de un robot de una celda a otra adyacente. De forma gráfica se representan por un rectángulo.

Arcos: son direccionales y conectan un lugar con una transición o una transición con un lugar. De forma gráfica se representan con una flecha.

Marcas: en nuestro modelo las marcas representan a los robots. De forma gráfica se representan con un punto.

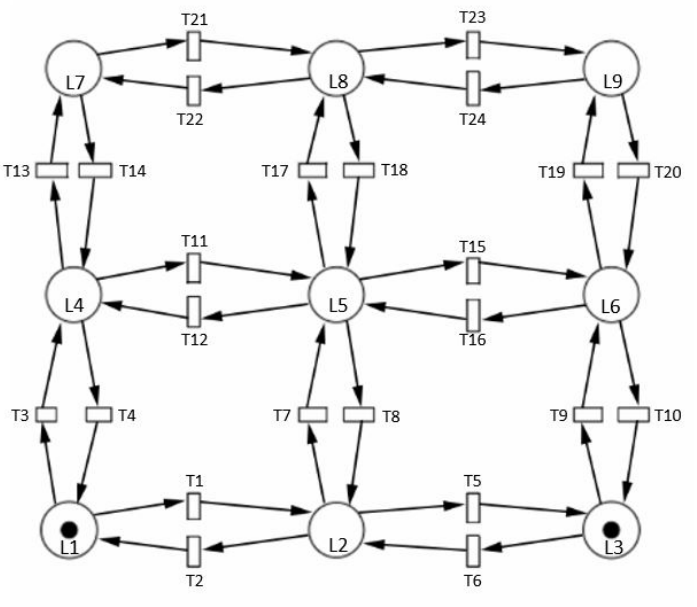

Figura 2.4: Red de Petri del entorno de la figura 2.1.

La figura 2.4 muestra la red de Petri modelando el entorno de la figura 2.1.

Formalmente, una red de Petri es una tupla `Q = <N,` m_0`, Y, h>`, donde,

- `N = <P, T, Post, Pre>`
  - **P** siendo el conjunto de lugares. Cada lugar modela una celda del entorno.
  - **T** siendo el conjunto de transiciones. Cada transición modela la posibilidad de movimiento de un robot entre dos celdas adyacentes.
  - **Pre** siendo la matriz de pre-incidencia. Define los arcos de lugares a transiciones, es decir, $Pre[p,t] = 1$ si el lugar p es un lugar de entrada en la transición t, en caso contrario $Pre[p,t] = 0$.
  - **Post** siendo la matriz de post-incidencia. Define los arcos de transiciones a lugares, es decir, $Post[p,t] = 1$ si la transición t es de entrada en el lugar p, en caso contrario $Post[p,t] = 0$.

- $\boldsymbol{m_0}$ es el marcado inicial. Este marcado indica el número inicial de marcas de cada lugar, es decir, para nuestro modelo indica el número de robots que hay inicialmente en cada celda, siendo $m_0[p_i]$ el número de robots en la celda modelada por el lugar p_i.

- **Y** es el conjunto de observaciones, en este caso los posibles tipos de región existentes en el entorno.

- **h** es la función de observación, donde h(p_i) corresponde al tipo de región de la celda modelada por el lugar p_i. Nótese que esta función de observación no es conocida por los robots, al estimar estos el entorno de manera probabilista gracias a sus sensores. Lo que sí conocen los robots son las probabilidades estimadas de cada tipo de región para cada celda del entorno.

La representación del entorno con una red de Petri es útil ya que gracias a su ecuación de estado o ecuación fundamental $\overline{m} = m + C\mathring{u}\sigma$, donde $C = Post - Pre$ es la matriz de incidencia y σ es el vector de disparos de transiciones para llegar al marcado final $\overline{m}$ desde el marcado inicial m, podemos escribir de forma matemática el estado obtenido desde un estado inicial tras disparar una secuencia de transiciones.

Esta ecuación se puede utilizar para definir y resolver problemas de optimización, ya que esta ecuación es una condición necesaria para que un marcado sea alcanzable desde el marcado inicial.

2.4. Especificaciones sobre el estado final

Para que una ejecución del programa sea considerada satisfactoria los robots deben cumplir con una misión en su estado final. Esta misión se compone por una serie de metas a las cuales deben o no llegar los robots. Estas metas corresponden con las posibles regiones de interés en las cuales queremos que los robots se encuentren o no en el estado final.

La misión que deben cumplir los robots se especifica con una fórmula booleana la cual debe ser introducida por parte del usuario en forma normal conjuntiva [4] (FNC o CNF en inglés), es decir, la fórmula debe estar formada a partir de una conjunción de disyunciones. Que esta fórmula deba ser introducida en este formato se debe a temas de implementación.

La fórmula se define como una cadena de texto, a continuación, se muestra la tabla 2.1 que indica las posibles opciones a insertar en esta fórmula.

Dato	**Cadena a introducir**
Región de interés	'yN' (siendo N el número de una región de interés)
Conjunción	'&'
Disyunción	'|'
Negación	'!'

Tabla 2.1: Opciones posibles para poder definir una fórmula booleana que deban cumplir los robots en su estado final.

Así pues, si se tuviese un entorno con 4 tipos de región de interés, 3 robots y quisiésemos que estos llegaran a una región de interés de tipo 1 y a otra de tipo 3 pero no quisiéramos que estos estuviesen o en la región de interés de tipo 2 o en la región de interés de tipo 4 la fórmula quedaría como:

'y_1 & y_3 & (!y_2 | !y_4)'

Como se puede observar esta fórmula cumple con la forma normal conjuntiva, al ser una conjunción de disyunciones, y sigue las reglas de la tabla 2.1.

2.5. Path planning probabilista

Un algoritmo de *Path planning* sirve para obtener las trayectorias de los robots las cuales evolucionan en un entorno desde unos puntos iniciales, que es donde se encuentran inicialmente los robots, hasta unos finales donde se cumpla una meta. Para el caso de entornos conocidos que sean estáticos y enfoques centralizados por parte de los robots bastaría con ejecutar este tipo de algoritmo una sola vez, dando como una de las variables de entrada a este algoritmo la información del entorno real, la cual sería conocida por todos los robots, además de ser información constante. Sin embargo, para nuestro caso esto no funciona, ya que en este problema el entorno real es desconocido a priori por los robots, por lo que tendrán que moverse según sus mapas de creencias, mapas los cuales han sido obtenidos a partir de información probabilista del entorno. Al ser esta información probabilista actualizada en cada iteración del programa (cada vez que se mueven todos los robots a una celda adyacente) el entorno en el que estos se muevan podrá variar en cada iteración.

Es por esto que necesitamos ejecutar el algoritmo que nos permite obtener las trayectorias que deben seguir los robots más de una vez, concretamente una vez por iteración para cada robot, finalizando cuando estos robots cumplan con la misión establecida o hasta alcanzar un número máximo de iteraciones.

Las variables necesarias para poder ejecutar este algoritmo son la red de Petri, la fórmula booleana en forma normal conjuntiva, la información probabilista del entorno para un robot R y las probabilidades tau y tau obs, donde tau y tau obs son valores introducidos por el usuario y cuya definición se comenta más adelante en esta sección del capítulo. Con estas variables el algoritmo de *Path planning* encuentra una posible solución, siendo esta las trayectorias que tendrían que realizar todos los robots para cumplir la misión satisfactoriamente. Nótese que las trayectorias para los demás robots que no sean R son trayectorias calculadas a partir de las posiciones estimadas, que no reales, de los otros robots por parte de R.

Para obtener esta solución el algoritmo de *Path planning* obtiene primero el vector de disparos de transiciones ejecutando un problema de tipo MILP (un problema lineal de enteros mixto). Este tipo de problema se basa en optimizar una función objetivo sujeta a una serie de restricciones lineales que debe cumplir.

$$\min_{\mathbf{m},\tilde{\sigma},\mathbf{x},b} \quad J_\sigma + J_{\mathbb{P}} + J_{obs} + J_{col}$$

$$s.t. \begin{cases} \mathbf{m} &= \mathbf{m}_0 + \mathbf{C}\cdot\tilde{\sigma} \\ \eta\cdot\tilde{\sigma} &= \mathbf{0}_T \\ \mathbf{A}_{task}\cdot\mathbf{x} &\leq \mathbf{b}_{task} \\ -\mathbf{W}\cdot\mathbf{m} + \tau_{\mathbb{P}}\cdot\mathbf{1}_M &\leq N\cdot\mathbf{x} \\ \mathbf{W}\cdot\mathbf{m} - \tau_{\mathbb{P}}\cdot\mathbf{1}_M &\leq N\cdot(\mathbf{1}_M - \mathbf{x}) \\ \mathbf{Post}\cdot\tilde{\sigma} &\leq b\cdot\mathbf{1}_P^T \end{cases}$$

Figura 2.5: Problema de optimización MILP a resolver [1].

La figura 2.5 muestra el problema MILP a resolver, donde $\boldsymbol{min\ J_\sigma + J_P + J_{obs} + J_{col}}$ se refiere a la función a optimizar cumpliendo las restricciones indicadas por 's.t'. Esta función a optimizar busca minimizar el número de transiciones disparadas ($\boldsymbol{J_\sigma}$), priorizar soluciones que tengan mayor probabilidad de que se cumpla la fórmula booleana ($\boldsymbol{J_P}$), priorizar los recorridos con la menor probabilidad de pasar por obstáculos ($\boldsymbol{J_{obs}}$) y penalizar los recorridos en los que dos o más robots puedan chocar ($\boldsymbol{J_{col}}$).

Para las restricciones, $\boldsymbol{m = m_0 + C\sigma}$ indica que tiene que ser posible llegar desde los lugares marcados iniciales hasta los marcados finales que cumplan la fórmula. $\boldsymbol{nu\sigma = 0_T}$ evita que los recorridos pasen por celdas que tengan una probabilidad mayor o igual a tau obs de ser obstáculos. $\boldsymbol{A_{task}x <= b_{task}}$ asegura que la fórmula booleana se cumpla suponiendo que la información probabilista del entorno recogida por el robot sea real. Es decir, asegura que el recorrido cumpla la misión en el mapa de creencias del robot, sin embargo, este recorrido puede que no cumpla con la fórmula booleana en el entorno real. $\boldsymbol{-Wm+T_P1_M <= Nx}$ y $\boldsymbol{Wm-T_P1_M <= N(1_M-x)}$ indican que la probabilidad de que realmente se cumpla la fórmula según la información probabilista del robot debe ser mayor o igual al parámetro tau. Por último, $\boldsymbol{Post\sigma <= b1_P^T}$ evita la colisión entre robots. Concretamente evita que más de 'b' robots puedan encontrarse en la misma celda a la vez, para el caso de nuestro programa consideramos 1 como valor de 'b', evitando así que dos robots o más puedan chocar en la solución a este problema de optimización.

En el caso de encontrar una solución al MILP se obtiene la trayectoria para el robot R a partir del vector de disparo obtenido y las supuestas trayectorias para el resto de robots. Con las trayectorias obtenidas el robot R realiza el primer movimiento indicado por la trayectoria y actualiza las posiciones estimadas de los otros robots según el primer movimiento de sus trayectorias calculadas.

Si no se ha encontrado una solución al MILP, según la anterior implementación, el robot se movía a una celda adyacente elegida de forma aleatoria. Sin embargo, esto se ha cambiado haciendo que ahora el robot se mueva a zonas del mapa que se encuentran aún sin explorar con el fin de obtener más información del entorno. Este cambio en la implementación se explica en el capítulo 5. Además, se hace la suposición de que las posiciones estimadas del resto de robots no cambian.

2.6. Especificación del problema en pseudocódigo

A continuación, se muestra la especificación del problema en pseudocódigo para un robot R.

Input : Dimensiones del entorno; número de robots; número de distintos tipos de regiones; posiciones de los robots; posiciones de las regiones; tipo de estimación; radio de comunicación; probabilidad tau; probabilidad tau obs; MAXITER.

