Universidad Zaragoza
1542

# Trabajo Fin de Grado

## Cálculo optimizado de puntos intermedios en la planificación de rutas.

## Optimized waypoints computation for path planning


Autor

Isabel Carrizo Ruiz

Director/es

Eduardo Montijano Muñoz
Cristian Mahulea

Escuela de Ingeniería y Arquitectura

2021

# Resumen

Los robots móviles tienen muchas aplicaciones, una de ella siendo la de transportar objetos dentro de una fabrica, almacén o entorno industrial desde una posición inicial hasta una posición final. Existen muchos caminos para llegar a un destino fijado, sin embargo, en muchos casos el camino de mayor interés es el de menor distancia recorrida, ya que permite reducir el tiempo de ejecución, así como el coste de producción. Este Trabajo Final de Grado (TFG) se centra en el problema de planificación de trayectorias de un robot móvil con el objetivo de obtener trayectorias más cortas para que el robot alcance el destino final.

Se parte de un mapa de un entorno de trabajo dividido en regiones (o celdas), al cual se le aplica un algoritmo de planificación de trayectorias para obtener la secuencia de regiones más corta para llegar al destino. Esta partición se abstrae en un grafo, donde los nodos modelan las regiones de dicho entorno. Aplicando el algoritmo Dijkstra para la planificación de trayectorias, se calcula, el camino más corto desde un nodo inicial al resto de los nodos del grafo. Este camino en realidad devuelve una secuencia de regiones que el robot tiene que seguir para alcanzar el destino final. Sin embargo, para obtener la trayectoria exacta para el robot, se debe especificar los puntos intermedios por cuales tiene que pasar el robot, en particular, para dos regiones adyacentes se debe saber el punto exacto de cruce entre las celdas. Este TFG consiste en la implementación y evaluación de dos algoritmos para calcular estos puntos intermedios. El primer algoritmo que se ha implementado está basado en el cálculo de los puntos medios del segmento que tienen en común dos regiones, y el segundo algoritmo se basa en el cálculo del punto de intersección entre el segmento común y la recta que une los centros de las dos celdas por las que tiene que pasar.

Estos algoritmos calculan los puntos de la trayectoria que el robot ha de seguir para llegar a una posición final sin colisionar con ningún obstáculo. Una vez implementados, es necesario evaluarlos y compararlos. El algoritmo que calcula la trayectoria que recorre una menor distancia es el que debe ser usado para que el cálculo de los puntos sea optimizado.

# Abstract

Mobile robots have many applications, one of them being to transport objects within a factory, warehouse or industrial environment from an initial position to a final position. There are many routes to reach a fixed destination, however, in many cases the route of greatest interest is the one with the shortest distance travelled, since it reduces the execution time, as well as the production cost. This Final Degree Project (TFG) focuses on the problem of planning the trajectories of a mobile robot with the aim of obtaining shorter paths for the robot to reach the final destination.

The starting point is a map of a work environment divided into regions (or cells), to which a trajectory planning algorithm is applied to obtain the shortest sequence of regions to reach the destination. This partition is abstracted into a graph, where the nodes model the regions of said environment. Applying the Dijkstra algorithm for trajectory planning, the shortest path from an initial node to the rest of the nodes of the graph is calculated. This path actually returns a sequence of regions that the robot has to follow to reach the final destination. However, to obtain the exact path for the robot, the intermediate points through which the robot has to pass must be specified, in particular, for two adjacent regions the exact crossing point between the cells must be known. This TFG consists of the implementation and evaluation of two algorithms to calculate these intermediate points. The first algorithm that has been implemented is based on the calculation of the midpoints of the segment that two regions have in common, and the second algorithm is based on the calculation of the point of intersection between the common segment and the line that joins the centers of the two cells it has to pass through.

These algorithms calculate the points of the trajectory that the robot has to follow to reach a final position without colliding with any obstacle. Once implemented, they need to be evaluated and compared. The algorithm that calculates the path that travels the shortest distance is the one that must be used so that the calculation of the points is optimized.

# Índice general

# Índice de figuras

# Índice de tablas

# Capítulo 1

# Introducción

En este capítulo se presenta una introducción al trabajo realizado donde se detallan la motivación, el contexto en el que se lleva a cabo y los objetivos. También se detalla la metodología que se ha seguido y los requisitos que se han tenido que cumplir.