Output: Información probabilista del entorno para el robot R y la secuencia de celdas recorridas por el robot R en el estado final

```
1  Partición del entorno en celdas cuadradas
2  Construir la red de Petri a partir de las dimensiones del entorno, el número de robots y
     sus posiciones y el número de regiones y sus posiciones
3  MisionCumplida = false
4  while not MisionCumplida & el número de iteraciones <MAXITER do
5  |  Obtener la información probabilista del entorno para el robot R según su sensor
6  |  Hacer que el robot R se comunique con el resto de robots que se encuentren dentro de
   |    su radio de comunicación para obtener información probabilista del entorno
7  |  Actualizar la información probabilista del robot R según el tipo de estimación utilizado
8  |  Resolver el MILP
9  |  if se ha encontrado una solución al MILP then
10 |  |  Obtener las trayectorias a realizar por todos los robots
11 |  |  Actualizar la posición del robot R según la primera celda a moverse según la
   |  |    trayectoria obtenida para este robot R
12 |  |  Actualizar las posiciones del resto de robots para el robot R de forma estimada
   |  |    según las primeras celdas de sus respectivas trayectorias
13 |  else
14 |  |  Calcular la trayectoria con el menor número de movimientos para llegar a una
   |  |    celda sin explorar
15 |  |  Actualizar la posición del robot R según la primera celda a moverse en la trayectoria
   |  |    calculada
16 |  |  No se actualizan las posiciones del resto de robots para el robot R, suponemos que
   |  |    no se mueven
17 |  end
18 |  Comprobar que se haya cumplido la misión
19 |  if se ha cumplido la misión then
20 |  |  MisionCumplida = true
21 |  end
22 end
```

Capítulo 3

Herramientas utilizadas

En este capítulo se indican las herramientas utilizadas para la realización de este trabajo de fin de grado. La guía de instalación de estas herramientas se puede encontrar en el Anexo A.

3.1. Matlab

Matlab [5] se ha usado como entorno de desarrollo donde implementar en el lenguaje de programación Matlab el problema abordado en este trabajo de fin de grado.

3.2. RMTool

RMTool, abreviatura de Robot Motion Toolbox, se trata de una herramienta integrada en Matlab desarrollada en el Departamento de Informática e Ingeniería de Sistemas, la cual sirve para poder definir entornos y realizar simulaciones del movimiento de un equipo de robots a través de una interfaz interactiva.

Es sobre esta herramienta donde se ha añadido la implementación para poder generar entornos desconocidos a priori por un equipo de robots y poder simular sus movimientos según una fórmula booleana que indique sus estados finales.

Cabe recalcar que esta herramienta de por sí ya tenía una opción para poder generar entornos y simular el movimiento de robots según una fórmula booleana, sin embargo, esta opción hacía la suposición de que existía una unidad central que hiciese que los mapas fuesen conocidos a priori por los robots.

3.3. Bibliotecas

En esta sección se nombran las bibliotecas de Matlab utilizadas por la herramienta RMTool.

3.3.1. Optimization Toolbox

Esta biblioteca [6] es útil para poder resolver problemas de optimización. Para el caso de este trabajo sirve para resolver el problema de optimización lineal con enteros mixtos (MILP) gracias a la función 'intlinprog' propia de esta biblioteca.

3.3.2. CPlex (opcional)

Esta biblioteca [7] tiene como funcionalidad resolver problemas de optimización. Sin embargo, se dice que es de instalación opcional debido a que no es utilizada para la resolución del problema planteado en este trabajo. Por otra parte, sí es utilizada por la herramienta RMTool para poder solucionar problemas de optimización ajenos a este trabajo.

3.3.3. Mapping Toolbox (opcional)

Esta biblioteca [8] ofrece funciones para poder analizar y visualizar información geográfica. Se dice que esta biblioteca es opcional ya que no es usada por el programa abordado en este trabajo de fin de grado; sin embargo, es utilizada por RMTool, por ejemplo, a la hora de generar un entorno con la especificación booleana que supone que los mapas sí son conocidos a priori por los robots. Esto es así ya que esta forma de generar el entorno usa la función 'polyxpoly' propia de esta biblioteca, la cual sirve para encontrar los puntos de intersección entre dos líneas.

Capítulo 4

Estimación distribuida

Como ya se ha comentado, en este problema los robots no conocen a priori el entorno en el que se encuentran, es por esto que el conjunto de robots debe realizar una estimación del entorno.

Para ello cada robot tiene que estimar este entorno de forma probabilista, otorgando a cada celda de este unas probabilidades, una por cada tipo de región (regiones de interés, obstáculo y espacio libre). Cada probabilidad va ligada a un tipo de región e indica cómo de seguro está el robot de que la celda a estimar sea de ese tipo. Para cada celda el tipo de región con la mayor probabilidad será el tipo de región que el robot cree ver y que se utilizará en el algoritmo de planificación de trayectorias, pudiendo ser con mayor o menor seguridad el tipo de región real.

Un dato a tener en cuenta es que los robots pueden estar más o menos seguros de estas probabilidades dependiendo de sus sensores. Suponemos que la seguridad de los sensores depende de la distancia respecto a la posición del robot y la posición de la celda a estimar. Esta tabla recoge la seguridad para cada rango de distancias, entendiendo seguridad como la probabilidad de que el robot acierte el tipo de región de la celda a estimar, siendo `NumTipos` el número de tipos de regiones (regiones de interés, obstáculo y vacío):

Seguridad	**Rango de distancias**
1 (100 %)	<=1
1/1.5 (66.$\overline{\mathbf{6}}$ %)	>1 & <=3
1/2 (50 %)	>3 & <=6
`1/NumTipos`	>6

Tabla 4.1: Probabilidad de que un robot acierte al estimar el tipo de región de una celda según la distancia entre este y la celda a estimar.

En la figura 4.1 se muestra una representación visual de esta tabla, siendo el símbolo '**o**' un robot, las celdas verdes aquellas en el rango de distancia <=1 las cuales el robot tiene una seguridad de 1, las amarillas aquellas en el rango >1 & <=3 las cuales el robot tiene una seguridad de 1/1.5, las naranjas aquellas en el rango >3 & <=6 las cuales el robot tiene una seguridad de 1/2 y las blancas aquellas en el rango >6 las cuales el robot tiene una seguridad de `1/NumTipos`.

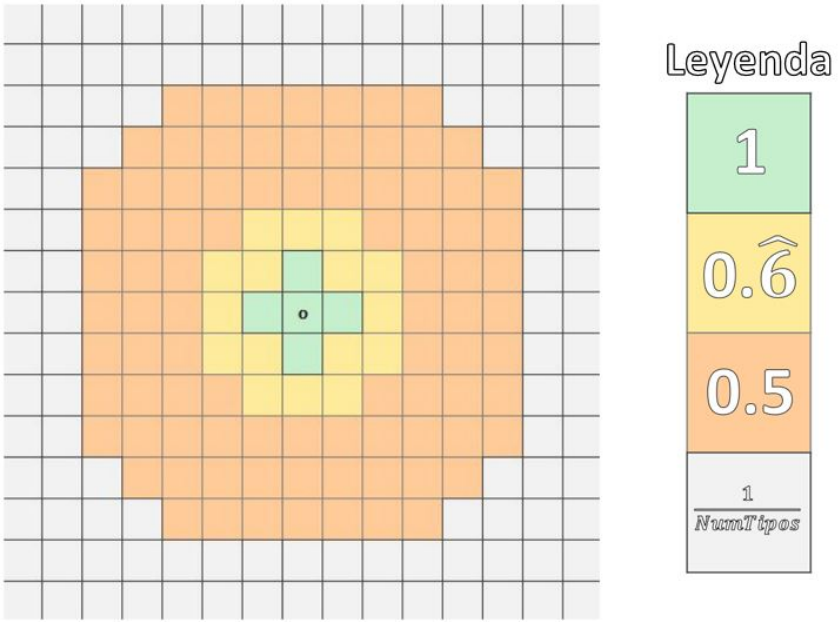

Figura 4.1: Visualización de las probabilidades de estimar el tipo de región de cada celda de forma correcta por parte de los sensores, donde la celda con el símbolo '**o**' corresponde a la posición del robot.

Con la información de la seguridad de cada celda realizamos la estimación del tipo de región para cada celda del entorno.

Como se puede observar, en el caso donde la distancia sea 1 (la celda a estimar es adyacente/vecina a la celda donde se encuentra el robot) el robot tiene una seguridad de 1 (el 100 %), por lo que la estimación del tipo de región será acorde a la realidad, para los casos donde las distancias sean mayores suponemos que el sensor del robot no es del todo preciso, perdiendo a mayor distancia más seguridad, haciendo así que los tipos de regiones estimadas puedan no coincidir en un principio con la realidad. Por ejemplo, para una celda a distancia mayor a uno y menor o igual a tres la probabilidad de estimar correctamente el tipo de celda será del $0.\overline{\mathbf{6}}$, tras esto el robot indicará que la celda estimada, ya sea una estimación correcta o no, tiene un $0.\overline{\mathbf{6}}$ de probabilidad de ser el tipo de región correcto, mientras que la probabilidad restante de $0.\overline{\mathbf{3}}$ se dividirá equitativamente para el resto de tipos de celda.

Esta estimación se realiza en cada iteración del *Path planning*. Ya que en cada itera-

ción se calculan las nuevas probabilidades a partir de aquellas obtenidas en la anterior iteración podemos decir que este tipo de estimación se retroalimenta, además, cuantas más iteraciones pasen más segura y acertada es la estimación de cada robot.

4.1. Algoritmos de estimación utilizados

A continuación, se presentan y explican los tres posibles algoritmos de estimación a utilizar.

Cabe recalcar que, aunque las probabilidades finales del tipo de región obtenidas por cada algoritmo puedan ser diferentes, todos estos algoritmos se apoyan sobre el teorema de Bayes a la hora de calcular estas probabilidades.

$$\boldsymbol{P}(\mathbf{A}|\mathbf{B}) = \boldsymbol{P}(\mathbf{A})\frac{\boldsymbol{P}(\mathbf{B}|\mathbf{A})}{\boldsymbol{P}(\mathbf{B})} \tag{4.1}$$

Donde, para nuestro problema, 'A' se refiere a la hipótesis planteada y 'B' se refiere a la información probabilista obtenida del tipo de región para una celda.

A $\boldsymbol{P}(\mathbf{A})$ se le da el nombre de 'prior', es decir, como de probable es nuestra hipótesis antes de observar la información probabilista obtenida del entorno.

$\boldsymbol{P}(\mathbf{B}|\mathbf{A})$ es el 'likelihood', es decir, la probabilidad de que la información probabilista obtenida sea correcta dada nuestra hipótesis.

$\boldsymbol{P}(\mathbf{A}|\mathbf{B})$ es el 'posterior'. Se refiere a la probabilidad de que nuestra hipótesis sea correcta dada la información probabilista.

Por último, $\boldsymbol{P}(\mathbf{B})$ se refiere a la probabilidad de la información probabilista. Esta es calculada a partir del producto entre el 'likelihood' y el 'prior'.

Debido a que el entorno real es fijo, es decir, no cambia a lo largo de la ejecución del programa, el valor de 'posterior' para una iteración i se convertirá en el valor 'prior' para la iteración i+1. De esta forma conseguimos que la información se retroalimente, obteniendo a cada iteración información más precisa del entorno. El valor de 'prior' para la primera iteración en todas las celdas se trata de un conjunto de probabilidades repartidas equitativamente para todos los tipos de región, es decir, si tuviésemos 4 tipos de región, el 'prior' de la primera iteración para todas las celdas se trataría del conjunto de probabilidades [0.25, 0.25, 0.25, 0.25].

Como ya se ha comentado para cada tipo de algoritmo de estimación se utiliza este teorema para conocer la estimación del entorno de cada robot en cada iteración del programa, sin embargo, cada tipo de algoritmo calcula de manera diferente el valor de los

'likelihoods', es decir, el valor de la probabilidad de que la información probabilista del tipo de región de cada celda sea correcta según la hipótesis.