## 1.1. Motivación y Contexto

Los robots móviles tienen muchas aplicaciones, una de ellas siendo la de transportar cargas desde un estado inicial a un estado final moviéndose por un entorno de trabajo. El mundo real está lleno de entornos con obstáculos y, es necesario que los robots diseñados sean capaces de generar trayectorias que garanticen su seguridad, la de los productos que transportan y la de todo aquello que les rodea. Este es el propósito de este Trabajo Final de Grado, la implementación de algoritmos que calculen una trayectoria en un entorno de trabajo, de manera que el robot no colisione con ningún obstáculo y llegue a su posición final de la forma más sencilla y segura posible.

Para poder llevar a cabo la planificación de trayectorias, se hace uso de algoritmos ya implementados, como puede ser el algoritmo Dijkstra. Estos algoritmos parten de un mapa dividido en regiones, conocidas también como celdas, y calculan a partir de una región inicial el camino más corto para alcanzar un destino final. Sin embargo, con la aplicación de este algoritmo, únicamente se obtiene la secuencia de regiones por la que debe circular el robot para llegar a la posición final sin colisionar con ningún obstáculo. Para obtener la trayectoria completa del autómata es necesario conocer los puntos exactos por los que pasará el robot de una celda a otra, conocidos como puntos intermedios de la trayectoria o *waypoints*. Esto se consigue con la implementación de algoritmos de cálculo optimizado de trayectorias, que tienen como función encontrar los puntos intermedios de la ruta más corta para ir de un estado inicial a un estado final. Este es el objetivo del Trabajo de fin de Grado.

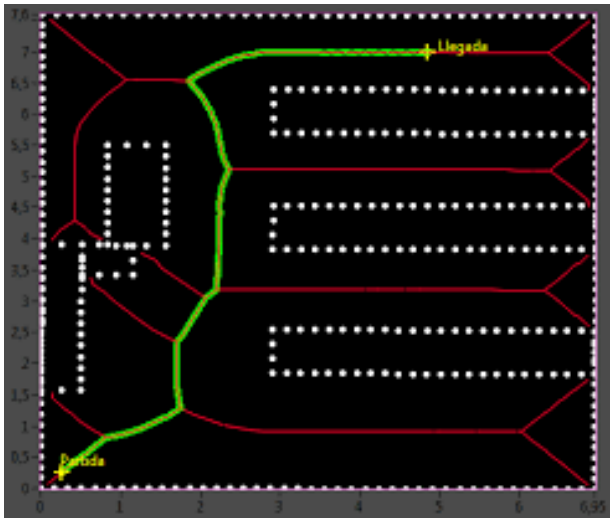


Figura 1.1: Generación de trayectorias para un robot móvil [1]

.

En la Figura 1.1, se ha generado la trayectoria desde el punto *salida* hasta el punto *llegada* para un robot móvil. Los puntos blancos representan obstáculos que el robot debe de esquivar para alcanzar su posición final. Las líneas rojas representan todas las posibles trayectorias que podría seguir el autómata. Se puede comprobar que hay diferentes caminos para llegar al destino final, sin embargo, el camino más corto es el verde.

## 1.2. Objetivos

El objetivo del trabajo propuesto es, a partir de un entorno dividido en celdas, realizar el cálculo optimizado de los puntos intermedios de la trayectoria que tiene que seguir un robot para llegar a una posición final.

Mediante la aplicación del algoritmo Dijkstra, se obtiene la secuencia de regiones que siguen el camino más corto para que el robot cumpla con la especificación. El objetivo del trabajo propuesto se consigue con la implementación y evaluación de dos algoritmos que usan diferentes métodos para calcular los puntos de cruce entre las regiones. Estos algoritmos se evalúan y comparan utilizando el entorno de simulación de ROS (Robotic Operating System).

## 1.3. Alcance del proyecto

Como se ha mencionado en la sección 1.2, el punto de partida del proyecto es un entorno dividido en celdas. A continuación se muestran las diferentes etapas a seguir en

la implementación de los algoritmos.

*Requisitos previos:* Para la realización del trabajo, se necesita el aprendizaje del manejo de ROS y del lenguaje de programación C++. Aunque durante la carrera de Ingeniería de Tecnologías Industriales se trabaja en una o dos asignaturas con dicho lenguaje de programación, el conocimiento necesario para realizar este trabajo es mucho más complejo y más amplio, requiriendo un mayor aprendizaje. A su vez, se necesita una formación previa sobre el manejo de ROS debido al desconocimiento total del programa.