Cuando se obtenga el valor de 'posterior' para una celda del robot R este es asignado como las nuevas probabilidades de dicha celda del robot R. El tipo de región con la mayor probabilidad es el tipo de región estimado para esa celda. Si la máxima probabilidad coincide con la probabilidad para el tipo de región de espacio libre entonces el tipo de región estimado para la celda es el espacio libre.

4.1.1. Algoritmo independiente

Como su nombre indica, este algoritmo hace que los robots obtengan la información probabilista de forma independiente al resto de robots, por lo que a la hora de estimar el entorno supondremos que estos robots no comparten la información probabilista del tipo de regiones, independientemente de su radio de comunicación.

El valor del 'likelihood' para cada celda del mapa de creencias del robot R se obtiene a partir de la estimación que haya hecho este robot R del entorno según su sensor. Se asume que los robots conocen la seguridad de sus sensores a la hora de realizar una estimación del entorno.

4.1.2. Algoritmo centralizado

En este algoritmo se asume que, independientemente del radio de comunicación de los robots, todos los robots tienen la misma estimación del entorno, siendo así todos sus mapas de creencias iguales.

El valor del 'likelihood' en este caso se trata del producto de las probabilidades obtenidas de cada celda para cada robot según su sensor. De esta forma obtenemos un mismo mapa de creencias para todos los robots donde el tipo de región para cada celda es aquella región con la mayor probabilidad según todos los robots.

4.1.3. Algoritmo de consenso

En este algoritmo los robots pueden compartir información probabilista entre ellos solo si estos se encuentran a una distancia inferior o igual a su radio de comunicación.

El valor del 'likelihood' para cada celda del mapa de creencias del robot R se obtiene gracias a un consenso de las creencias del robot R y las de los robots que puedan comunicarse con R, es decir, se encuentren a una distancia de este menor o igual al radio de comunicación.

De esta forma si el radio de comunicación de los robots es un valor suficientemente grande como para que todos los robots puedan comunicarse entre sí este algoritmo se asemejará al algoritmo centralizado; si por el contrario el radio de comunicación resulta tan pequeño como para que ningún robot pueda comunicarse entre sí entonces este algoritmo se asemejará más al algoritmo independiente.

Capítulo 5

Mejoras implementadas

Como se ha comentado, parte del código utilizado para poder implementar este problema en la herramienta RMTool viene de una implementación realizada en un grupo de investigación del Departamento de Informática e Ingeniería de Sistemas de la Universidad de Zaragoza.

Sin embargo, para este trabajo, además de integrar esta implementación en RMTool, también se realizaron una serie de mejoras.

En este apartado se comentan las mejoras introducidas más importantes.

5.1. Introducción de datos por parte del usuario

Dado que en la anterior implementación todos los datos eran constantes, todas las ejecuciones del programa tenían un entorno con las mismas dimensiones, el mismo número de tipos de región, el mismo número de regiones para cada tipo de región, el mismo número de robots, la misma fórmula booleana indicando la misión a alcanzar, etc..., donde la única diferencia entre ejecuciones consistía en la posición de estas regiones y la posición de los robots, al ser estas aleatorias.

Al ser esto un factor limitante a la hora de realizar distintas pruebas, se decidió implementar una mejora haciendo que estos datos pasasen a ser datos que pudiesen ser modificados por el usuario.

Con esta mejora se hizo posible que el usuario introdujese valores para los datos de la tabla 5.1.

Dato	Valor por defecto
Dimensiones del entorno	20x10
Número de robots	3
Número de tipos de regiones de interés	4
Número de regiones para cada tipo de región	$[\mathbf{2}_{y_1}, \mathbf{2}_{y_2}, \ldots, \mathbf{2}_{y_{NumInterés+1}}]$
Fórmula booleana	$!\boldsymbol{y_1}$ & $\boldsymbol{y_2}$
Tipo de estimación	3 (consenso)
Radio de comunicación de los robots	25
Tau	0.75
Tau obs	0.25
MAXITERS	100

Tabla 5.1: Datos que pueden ser modificados por parte del usuario.

Donde `NumInterés` es el número de tipos de regiones de interés.

Cómo modificar todos estos datos dentro de la herramienta RMTool se encuentra explicado en el Anexo B.

5.2. Creación del entorno de forma manual

Otra mejora con respecto a la anterior implementación fue el añadir que los entornos pudiesen ser creados de forma manual por el usuario, en lugar de ser siempre aleatorios.

De esta forma el usuario a la hora de generar un entorno será avisado por el programa para que elija cómo quiere que se genere este, aleatoriamente o de forma manual.

Si el usuario elige la forma manual el programa preguntará al usuario el número de regiones de tipo 1, tras introducir este número el usuario podrá seleccionar las celdas del entorno que quiere que sean de este tipo haciendo clic izquierdo o derecho sobre estas, tras esto se le volverá a pedir el número de regiones y el seleccionar las celdas, esta vez para el tipo 2, así hasta llegar al tipo `NumInterés + 1` (las regiones de tipo obstáculo).

Tras esto el programa pedirá al usuario que introduzca las posiciones iniciales de los robots una a una haciendo clic con el botón derecho del ratón sobre el entorno.

Los pasos a seguir para generar un entorno de forma manual se encuentran en el Anexo B.

5.3. Animar el movimiento de los robots

Hasta ahora la implementación anterior solo mostraba el estado final de ejecutar el *Path Planning* sobre un entorno desconocido a priori por los robots.

Con esta mejora se pretende mostrar las posiciones reales de todos los robots en el entorno real y las posiciones estimadas de cada robot en sus respectivos mapas de creencias en cada iteración de la ejecución del *Path Planning*, donde la posición de cada robot viene marcada por un punto de un color y un número que indica el número del robot.

De esta forma se consigue la sensación de movimiento por parte de los robots, ayudando al usuario a entender de forma visual lo que está ocurriendo durante la ejecución.

Tras la terminación del *Path Planning* se muestra sobre el entorno real la ruta seguida por parte de cada robot gracias a un conjunto de líneas coloreadas según el número del robot.

5.4. Movimiento a zonas sin explorar

Como ya se ha mencionado en anteriores apartados, para cada iteración del programa, a la hora de ejecutar el *Path Planning*, hace falta resolver un problema de optimización de tipo MILP. La idea de esta mejora parte sobre la duda de qué hacer en las iteraciones donde el programa no encuentre una solución a este problema de optimización.

La implementación anterior que solucionaba esta duda consistía en que el robot sin solución a este problema MILP realizase un movimiento aleatorio a cualquiera de sus cuatro celdas vecinas. Sin embargo, esta posible solución tenía el problema de no ser una solución óptima a lo que explorar el entorno se refiere, ya que estos casos donde el programa no encuentra una solución al problema MILP se deben principalmente a la falta de información sobre el entorno, por lo que se debería priorizar realizar movimientos que nos permitan alcanzar este fin, por ejemplo, realizando movimientos a zonas del mapa que aún no hayan sido exploradas.

Este último ejemplo es el que se utilizó como propuesta para poder mejorar esta parte del programa.

La implementación de esta mejora se puede explicar en dos fases, siendo la primera el saber la zona del mapa a la cual ir y la segunda saber la dirección a movernos para poder llegar a la zona previamente mencionada. A continuación se explican estas dos fases.

5.4.1. Saber a qué zona del mapa ir

Para saber hacia qué zona sin explorar ir lo primero que se realizó fue averiguar qué celdas no habían sido aún exploradas por cada robot. Para hacer esto se hizo la suposición de que aquellas celdas donde la probabilidad máxima del tipo de región estimado fuese mayor a un cierto valor ya se encontraban exploradas, por lo que aquellas con una probabilidad máxima menor o igual a este valor serían consideradas celdas no exploradas (en este caso el valor elegido fue `2/NumTipos`, donde `NumTipos` se refiere al número de tipos de regiones). Cabe destacar que al no existir una unidad central que se comunique con todos los robots el conjunto de celdas que se consideren no exploradas para un robot puede no ser igual al conjunto de celdas no exploradas de otro. En caso de que los robots se encuentren a una distancia menor o igual a su radio de comunicación, estos podrán compartir la información probabilista del entorno respecto a los tipos de región estimados, de esta forma se obtienen unas probabilidades de estimación más exactas del entorno y, por consiguiente, un conjunto de celdas sin explorar más preciso, pudiendo evitar casos donde un robot decida visitar una zona del mapa ya explorada por otro.

Con las celdas no exploradas ya obtenidas se pasó a pensar a qué zona debería ir el robot para la exploración del mapa. Para esto se crearon dos posibles implementaciones:

Moverse a la celda donde se encuentre la posición media de todas las celdas no exploradas.

Esta implementación es útil ya que permite a los robots llegar a la zona con mayor densidad de celdas no exploradas con el menor número de movimientos.

Sin embargo, existe un problema con esta estrategia, siendo este que en el caso de que la posición media de todas las celdas no exploradas sea una posición cercana a la posición del robot este apenas se moverá de su sitio, provocando que, si en las siguientes iteraciones tampoco se puede resolver el problema MILP que cumpla la fórmula booleana, el robot se quede en esta misma zona por un tiempo indefinido de iteraciones.

Es por este problema que se descartó esta implementación y se utilizó la siguiente.

Moverse a la celda no explorada más cercana con respecto a la posición actual del robot.

Esta implementación evita el problema de la implementación anterior, al ser las celdas no exploradas distantes a la posición del robot, siendo la celda de destino en cada iteración distante con respecto a la posición actual del robot.

Nótese que esta implementación puede obtener más de una celda destino a explorar

debido a que varias celdas no exploradas puedan tener la misma distancia respecto a la celda donde se encuentra el robot a mover.

5.4.2. Obtener la dirección a moverse

Una vez elegida la celda o conjunto de celdas posibles a las cuales ir, se realizó la implementación para averiguar qué movimiento debía hacer el robot en esta iteración (arriba, abajo, izquierda o derecha).

Para ello se implementó el algoritmo de búsqueda en anchura [9] (Breadth-first search o BFS) en Matlab. Este algoritmo es útil para nuestro caso ya que siempre encontrará la celda destino con el menor número de movimientos posibles. Esto se consigue gracias a que este algoritmo comienza explorando todas las celdas vecinas respecto a una celda inicial (en nuestro caso la celda donde se encuentra el robot), tras explorar todas estas celdas se exploran sus respectivas celdas vecinas (sin explorar de nuevo las celdas ya exploradas anteriormente), así hasta encontrarse con la celda destino (en nuestro caso la celda no explorada más cercana).

Para el caso donde tengamos más de una celda destino posible a explorar el algoritmo se ejecutará hasta encontrar la celda de este conjunto con la menor distancia, en caso de empate la celda destino a explorar será aquella que haya encontrado antes el algoritmo.

A continuación, se muestra un ejemplo en la figura 5.1 para una iteración donde el programa no encuentra una solución para el problema, haciendo que el robot tenga que moverse a la celda más cercana no explorada.

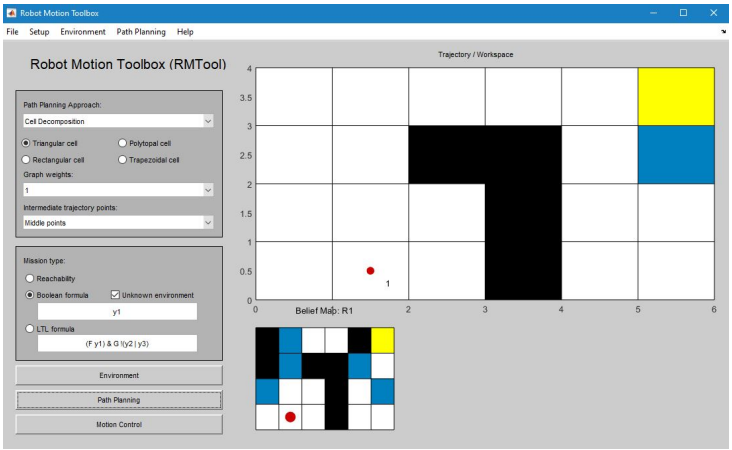

Figura 5.1: Iteración en la cual el robot debe moverse de la celda 2 a la celda más cercana sin explorar en un entorno 4x6 según su información probabilista del tipo de regiones, la cual se puede observar en su mapa de creencias.