*Lectura del fichero de texto:* El primer paso es leer la información del mapa dividido en regiones y almacenarla para después poder utilizarla. Esta información viene dada en un fichero de texto.

*Obtención de una secuencia de regiones:* Con el uso del algoritmo Dijkstra, o algoritmo de caminos mínimos, se obtiene una secuencia de regiones por las que tiene que cruzar el robot para seguir el camino más corto.

*Desarrollo de los algoritmos de cálculo de puntos intermedios:* A partir de la secuencia de regiones, se implementan dos algoritmos en c++ que calculan los puntos medios de cruce entre las regiones que deberá seguir el robot para llegar a una posición final. Estos algoritmos siguen diferentes métodos para realizar el cálculo.

*Evaluación y comparación:* Una vez implementados los algoritmos, se comprueba su correcto funcionamiento con la información de tres mapas diferentes. Se comparan entre ellos para evaluar cuál es el algoritmo que calcula el camino de menor distancia.

## 1.4. Estructura de la memoria

A continuación se muestra la estructura de la memoria del trabajo a realizar.

*Capítulo 1, Introducción:* Se explica el objetivo del proyecto a realizar y la motivación para llevarlo a cabo, también se detalla el proceso que ha sido necesario seguir para completarlo.

*Capítulo 2, Planificación de alto nivel:* En este capítulo se comenta el punto de partida del Trabajo Final de Grado y se explican algunos de los algoritmos de planificación de trayectorias que actualmente existen.

*Capítulo 3, Planificación de bajo nivel:* Se explica el funcionamiento completo del algoritmo Dijkstra y se detalla todo el proceso de creación del código del programa, desde la lectura del fichero de texto hasta la creación de los dos algoritmos de cálculo de puntos intermedios.

*Capítulo 4, Evaluación y comparación:* En este capítulo se comprueba el funciona-

miento de los dos algoritmos implementados mediante tres ejemplos diferentes. Los dos algoritmos se comparan entre sí para poder observar las ventajas de un algoritmo frente al otro.

*Capítulo 5, Conclusiones:* Este es el último capítulo de la memoria, se hace un pequeño resumen del proceso a seguir para la creación del programa. Se comenta también el cumplimiento, o no, del objetivo del trabajo propuesto y por último se hace una evaluación global de los algoritmos.

# Capítulo 2

# Planificación de alto nivel

El objetivo de este Trabajo Final de Grado es la implementación de dos algoritmos que calculan de forma optimizada los puntos de una trayectoria. Actualmente, estos algoritmos no existen en ninguna librería de ROS (Robotic Operating System). ROS es un software utilizado para trabajar con robots móviles que se ha usado para compilar y ejecutar los dos algoritmos implementados.

## 2.1. Partición de regiones

El punto de partida de este trabajo es un mapa de un entorno dividido en regiones. Para llevar a cabo la partición del mapa existen varios métodos aplicados a diferentes dominios; en la robótica móvil esta partición se conoce como descomposición de celdas, donde cada región en la que se divide el mapa del entorno se denomina celda.

La idea principal de la descomposición de celdas es dividir el entorno en regiones con la misma forma (triangular, rectangular, etc.). Estas regiones cubren todo el espacio donde no hay obstáculos y donde el robot puede moverse libremente. En la robótica móvil las técnicas de descomposición de celdas están usadas para solucionar problemas de navegación de los robots en determinados entornos, donde una posición final debe ser alcanzada sin colisionar con los objetos que se puede encontrar en dicho entorno [2].

Uno de los métodos más conocidos es la descomposición trapezoidal, también existen otros diferentes como la descomposición triangular o rectangular. Estas técnicas se diferencian entre sí en la forma de sus regiones. La descomposición rectangular divide en regiones todo el mapa, de esta manera, algunas celdas quedan ocupadas por obstáculos y no pueden formar parte de la trayectoria final [2]. En las figuras 2.1, 2.2 y 2.3 se pueden observar los tres métodos de descomposición nombrados.