Para saber las celdas sin explorar más cercanas y la dirección a la que ir para poder

llegar a aquella con el menor número de movimientos nos fijaremos en el mapa de creencias del robot.

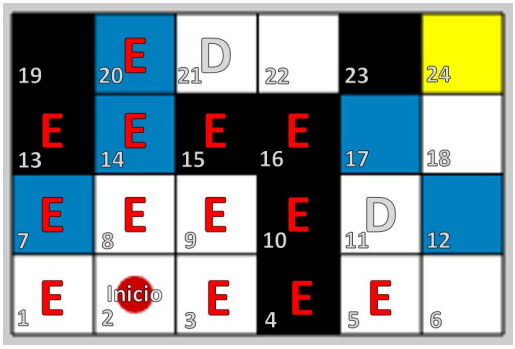

Figura 5.2: Mapa de creencias del robot para la iteración de la figura 5.1, donde las celdas con la letra 'E' y la celda de inicio se consideran celdas ya exploradas y las celdas con la letra 'D' se consideran las celdas destino a visitar. La celda destino alcanzada con el menor número de movimientos se considera la celda a la que el robot tendrá que ir.

Como se puede observar en la figura 5.2 las celdas sin explorar más cercanas se tratan del conjunto 11, 19 y 21; esto es debido a que sus probabilidades máximas no superan el valor de `2/NumTipos` (en este caso 2/4, el 50 %). Sin embargo, en este mapa de creencias el robot considera la celda 19 como una región de tipo obstáculo, por lo que no se considerará como posible celda destino, dejando así el conjunto de posibles celdas destino como las celdas 11 y 21.

Así pues, ejecutaremos el algoritmo de recorrido en anchura para este mapa de creencias. A continuación, se muestra en la figura 5.3 el grafo en forma de árbol generado por este algoritmo, donde cada número de nodo representa el número de celda (estos números de celdas pueden verse en la figura 5.2). Este grafo se ha dibujado gracias a una herramienta de la página web 'csacademy' [10].

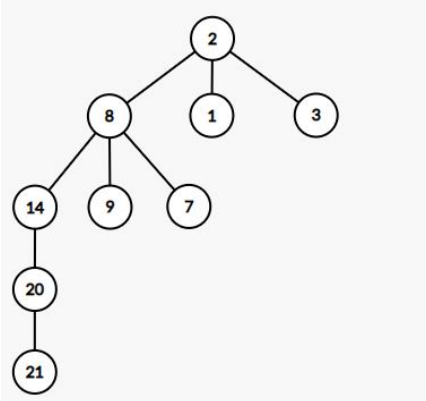

Figura 5.3: Grafo en forma de árbol generado por el algoritmo de recorrido en anchura para el ejemplo de la figura 5.2.

Además, en la figura 5.4 se muestran los resultados devueltos por este algoritmo, siendo estos la ruta para llegar a la celda de destino que requiera el menor número de movimientos y el número de movimientos realizados.

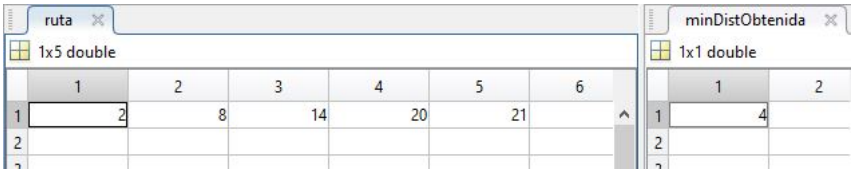

Figura 5.4: Resultados devueltos por el algoritmo de recorrido en anchura para el ejemplo de la figura 5.2.

5.5. Identificar cuando se ha cumplido la misión

A la hora de hacer que la ejecución del *Path Planning* terminase, en la anterior implementación solo se tenía en cuenta el número de iteraciones realizadas, terminando la ejecución tras haber alcanzado un número arbitrario de iteraciones, en este caso 50.

Con esta otra posibilidad se ha pretendido hacer que el *Path Planning* terminase si y solo si los robots han cumplido la misión. De esta forma se evitaría tener que esperar al número arbitrario de iteraciones para hacer que la ejecución del *Path planning* terminase, en el caso de que los robots cumpliesen la misión antes, y también se evitaría el caso de terminar el *Path planning* antes de que los robots pudiesen cumplir su misión. Un dato importante a mencionar es que esta nueva posibilidad de hacer que el programa termine supone una comunicación centralizada entre los robots para saber si estos han cumplido la misión. Por otra parte, debido a posibles casos que fuesen irresolubles se tuvo que indicar que la ejecución terminase también para el caso donde se llegase a un número máximo de iteraciones para asegurar la finalización de la ejecución, siendo este número máximo de iteraciones el valor del parámetro `MAXITERS` introducido por el usuario.

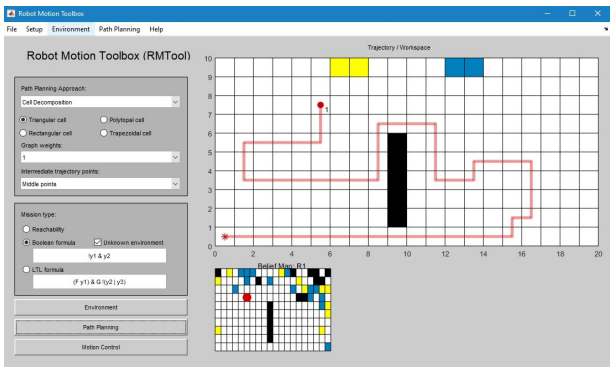

Figura 5.5: *Path planning* finaliza al llegar a las 50 iteraciones.

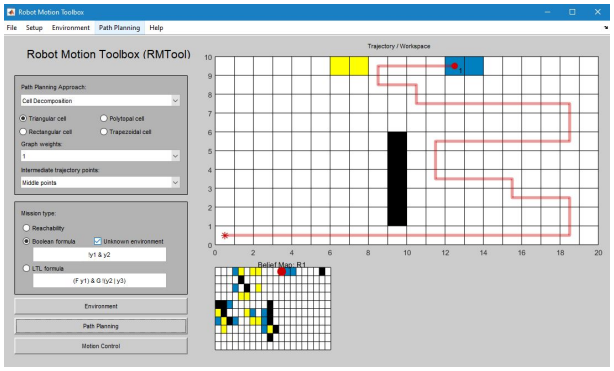

Figura 5.6: *Path planning* finaliza cuando los robots cumplen con la misión.

En las figuras 5.5 y 5.6 se muestra el resultado de haber implementado esta nueva posibilidad. Como se puede observar en la figura 5.6, el *Path Planning* ha finalizado tras pasar el número arbitrario de iteraciones, el cual en este caso eran 50, y con el robot cumpliendo la misión indicada por la fórmula booleana $!y_1 \ \& \ y_2$.

Capítulo 6

Evaluación

En este capítulo se comentan las pruebas realizadas junto con los resultados obtenidos para cada una de estas. El objetivo es descubrir cómo influyen diferentes parámetros a la hora de conseguir que los robots puedan alcanzar la misión propuesta. A continuación se comentan las diferentes pruebas realizadas.

6.1. Radio de comunicación

Como ya se ha comentado, este parámetro sirve para indicar la distancia en la que los robots puedan comunicarse para compartir información probabilista del entorno. Esta información a compartir se trata de la información probabilista del tipo de regiones de las celdas y la posición donde se encuentra realmente el robot con el que nos comunicamos.

Esto es importante debido a que, para casos donde este parámetro sea nulo o tan pequeño como para que los robots apenas se puedan comunicar, la información probabilista para cada robot del tipo de regiones será independiente al resto de robots (a excepción de utilizar el algoritmo centralizado, el cual supone que los robots pueden comunicarse siempre la información del tipo de regiones de las celdas para que todos tengan la misma información del tipo de regiones) y la información de la posición de los otros robots para un robot R será una información estimada, haciendo posible que este robot R crea que los otros robots se encuentran en una posición en la que realmente no están.

Esto último puede dar lugar a situaciones donde todos los robots crean haber cumplido con la misión según sus mapas de creencias cuando en el mapa real esto no sucede, en la figura 6.1 se muestra un ejemplo de esta situación.

Para este ejemplo se utilizaron 3 robots, donde la misión a alcanzar por estos se encontraba definida por la fórmula booleana 'y_1 & y_2 & y_3'. El radio de comunicación de los

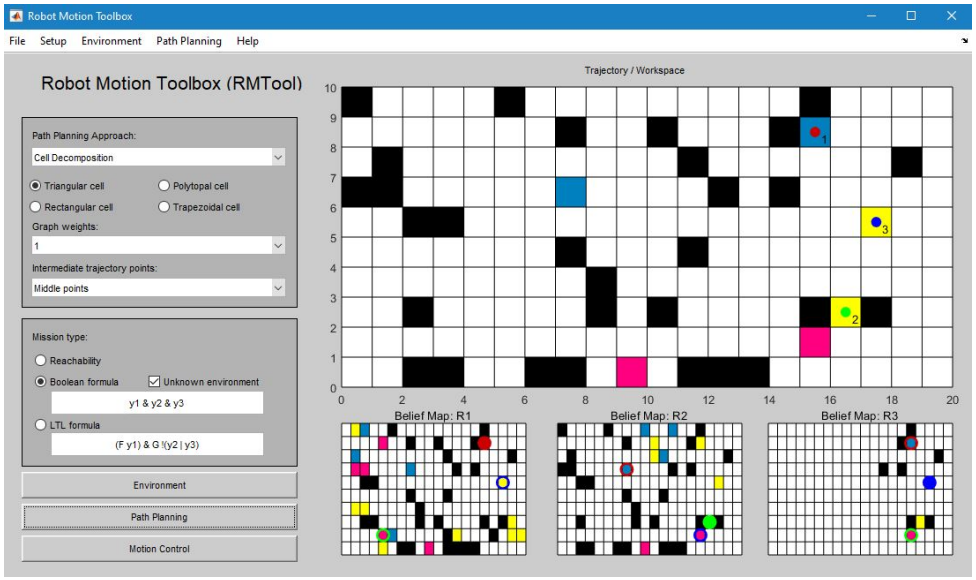

Figura 6.1: Estado final de la ejecución del *Path planning* en un entorno 10x20 con 3 robots donde no se ha podido cumplir la misión de ir a las celdas de color amarillo, azul y rosa según la fórmula booleana 'y_1 & y_2 & y_3' debido a la falta de comunicación por parte de los robots.

robots fue definido como nulo, 0, evitando así la comunicación de información probabilista entre robots. El valor de la probabilidad de que se cumpliese la misión booleana fue de 0.95 (95 %) y la probabilidad para suponer que una celda se trataba de un obstáculo fue de 0.25 (25 %). Aquí el programa supone que se ha cumplido la misión ya que, para todos los mapas de creencias de los robots se cumple la misión; sin embargo, en la realidad esto no ocurre. Esto pasa debido a que al no existir comunicación entre los robots estos han debido suponer las posiciones del resto de robots de manera errónea, además de suponer también de forma errónea el tipo de celdas donde estos se encuentran.

Para solucionar este problema se realizaron varias ejecuciones del *Path planning* sobre este mismo entorno cambiando el valor del radio de comunicación de 0 a 25, es decir, un valor suficiente para asegurar que todos los robots pueden comunicarse entre sí, de esta forma estos robots no tienen que estimar la posición de los demás robots, ya que al encontrarse todos dentro del radio de comunicación estos demás robots pueden compartir su posición real. Con esto se observó que en todas las ejecuciones realizadas los robots cumplían realmente con su misión, lo cual tiene sentido al ser las posiciones del resto de robots siempre las mismas a la realidad y al conocer cada robot el tipo de región real donde se encuentran. En la figura 6.2 se muestra un ejemplo de una iteración para el mismo entorno donde todos los valores pasados son iguales a excepción del radio de comunicación, siendo este de 25.

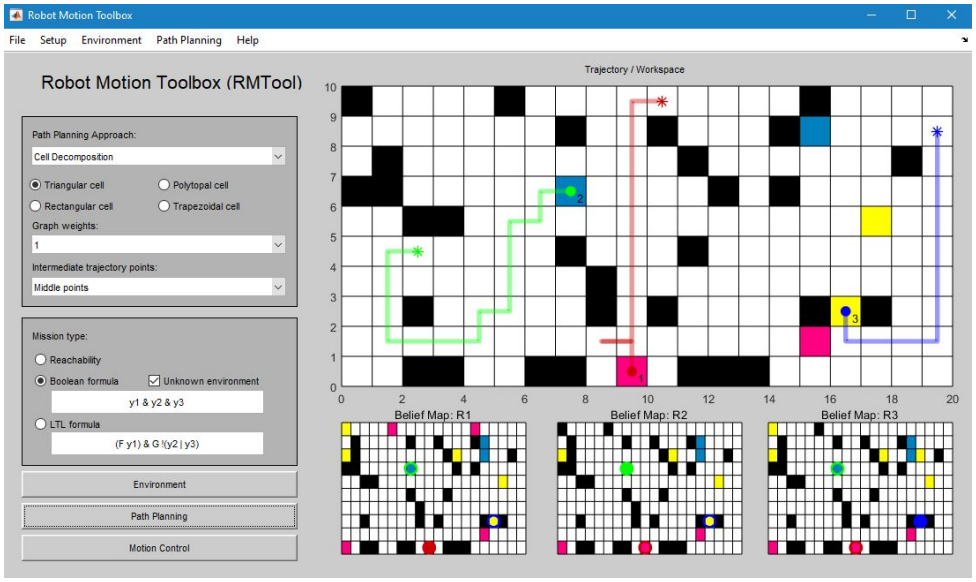

Figura 6.2: Estado final de otra ejecución sobre el mismo entorno donde sí se cumple la misión de ir a las celdas de color amarillo, azul y rosa según la fórmula booleana 'y_1 & y_2 & y_3' gracias a cambiar el radio de comunicación a 25 para que los robots puedan comunicarse.

6.2. Movimientos a zonas sin explorar

A continuación se muestra una prueba que muestra la comparativa entre hacer que los robots se muevan de forma aleatoria en el caso de no encontrar una solución al problema de optimización MILP y hacer que estos se muevan a las zonas sin explorar más cercanas.

Para esto se ha creado el entorno de las figuras 6.3 y 6.4. Este entorno tiene en su parte derecha un pequeño laberinto el cual tiene que recorrer un robot con el fin de salir de este y cumplir la misión indicada por la fórmula booleana '!y_1 & y_2'. Además, el parámetro para indicar con qué probabilidad se tiene que cumplir la misión se ha establecido con un valor de 0.95 (95 %), de esta forma el problema MILP no encontrará una solución la mayoría de las veces, forzando al robot a tener que moverse según la implementación anterior (moverse aleatoriamente) o según la implementación mejorada (moverse a las zonas sin explorar más cercanas).

Figura 6.3: Ejecución donde los robots se mueven de forma aleatoria si el problema MILP no encuentra solución.

Figura 6.4: Ejecución donde los robots se mueven a las zonas sin explorar más cercanas si el problema MILP no encuentra solución.

Para estas figuras, la implementación donde los robots se mueven de forma aleatoria en el caso de que el problema MILP no encuentre una solución ha tardado 490 iteraciones; por otro lado, la mejora a esta implementación donde los robots se mueven a las celdas sin explorar más cercanas ha tardado un total de 28 iteraciones. Con esto observamos cómo esta mejora permite a los robots cumplir una misión de forma más eficiente con respecto a la implementación anterior cuando el problema MILP no encuentra una solución.

6.3. Algoritmos de estimación

Las pruebas a realizar en este apartado tienen como finalidad saber qué tipo de algoritmo de estimación resulta ser el más eficiente a la hora de que el equipo de robots cumpla una misión con el menor número de movimientos posibles. Para ello se realizaron varias ejecuciones (en este caso 50) para cada algoritmo de estimación con el fin de hallar la media y la desviación típica del número de iteraciones necesarias para cumplir una misión. El único parámetro a cambiar en cada conjunto de iteraciones ha sido el tipo de algoritmo de estimación utilizado, siendo los valores de todos los demás parámetros los valores por defecto.

6.3.1. Primera prueba

La primera de las pruebas se ha realizado sobre el entorno de la figura 6.5 con un solo robot el cual debe cumplir la misión definida por la fórmula booleana 'y_3' (llegar a la región de interés de color rosa).

Los resultados de las medias y desviaciones típicas para cada algoritmo de estimación aparecen en la tabla 6.1.

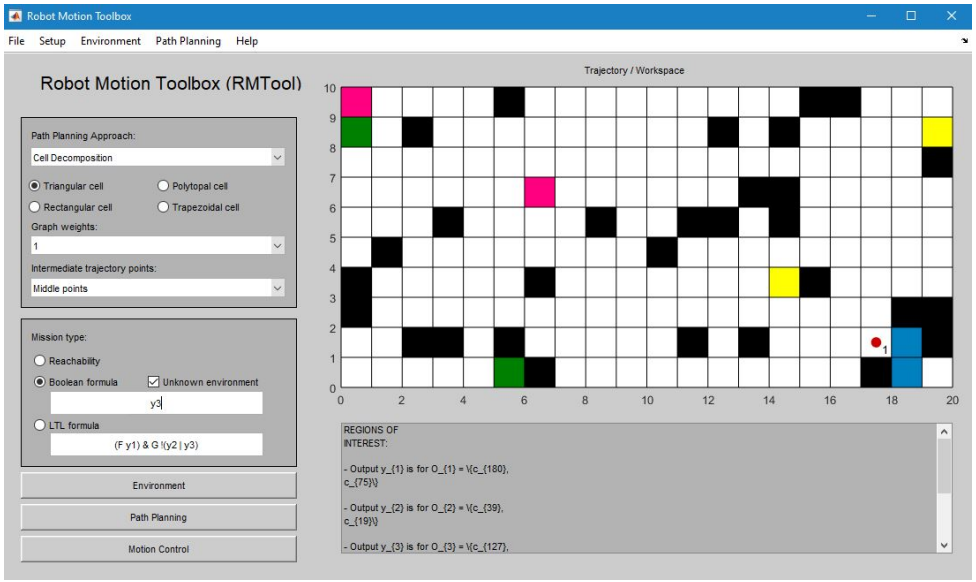

Figura 6.5: Entorno 10x20 utilizado para comparar la media y desviación típica de 50 ejecuciones para cada algoritmo de estimación con un solo robot cuya misión es ir a una celda de color rosa según la fórmula booleana 'y_3'.

	Independiente	**Consenso**	**Centralizado**
Media	29.86	29.84	29.22
Desviación	6.624	6.6742	5.9805

Tabla 6.1: Resultados obtenidos tras 50 ejecuciones del programa para cada algoritmo de estimación en un entorno de un solo robot con una misión booleana formada por una meta (y_3).

Con esto observamos que para casos donde exista solamente un robot que tenga que cumplir una misión, no importa el algoritmo de estimación a utilizar, al ser todas las medias y desviaciones muy similares. Esto se debe a que no existe una comunicación entre robots que permita compartir información probabilista, haciendo así que todos los algoritmos se comporten como el algoritmo independiente.

6.3.2. Segunda prueba

Para esta segunda prueba se ha utilizado el entorno de la figura 6.6 con más de un solo robot, en este caso 3, los cuales deben cumplir la misión 'y_1 & y_2 & y_3' (llegar a las regiones de interés de color amarillo, azul y rosa).

Los resultados se pueden observar en la tabla 6.2.

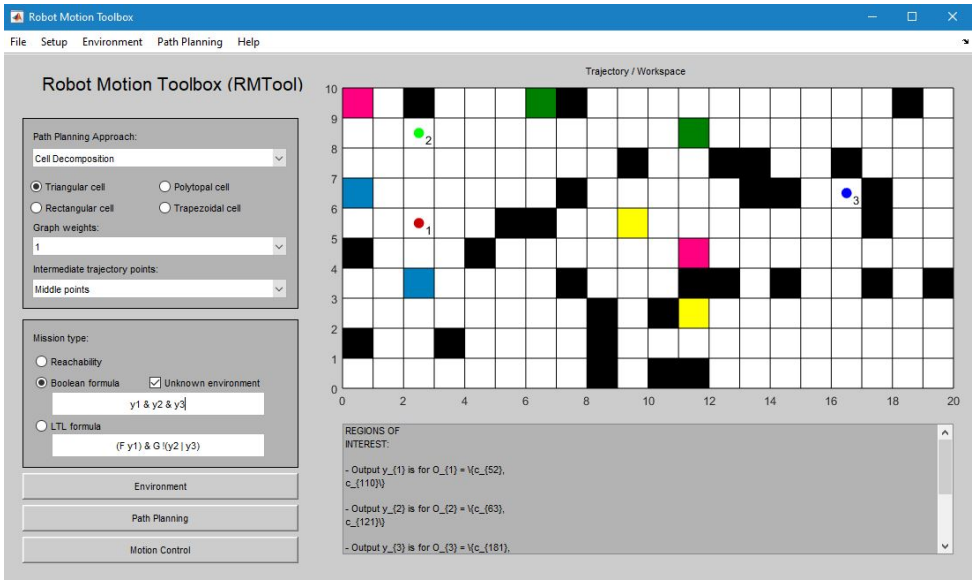

Figura 6.6: Entorno 10x20 utilizado para comparar la media y desviación típica de 50 ejecuciones para cada algoritmo de estimación con tres robots cuya misión es ir a una celda de color amarillo, otra azul y otra rosa según la fórmula booleana 'y_1 & y_2 & y_3'.

	Independiente*	**Consenso**	**Centralizado**
Media	70.938*	14.86	15.22
Desviación	19.689*	5.083	4.244

Tabla 6.2: Resultados obtenidos tras 50 ejecuciones del programa para cada algoritmo de estimación en un entorno de tres robots con una misión booleana formada por tres metas (y_1 & y_2 & y_3).

Se ha supuesto que los robots no han podido cumplir la misión en el caso de que el número de iteraciones llegue a 100. Esto solamente ha ocurrido en algunas ejecuciones del algoritmo independiente, en concreto se ha dado en 18 de las 50 ejecuciones (el 36 %), por lo que se ha calculado la media a partir del resultado de las otras 32 ejecuciones que sí han podido cumplir la misión. Que esto haya ocurrido y que la media del número de iteraciones sea mucho más alta respecto a los otros dos algoritmos se debe a que en este algoritmo los robots conocen con cierta seguridad solo el tipo de regiones por las que hayan pasado, por lo que, aunque estos sepan de forma exacta la ubicación del resto de robots gracias a que el radio de comunicación por defecto de 25 permita que siempre exista comunicación con todos los robots, estos pueden no conocer con seguridad el tipo de región donde se encuentran los otros robots debido a que es posible que no hayan visitado aún esas zonas.

Por otra parte, los otros dos algoritmos tienen unos resultados parecidos ya que ambos permiten la comunicación con todos los robots, permitiendo que todos compartan la información probabilista del tipo de regiones y su posición exacta con el resto de robots.

Dicho esto, podemos indicar que para casos donde podamos tener una comunicación con todos los robots en un entorno no importa si elegimos como algoritmo de estimación el algoritmo centralizado o independiente. Sin embargo, para casos donde no podamos tener siempre una comunicación con todos los robots tendremos que elegir el algoritmo de consenso, ya que el centralizado supone que siempre existe esta comunicación con todos los robots.

6.3.3. Tercera prueba

Para esta última prueba se ha probado a modificar el valor del radio de comunicación al utilizar el algoritmo de consenso con los valores 5, 10 y 25, utilizando como resultados para el radio 25 los mismos observados en la anterior prueba.

	Radio 5	**Radio 10**	**Radio 25**
Media	33.04	25.26	14.86
Desviación	10.041	7.174	5.083

Tabla 6.3: Resultados de las 50 ejecuciones del programa utilizando el algoritmo de consenso cambiando el radio de comunicación por los valores 5, 10 y 25. El entorno y misión que se han utilizado para esta prueba son los mismos que los de la figura 6.6.