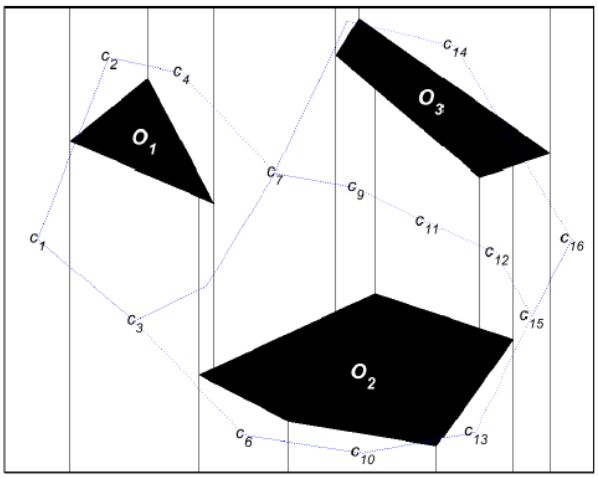


Figura 2.1: Descomposición trapezoidal [2].

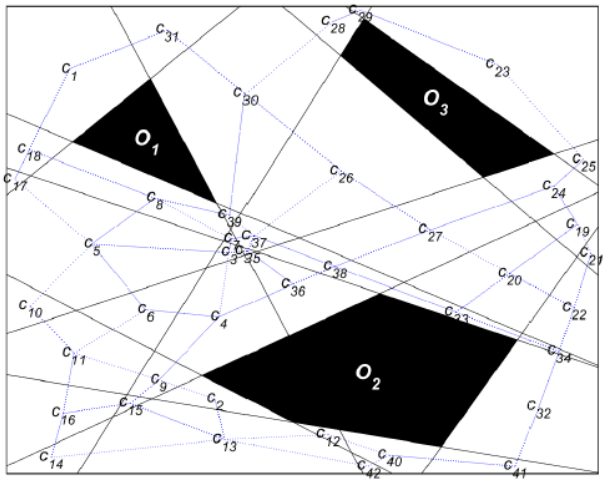


Figura 2.2: Descomposición triangular [2].

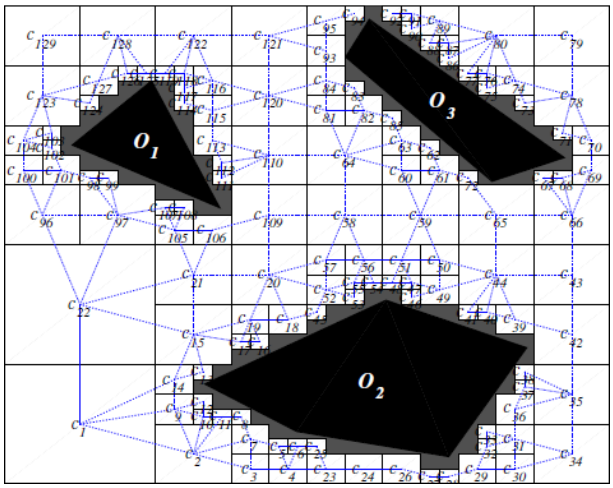


Figura 2.3: Descomposición rectangular [2].

Este Trabajo Final de Grado comienza, por lo tanto, con un mapa dividido en regiones cuya información se recoge en modelos de representación finita, como puede ser un grafo o una red de Petri. En este proyecto se trabaja con el modelo de grafo. Un grafo es una composición de objetos que se denominan nodos, donde se almacena información sobre el entorno. Dicha información es usada para procesar o conocer un fin específico. Estos nodos están unidos entre sí mediante aristas.

Dentro de los grafos más comunes se encuentra el grafo dirigido. Este tipo de grafo se caracteriza porque las aristas que unen dos nodos tienen una direccionalidad clara. A su vez, si dichas aristas incorporan datos, el grafo se denomina etiquetado, y los datos son conocidos como el peso de cada arista. Este tipo de grafo es el más utilizado en el mundo informático.

En la Figura 2.4 se puede observar un ejemplo de grafo de tres nodos dirigido, ya que todas sus aristas apuntan hacia una dirección, y etiquetado porque todas sus aristas poseen un peso. Toda la información de la partición de regiones del mapa del entorno se

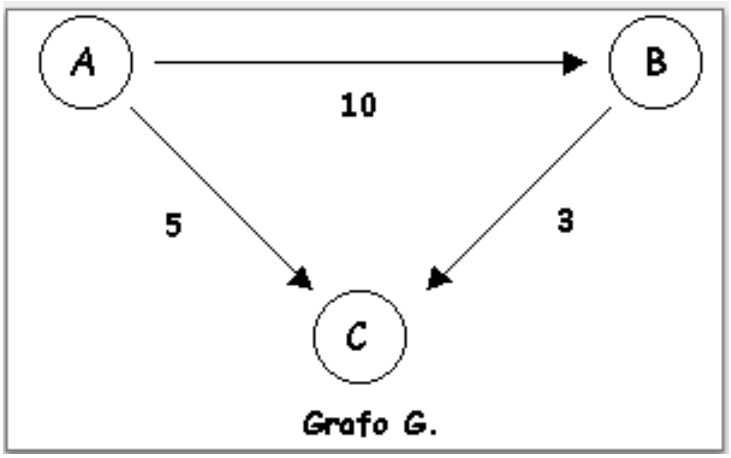


Figura 2.4: Grafo dirigido y etiquetado.

recoge en un grafo dirigido y etiquetado al que se le aplica un algoritmo de planificación de trayectorias para obtener una secuencia de regiones desde una celda inicial hasta una celda final.

## 2.2. Algoritmos de Planificación de trayectorias

La función de los algoritmos de planificación de trayectorias consiste en llevar un cuerpo, en este caso un autómata, desde una posición inicial hasta otra final siguiendo el camino más corto, dentro del entorno de trabajo, sin colisionar con ningún obstáculo [5]. En la actualidad existen varios algoritmos diferentes que calculan estas trayectorias. A continuación se van a explicar brevemente.

En primer lugar, el algoritmo A* se caracteriza por realizar una búsqueda de trayectorias completa y óptima. Consiste en encontrar un camino entre un nodo origen y un nodo destino, siendo el camino el de menor costo. Una de sus ventajas es que orienta la búsqueda de la mejor ruta teniendo en cuenta la posición del objetivo final, de esta manera, se evita visitar nodos innecesarios y, por lo tanto, supone un ahorro considerable del tiempo de ejecución. Sin embargo, entre sus desventajas está que el algoritmo funciona estimando la distancia al nodo final, por lo que no siempre la solución encontrada es la mejor, todo depende de la calidad de la estimación [6].

En segundo lugar, el algoritmo D* es un método de planificación de trayectorias que calcula el camino mínimo para ir desde un punto actual a un punto objetivo, cuando se desconoce total o parcialmente el entorno por el que se mueve el robot. Para crear

la trayectoria, primero realiza una suposición de la zona que desconoce y en base a esa suposición calcula la ruta para alcanzar el objetivo final. El autómata comienza su camino y conforme va encontrado información lo recalcula si es necesario [5].

En tercer lugar, el algoritmo RRT (*Rapidly-exploring Random Tree*) calcula un camino continuo conectando una configuración inicial y una configuración final teniendo en cuenta los obstáculos y las restricciones. Cada configuración determina la posición y orientación del robot en un espacio bidimensional o tridimensional. El funcionamiento del algoritmo consiste en, a partir de una configuración inicial, ir explorando las demás configuraciones y determinar dónde se debe colocar la siguiente para que el robot pueda acceder a ella sin colisionar con ningún obstáculo [7].

Estos son sólo tres algoritmos de planificación de trayectorias de la gran variedad que existe en la actualidad, sin embargo, en el cálculo de la trayectoria del robot de este Trabajo Final de Grado se ha empleado el algoritmo Dijkstra.

El algoritmo Dijkstra, también conocido como algoritmo de caminos mínimos, consiste en calcular la trayectoria más corta desde un nodo origen al resto de los nodos del grafo teniendo en cuenta el peso de las aristas. El peso de las aristas corresponde a un coste, que puede ser distancia, energía, etc. Fue descrito por primera vez en 1959 por Edsger Dijkstra. Este algoritmo evalúa, desde un nodo inicial, el coste invertido en desplazarse a cada uno de sus nodos adyacentes y se desplaza al de menor coste acumulado; desde este nuevo nodo se repite el proceso. Cuando ha pasado por todos los nodos del grafo, se detiene y se obtiene la secuencia de regiones. El algoritmo devuelve el camino de coste mínimo para ir de un nodo inicial a otro final. Se ha de tener cuidado porque este método tiene una única limitación y es que no funciona en grafos cuyas aristas tienen un valor negativo [6]. A continuación se detalla el funcionamiento del algoritmo.