Como se puede observar en la tabla 6.3 a mayor radio de comunicación menor es el número de iteraciones. Esto tiene sentido ya que cuanto mayor sea este radio más probable será que los robots se encuentren dentro de este radio con otros y puedan compartir información probabilista del entorno, pudiendo así obtener información más precisa del entorno con el fin de encontrar las trayectorias que permitan a los robots cumplir la misión.

También vemos como el algoritmo independiente y centralizado se tratan de dos casos extremos, donde el primero al suponer que no existe comunicación con los robots estos no pueden compartir información de su entorno, cumpliendo así la meta en un número de iteraciones mayor al resto de algoritmos para el caso de que existan varios robots que tengan que cumplir una misión definida por varias metas. Por otro lado, el algoritmo centralizado supone una comunicación centralizada de los robots, permitiendo en todo momento compartir la información del entorno, haciendo así que los robots siempre tengan la información más precisa del entorno, permitiendo de esta manera que estos cumplan la misión en el menor número de iteraciones posible.

Capítulo 7

Conclusiones

Tras la realización de este trabajo podemos indicar que se han realizado todos los objetivos propuestos, siendo estos la implementación y mejora en la herramienta RMTool de un programa capaz de simular trayectorias para equipos de robots en un entorno distribuido, además del estudio de problemas de optimización y algoritmos de búsqueda, siendo estos el problema de tipo MILP y el algoritmo de recorrido en anchura utilizados para obtener el próximo movimiento a realizar por parte de los robots.

7.1. Líneas de futuro

A continuación se presentan dos posibles líneas de futuro que sirvan para expandir el trabajo realizado en este trabajo de fin de grado.

7.1.1. Mejora en la estimación de posiciones de los robots

La primera línea de futuro tiene que ver con la mejora de las posiciones estimadas por un robot del resto del equipo. Como ya se ha visto, para casos donde los robots tengan un radio de comunicación muy limitado estos no pueden comunicarse con los otros robots, impidiendo obtener información probabilista del entorno y las posiciones reales de los demás robots. El que estos robots no puedan comunicarse entre ellos implica que estos deben estimar las posiciones donde se encuentran el resto de robots, sin embargo, si estas estimaciones resultan ser erróneas podríamos llegar a estados donde los robots crean haber cumplido con su misión cuando en la realidad esto no es así. La idea de esta línea a futuro es intentar evitar esto realizando una implementación que, en caso de ser posible, permita a todos los robots estar comunicados en todo momento por lo menos con otro robot. De esta forma evitaríamos casos donde los robots no puedan comunicarse,

obligándoles a tener que estimar las posiciones del resto de robots, las cuales pueden ser erróneas y pueden llevar a casos donde la misión no se cumpla.

7.1.2. De simulación a realidad

La otra línea de futuro consiste en llevar esta simulación a la realidad, con robots reales que hagan una estimación del entorno gracias a sensores y puedan comunicarse con otros robots a través de dispositivos, de forma que no sea necesaria una unidad central que indique a los robots hacia donde moverse con el fin de cumplir cierta misión. Esta línea puede ser interesante ya que hasta ahora los trabajos de la Universidad de Zaragoza que implicasen el movimiento de unos robots desde un sitio a otro con el fin de cumplir una misión requerían de un sistema central, en este caso una cámara conectada a un ordenador, todo esto situado en uno de los laboratorios del edificio Ada Byron de la Universidad de Zaragoza, el cual recogía la información del entorno e indicaba a los robots como moverse. Con esta línea se podría evitar el tener que utilizar este sistema central, haciendo así que el movimiento de estos robots no se vea limitado al entorno recogido por una cámara.

Apéndice A

Guía de instalación

Se presenta una guía con los pasos a seguir para poder instalar todas las herramientas y bibliotecas utilizadas en este trabajo.

Antes de esto se presenta la información de versiones:

Matlab:

- 9.9.0.1570001 (R2020b) Update 4

Sistema operativo:

- Microsoft Windows 10 Pro Version 10.0

Java:

- Java 1.8.0_202-b08 with Oracle Corporation Java HotSpot(TM) 64-Bit Server VM mixed mode

Bibliotecas:

- **Optimization toolbox:** Versión 9.0
- **Mapping toolbox:** Versión 5.0
- **CPlex:** Versión 12.6.2

A.1. Matlab

El entorno de desarrollo Matlab puede ser descargado desde su página web oficial [11]. Para ello es necesario tener una cuenta de Matlab y una licencia. En caso de no tener una licencia una forma de obtenerla sería crear una cuenta de Matlab con el correo propio de la Universidad de Zaragoza, de esta forma podremos adquirir una licencia a través de la página web de la universidad de Zaragoza [12], la cual ofrece una licencia corporativa de Matlab para cuentas donde el identificador sea el correo de la universidad.

A.2. RMTool y CPlex

La descarga de la herramienta RMTool modificada para este trabajo puede ser encontrada en la nube accediendo a través de este enlace citado [13]. Al descargar y descomprimir este archivo ya tendremos descargada la herramienta RMTool con todas las modificaciones realizadas para este trabajo además de la biblioteca CPlex, la cual se encuentra en forma de carpeta con el nombre 'cplex'.

Para poder abrir la herramienta RMTool ejecutaremos el archivo 'rmt.m' de la carpeta 'RMTool'.

Esta herramienta RMTool posee su propia página web [14] desde la cual se puede acceder a su código fuente en GitHub [15] y a su documentación [16].

A.3. Optimization Toolbox y Mapping Toolbox

Una forma de descargar estas bibliotecas es abriendo el programa Matlab previamente instalado. Con Matlab abierto procedemos a seleccionar la parte superior de la figura 'Add-ons' de la pestaña 'Home'. Tras hacer esto se abrirá una nueva ventana, en esta ventana buscaremos estas bibliotecas introduciendo en el campo de texto donde pone 'Search for add-ons' los nombres 'Optimization Toolbox' y 'Mapping Toolbox'. Esto podemos observarlo en la figura A.1.

Una vez encontradas procederemos a su instalación haciendo clic sobre el botón de instalar, las bibliotecas encontradas deberían ser las de las figuras A.2 y A.3.

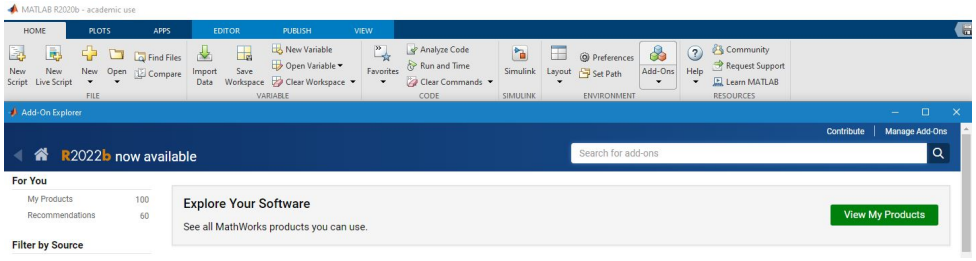

Figura A.1: Pasos para buscar las bibliotecas.

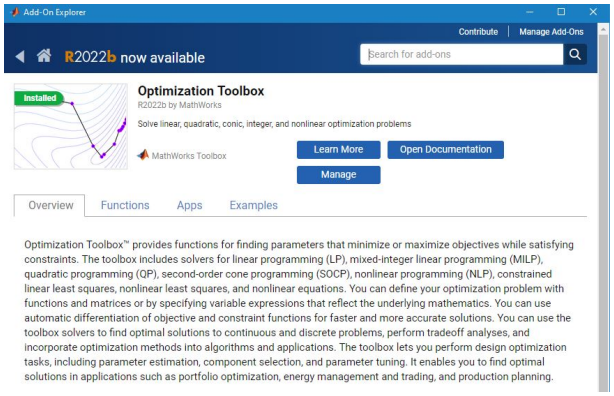

Figura A.2: Ventana donde poder instalar la biblioteca Optimization Toolbox.

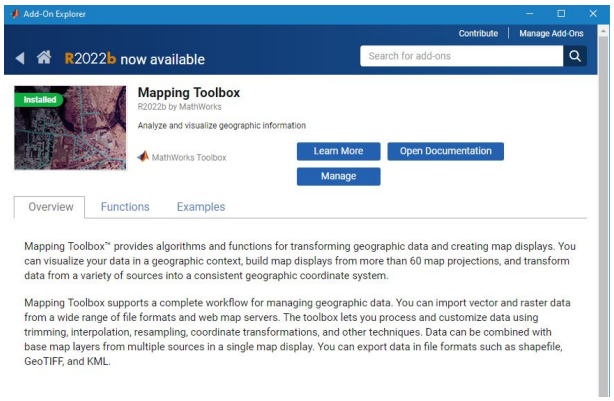

Figura A.3: Ventana donde poder instalar la biblioteca Mapping Toolbox.

Para que estas bibliotecas se instalen deberemos reiniciar la aplicación de Matlab.

Apéndice B

Robot Motion Toolbox

En este apartado se comentará la interfaz de la Robot Motion Toolbox o RMTool, a su vez que los pasos a seguir para poder realizar una correcta ejecución del problema abordado en este trabajo de fin de grado.

B.1. Interfaz

En la figura B.1 podemos observar cómo se muestra la interfaz de RMTool nada más ejecutar el programa 'rmt.m'.

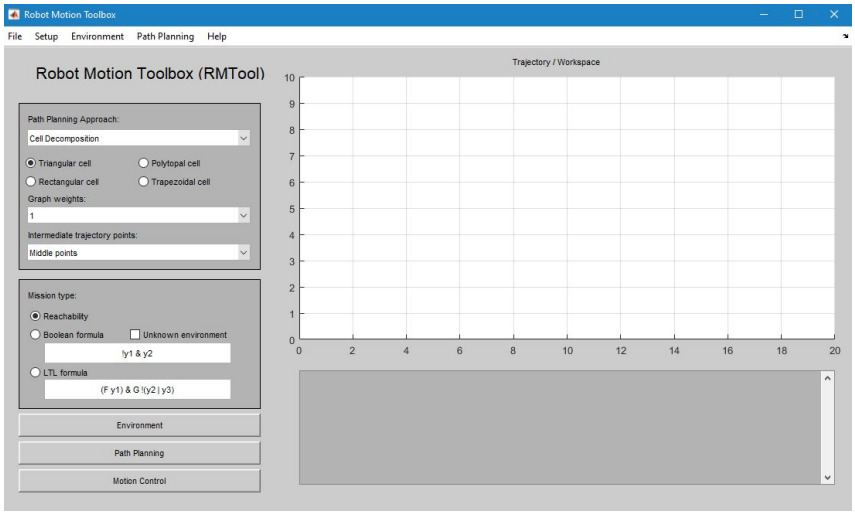

Figura B.1: Interfaz de RMTool.

A partir de esta figura B.1 se pasará a comentar cómo poder crear un entorno desconocido a priori por los robots. Además, también se comentará cómo hacer que estos

robots vayan desde una posición inicial hasta una posición final indicada por una fórmula booleana escrita por el usuario.

B.2. Creación del entorno

B.2.1. Definición de las dimensiones

Para poder crear un entorno el primer paso consistirá en definir las dimensiones de este.

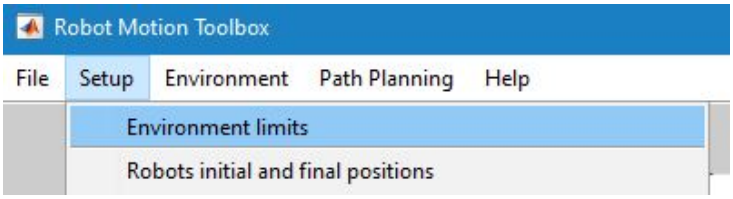

Figura B.2: Desplegable a clicar para poder modificar los valores de las dimensiones de un entorno.

Para hacer esto se deberá hacer clic en 'Setup »Environment limits', tal y como se muestra en la figura B.2.