## 2.3. Funcionamiento del algoritmo

Partiendo de un grafo dirigido ponderado de $N$ nodos no aislados, un vector $D$ de $N$ tamaño guardará al final del algoritmo las distancias desde un nodo inicial $x$, al resto de los nodos. Los pasos que se siguen son los siguientes:

1. Se inicializan todas las distancias en $D$ con un valor infinito dado que no se conocen desde el principio, excepto la del nodo inicial, $x$ que tiene que ser 0 porque la distancia de $x$ a $x$ es nula.
2. Se llama $a$ al nodo actual.
3. Se recorren todos los nodos adyacentes de $a$, excepto los que tienen un obstáculo, debido a que esos nodos no forman parte de ninguna trayectoria. Los nodos sin obstáculos se llamarán $vi$.

4. Se calcula para el nodo actual la distancia tentativa hasta todos sus nodos adyacentes mediante la siguiente fórmula:

$$D_t(v_i) = D_x + d(v_i, a) \tag{2.1}$$

La distancia tentativa del nodo $vi$ es la distancia que actualmente tiene en el vector $D$ más la distancia desde el nodo actual $a$ al nodo $vi$. Si la distancia tentativa calculada es menor que la distancia actual en el vector, se actualiza ese dato, ya que se busca el camino más corto.

$$d_t(v_i) < D_v i- > D_v i = d_t(v_i) \tag{2.2}$$

5. Se marca como completo el nodo actual en el que estamos, $a$.

6. Se toma como próximo nodo actual el de menor valor en $D$ y se vuelve al paso 3 mientras que sigan existiendo nodos sin obstáculos.

Una vez se recorran todos los nodos sin obstáculos, el algoritmo se dará por finalizado y el vector $D$ estará completamente lleno. En la Figura 2.5 se muestra el pseudocódigo del algoritmo Dijkstra.

```
DIJKSTRA (Grafo G, nodo_fuente s)
    para u ∈ V[G] hacer
        distancia[u] = INFINITO
        padre[u] = NULL
        visto[u] = false
    distancia[s] = 0
    adicionar (cola, (s, distancia[s]))
    mientras que cola no es vacía hacer
        u = extraer_mínimo(cola)
        visto[u] = true
        para todos v ∈ adyacencia[u] hacer
            si ¬ visto[v]
                si distancia[v] > distancia[u] + peso (u, v) hacer
                    distancia[v] = distancia[u] + peso (u, v)
                    padre[v] = u
                    adicionar(cola,(v, distancia[v]))
```

Figura 2.5: Pseudocódigo del algoritmo Dijkstra [3].

Para la aplicación de numerosas funciones o algoritmos se hace uso de librerías. Una librería es un conjunto de archivos que contiene funciones ya implementadas y que se usa para facilitar la programación. El algoritmo Dijkstra viene integrado en la librería Boost y se denomina *Dijkstra's Shortest Path*. Boost está compuesta por una serie de documentos enfocados cada uno a un campo específico, en este caso se ha hecho uso de *The Boost Graph Library (BGL)*, que contiene varios algoritmos y estructuras para trabajar con grafos [8].

En la aplicación del algoritmo Dijkstra de Boost hay dos vectores de gran importancia, *distance map (d)* y *predecessor map (p)*. Para cada nodo $u$, se calculan las distancias hacia todos sus nodos adyacentes, y la distancia más corta encontrada se almacena en el vector $d[u]$. Por otro lado, el vector $p$ es el encargado de almacenar para cada nodo $u$ en $V$, siendo $V$ el vector que contiene todos los nodos del grafo, el nodo predecesor de cada $u$. $p[u]$ contiene el nodo que va antes de $u$ en la secuencia de regiones final. El algoritmo devuelve ambos vectores, $d$ y $p$, que permiten conocer para cada nodo $u$ la distancia más corta hacia sus adyacentes y su nodo predecesor [8]. Existe además una cola de prioridad, $Q$, en la que se encuentran los nodos por orden de prioridad en función de sus distancias.

# Capítulo 3

# Planificación de bajo nivel

En el capítulo anterior se ha explicado el funcionamiento del algoritmo de planificación de trayectorias Dijkstra y, en este capítulo se detalla cómo, a partir de la secuencia de regiones obtenida por dicho algoritmo, se calculan los puntos de la trayectoria del un robot.