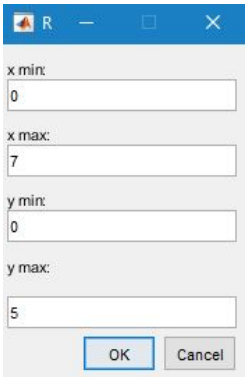

Figura B.3: Ventana donde introducir los valores de las dimensiones de un entorno.

Una vez hecho esto se abrirá una nueva ventana donde poder modificar las dimensiones por defecto. En la figura B.3 se muestra dicha ventana habiendo modificado los valores para que se cree un entorno de dimensiones 5x7.

Tras pulsar el botón 'Ok' veremos cómo las dimensiones de la gráfica central de RMTool han sido modificadas, tal y como se puede apreciar en la figura B.4.

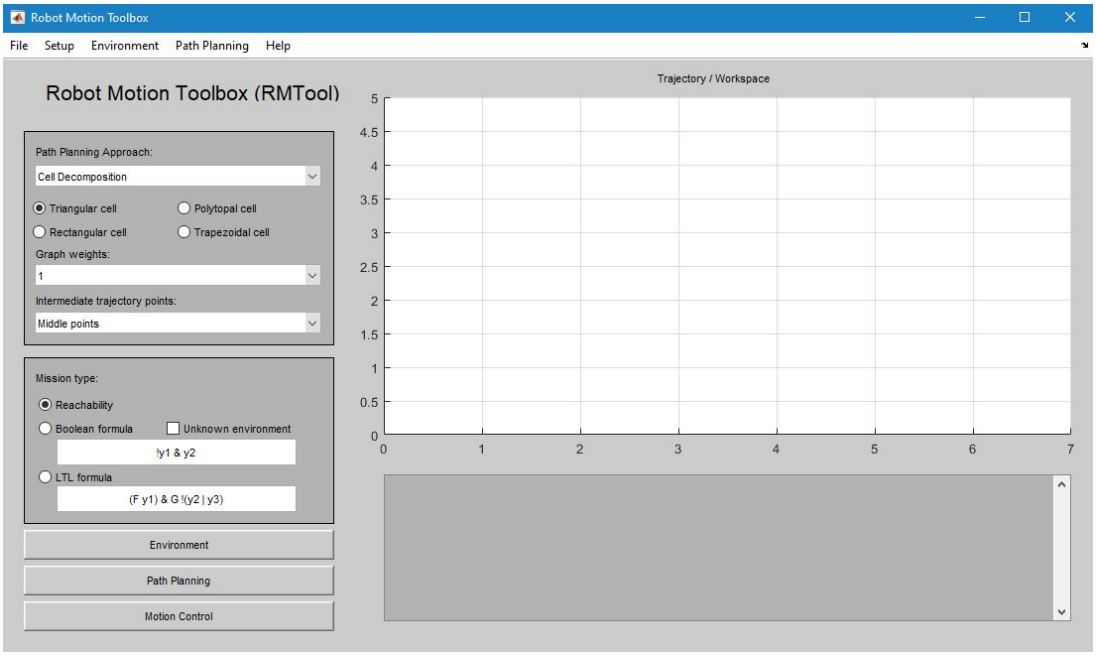

Figura B.4: Interfaz de RMTool tras cambiar las dimensiones del entorno a 5x7.

B.2.2. Inserción de las regiones y robots

Una vez definidas las dimensiones del entorno podremos pasar a insertar en este tanto las regiones como los robots.

Para ello daremos clic primero sobre el botón 'Unknown environment', de esta forma estaremos indicando que el entorno a crear será desconocido a priori por los robots. Con este botón activado pasaremos a dar clic sobre el botón 'Boolean formula' tal y como aparece en la figura B.5, así indicaremos que queremos crear un entorno donde los estados finales vienen definidos a partir de una fórmula booleana dada por el usuario (por ahora no será necesario modificar el campo de texto de la fórmula booleana que se encuentra debajo de los botones mencionados anteriormente, ya que esta será empleada más adelante cuando hagamos clic sobre el botón 'Path Planning' tras haber creado este entorno desconocido a priori por los robots).

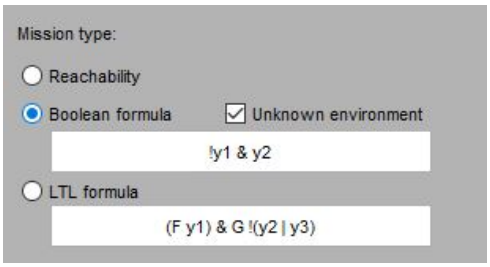

Figura B.5: Botones 'Boolean formula' y 'Unknown environment' activados.

Tras hacer clic en el botón 'Boolean formula' estando el botón 'Unknown environment' previamente activado aparecerá la ventana de la figura B.6 donde introducir el valor de dos variables, siendo estas el número de robots y el número de tipos de regiones de interés, en ese orden.

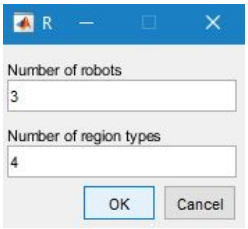

Figura B.6: Ventana donde introducir los valores del número de robots y el número de tipos de regiones de interés.

Con estos datos ya introducidos y tras darle al botón 'Ok' se abrirá una nueva ventana, esta vez con dos opciones para poder indicar si se quiere generar el entorno de forma manual o aleatoria, como se observa en la figura B.7.

Figura B.7: Ventana donde elegir la forma de generar el entorno.

Si elegimos la opción 'Random' a continuación se abrirán ventanas, una por cada tipo de región incluyendo los obstáculos, donde podremos indicar el número de regiones de cada tipo a insertar de forma aleatoria en nuestro entorno. La apariencia de estas ventanas se puede ver en la figura B.8.

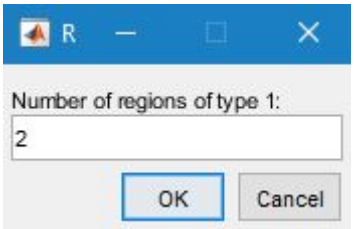

Figura B.8: Ventana donde introducir el número de regiones de cada tipo.

Tras introducir el número de regiones para cada tipo de región podremos observar como en la gráfica propia del entorno se ha dibujado un nuevo entorno de forma aleatoria siguiendo las especificaciones anteriores. Además, debajo de este entorno, aparecerá la información de las posiciones de todas las regiones en formato texto. Todo esto se puede observar en la figura B.9.

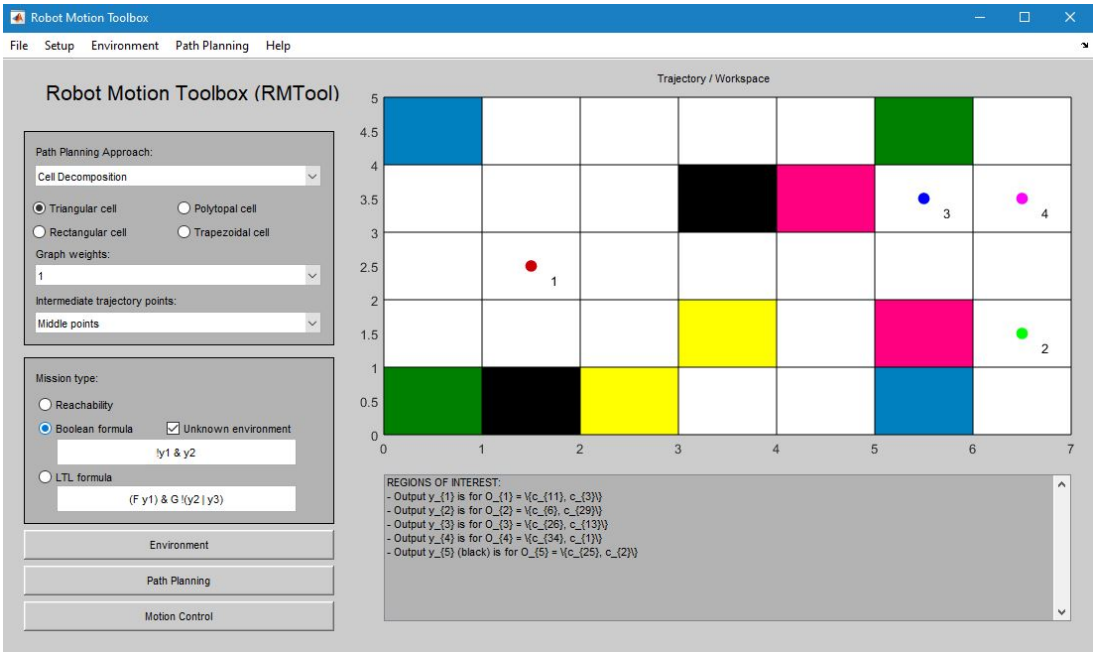

Figura B.9: Interfaz de RMTool tras generar un entorno desconocido de forma aleatoria.

Si por el contrario decidimos seleccionar la opción para generar el entorno de forma manual de la figura B.7 se nos indicará por medio de una ventana que podemos seleccionar celdas haciendo clic con el botón izquierdo o derecho del ratón sobre el entorno, tras esto se abrirá la ventana de la figura B.8 que pedía el número de regiones de cada tipo, sin embargo, ahora al seleccionar 'Ok' podremos hacer clic con el botón izquierdo o derecho del ratón sobre el entorno para seleccionar las celdas del tipo mencionado en la anterior ventana, esto se puede ver en la figuras B.10 y B.11. Tras seleccionar todas las celdas de ese tipo el programa creará una nueva ventana pidiendo el número de celdas para el siguiente tipo y podremos seleccionar las celdas de ese tipo de la misma forma mencionada anteriormente, así hasta llegar a las celdas de tipo obstáculo. En caso de seleccionar una celda donde previamente ya se encontraba un tipo de celda que no fuese espacio libre el programa avisará al usuario indicándole que no es posible seleccionar esa celda, obligando a este a seleccionar otra.

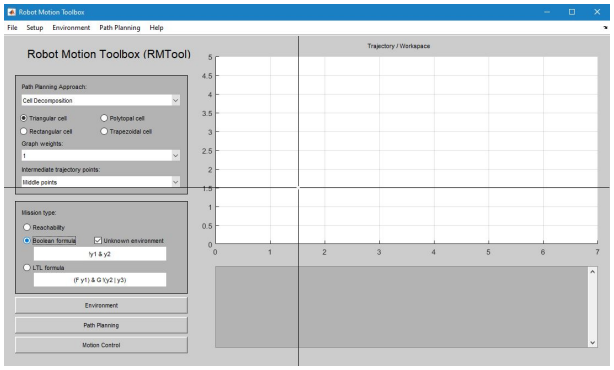

Figura B.10: Antes de hacer clic sobre la celda a cambiar a región de tipo 1.

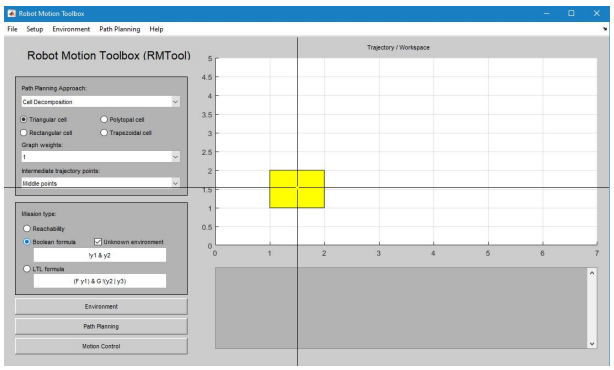

Figura B.11: Tras hacer clic la celda pasa a ser una región de tipo 1.

Una vez seleccionadas las celdas de tipo obstáculo aparecerá una ventana informándonos que debemos seleccionar la posición inicial de los robots con el clic derecho del ratón sobre el entorno, tras esto podremos seleccionar las posiciones iniciales de todos los robots en el entorno como se ve en las figuras B.12 y B.13. Cabe destacar que si el usuario seleccionase como posición inicial una celda que no fuese espacio libre el programa avisaría al usuario indicando que la posición inicial de un robot no puede encontrarse en una celda que no sea espacio libre, obligándole a seleccionar una celda que sea espacio libre como posición inicial del robot.

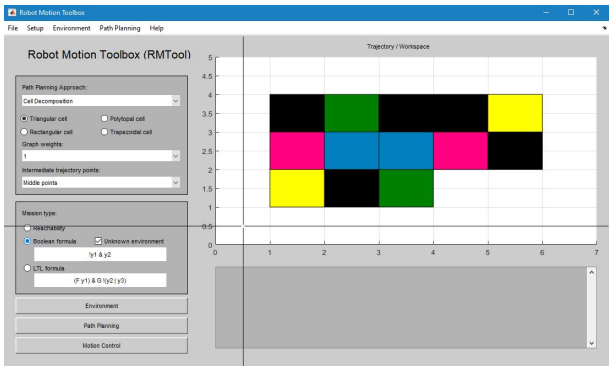

Figura B.12: Antes de hacer clic sobre la celda donde colocar el primer robot.

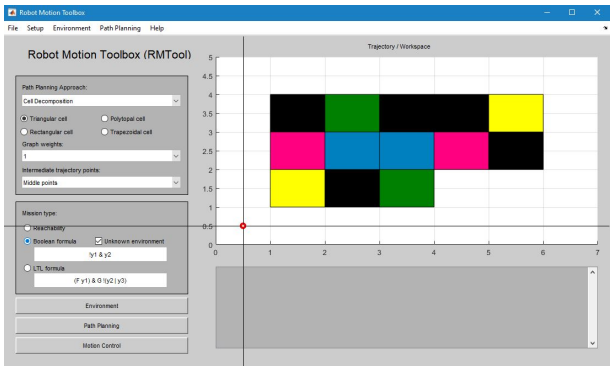

Figura B.13: Tras hacer clic la celda pasa a contener el robot 1.

Tras seleccionar las posiciones iniciales de los robots ya estará creado nuestro entorno de forma manual como se ve en la figura B.14.

Para el caso donde se quiera generar otro entorno desconocido por los robots teniendo ya los botones 'Boolean' y 'Unknown Environment' activos el usuario tendrá que dar clic sobre el botón 'Environment', el cual se encuentra en la zona inferior izquierda de la interfaz de RMTool.

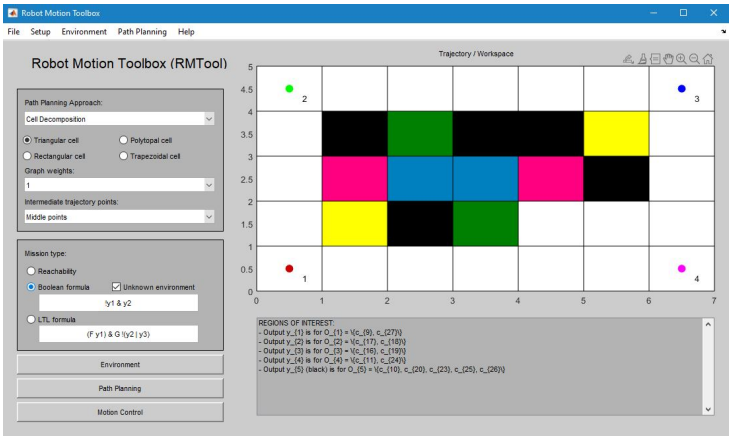

Figura B.14: Interfaz de RMTool tras generar un entorno desconocido manualmente.

B.3. Path Planning

B.3.1. Definición de la fórmula booleana

Con el entorno ya creado se podrá ejecutar el *Path planning*. El primer paso será definir la fórmula booleana en forma normal conjuntiva (FNC) que indique la misión a cumplir por los robots.

Para ello seleccionaremos el campo de texto que se encuentra debajo de los botones 'Boolean formula' y 'Unknown environment' (ver figura B.5), y escribiremos sobre este la fórmula booleana en FNC que queramos que los robots cumplan.

B.3.2. Ejecución del Path Planning

Para poder ejecutar el *Path Planning* haremos clic sobre el botón 'Path Planning' en RMTool. Tras hacer clic el programa abrirá una nueva ventana donde nos pedirá introducir, a partir de un número entre el 1 y el 3, el tipo de algoritmo de planificación a usar, siendo 1 el algoritmo independiente, 2 el algoritmo centralizado y 3 el algoritmo de consenso; también nos pedirá los valores para el radio de comunicación de los robots, las probabilidades tau y tau obs a usar para la resolución del problema de optimización MILP y el número máximo de iteraciones a realizar por el *Path Planning*, tal y como se muestra en la figura B.15.

Una vez elegidos los valores y darle a 'Ok' podremos ver cómo los robots se van desplazando en nuestro entorno hasta cumplir con la fórmula booleana introducida anteriormente. Además, también veremos los mapas de creencias o belief maps de los tres primeros robots debajo de nuestro entorno creado donde también veremos cómo se mueven los ro-

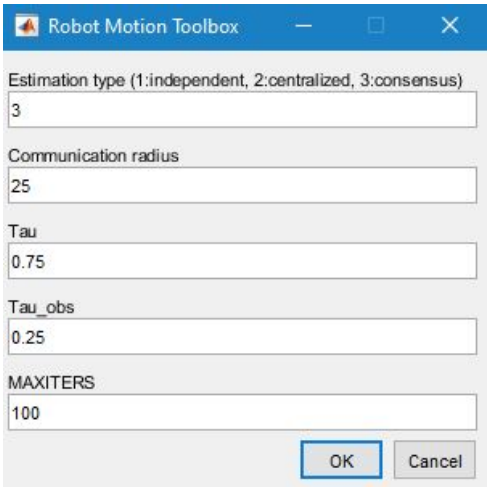

Figura B.15: Ventana donde introducir el tipo de algoritmo a utilizar en el *Path Planning*, el radio de comunicación de los robots, las probabilidades tau y tau obs y el número máximo de iteraciones del *Path Planning*.

bots además de cómo perciben estos el entorno y la posición de sus robots compañeros de forma probabilista. En la figura B.16 se puede observar RMTool tras la ejecución del *Path planning*.

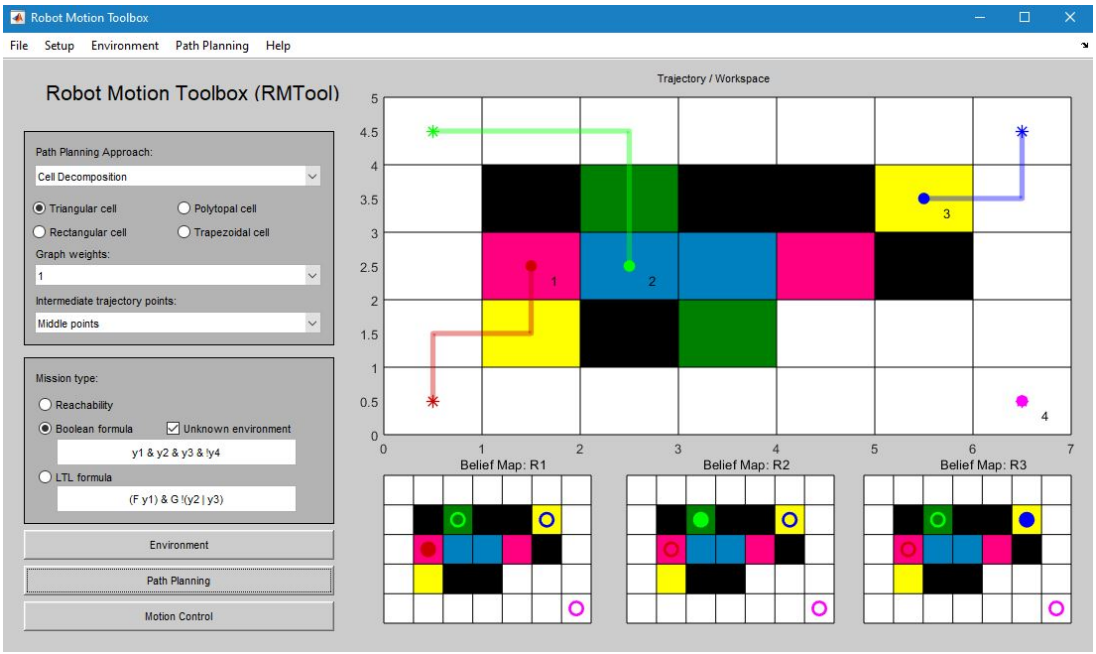

Figura B.16: Ejecución del *Path planning* con fórmula booleana y_1 & y_2 & y_3 & $!y_4$.

Se puede ver cómo para este ejemplo los mapas de creencias de los tres primeros robots coinciden con la realidad; además, tras la ejecución vemos como se han dibujado unas lineas y asteriscos del mismo color de los robots. Las líneas simbolizan la trayectoria

que los robots han realizado durante la ejecución, mientras que los asteriscos simbolizan las posiciones iniciales desde donde parte cada robot.

Las regiones y_1, y_2, y_3 e y_4 corresponden a los colores de celda amarillo, azul, rosa y verde, respectivamente.

Bibliografía

[1] E. Montijano and C. Mahulea, "Probabilistic multi-robot path planning with high-level specifications using petri net models," in *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, 2021, pp. 2188–2193.

[2] C. Mahulea, E. Montijano, and M. Kloetzer, "Distributed multirobot path planning in unknown maps using petri net models," *IFAC-PapersOnLine*, vol. 53, pp. 2063–2068, 01 2020.

[3] C. Mahulea, R. González, E. Montijano, and M. Silva, "Planificación de trayectorias en sistemas multirobot utilizando redes de petri. resultados y problemas abiertos," *Revista Iberoamericana de Automática e Informática industrial*, vol. 18, no. 1, p. 19–31, dic. 2020. [Online]. Available: https://polipapers.upv.es/index.php/RIAI/article/view/13785

[4] Wikipedia, "Forma normal conjuntiva — wikipedia, la enciclopedia libre," 2020. [Online]. Available: https://es.wikipedia.org/w/index.php?title=Forma_normal_conjuntiva&oldid=127371297

[5] ——, "Matlab — wikipedia, la enciclopedia libre," 2022. [Online]. Available: https://es.wikipedia.org/w/index.php?title=MATLAB&oldid=143229599

[6] Matlab, "Página web de la biblioteca optimization toolbox," 2022. [Online]. Available: https://www.mathworks.com/products/optimization.html

[7] Wikipedia contributors, "Cplex — Wikipedia, the free encyclopedia," 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=CPLEX&oldid=1078866502

[8] Matlab, "Página web de la biblioteca mapping toolbox," 2022. [Online]. Available: https://www.mathworks.com/products/mapping.html

[9] Wikipedia contributors, "Breadth-first search — Wikipedia, the free encyclopedia," 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Breadth-first_search&oldid=1091199031

[10] csacademy, "Página web de csacademy que permite generar grafos," 2022. [Online]. Available: https://csacademy.com/app/graph_editor/

[11] Matlab, “Página web de matlab,” 2022. [Online]. Available: https://www.mathworks.com/products/matlab.html

[12] Unizar, “Página web donde obtener una licencia para matlab,” 2022. [Online]. Available: https://sicuz.unizar.es/soporte-equipamiento-puesto-de-trabajo/gestion-de-software-corporativo/matlab

[13] Google, “Acceso al fichero de instalación de la herramienta rmtool modificada y cplex,” 2022. [Online]. Available: https://drive.google.com/drive/folders/1XIAHYznSVv4_PeSm6mjbeXaggm-OUByG?usp=sharing

[14] Unizar, “Página web de la herramienta rmtool,” 2022. [Online]. Available: http://webdiis.unizar.es/RMTool/

[15] GitHub, “Repositorio oficial de la herramienta rmtool,” 2022. [Online]. Available: https://github.com/cmahulea/RobotMotionToolbox-RMTool-under-MATLAB

[16] RMTool, “Documentación de la herramienta rmtool,” 2022. [Online]. Available: http://webdiis.unizar.es/RMTool/extra/manv1.pdf