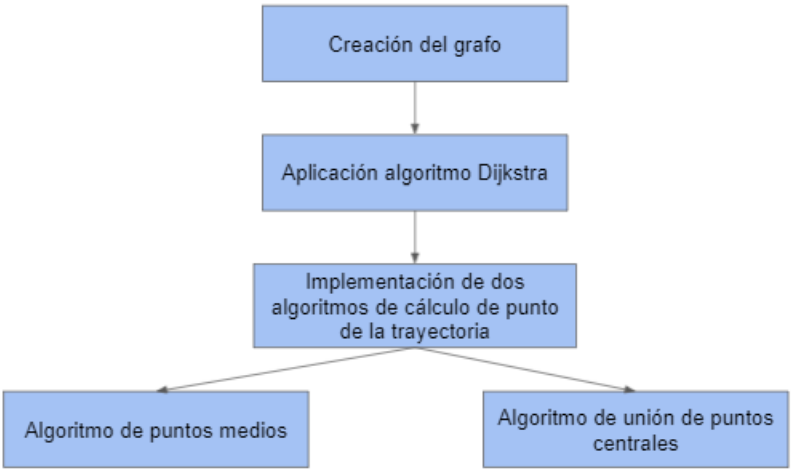


Figura 3.1: Esquema del proceso a seguir para alcanzar el objetivo del trabajo.

En la Figura 3.1 se muestra un esquema del proceso a seguir. El proceso comienza con la creación de un grafo a partir de la información leída de un fichero de texto, cada celda en la que está dividido el mapa es un nodo del grafo. Una vez creado el grafo, se aplica el algoritmo Dijkstra para obtener una secuencia de regiones que tiene que seguir el robot para ir de una posición inicial a una posición final, de manera que, el camino sea el más corto posible. A partir de la secuencia de regiones, se implementan dos algoritmos que calculan de manera optimizada los puntos exactos de cruce entre las regiones para

alcanzar la posición final. El primer algoritmo se basa en el cálculo de los puntos medios del segmento que tienen en común dos celdas, mientras que el segundo algoritmo se basa en la intersección entre el segmento común y la recta que une los dos centros.

A continuación se detalla la metodología a seguir para alcanzar el objetivo del trabajo. El código implementado para la realización de este trabajo se encuentra en una carpeta cuyo enlace está en el Anexo A.2.

## 3.1. Metodología

El desarrollo del trabajo se puede separar en tres partes fundamentales; la primera parte es la creación del grafo a partir de la información de un mapa dividido en celdas, la segunda parte es la aplicación del algoritmo de búsqueda de trayectorias (Algoritmo Dijkstra) para obtener la secuencia de regiones, y la tercera parte es la implementación de dos algoritmos que calculan los puntos de cruce entre regiones que forman parte de la secuencia obtenida.

### 3.1.1. Parte 1: Creación del grafo

La primera parte que se ha desarrollado ha sido la creación del grafo a partir de información que se recibe en un fichero de texto. El fichero de texto tiene una estructura determinada que hará posible el entendimiento de la información. La primera línea del fichero muestra un encabezado donde se indica qué parámetro se está leyendo en cada momento. La segunda línea del fichero de texto indica el número de celdas totales que tiene el entorno, es decir, el número de nodos del grafo. A partir de la tercera línea, se muestra en cada una la información de un nodo diferente de la manera que ha sido descrita, hasta llegar al final del fichero. Los datos de cada línea del fichero se muestran de la siguiente forma:

*Celda; ocupado; número vértices; lista de x; lista y; número adyacentes; ady(Puntos en común):(primer punto)...(último punto);*

A continuación se detalla qué es cada elemento.

- *Celda:* Indica el número de celda que se está leyendo, es decir, el nodo del grafo.
- *Ocupado:* Es un 0 si la celda está libre de obstáculos y un 1 si tiene un obstáculo.
- *Número vértices:* Indica el número de vértices que posee la celda.
- *Lista de $x$ y lista de $y$:* Son dos vectores en los que están almacenadas las coordenadas $x$ e $y$ de cada punto. Están almacenadas en el mismo orden, es decir el primer valor

de la lista de $x$ y de la lista de $y$ serían las coordenadas de un punto en concreto.

- *Número de adyacentes:* Es el número de celdas adyacentes que tiene cada una. Esta información es necesaria debido a que el robot únicamente podrá pasar de una celda a otra si estas dos son adyacentes.
- *ady(Puntos en común):* 'Ady' es el número de la celda adyacente, y entre paréntesis se especifica los dos puntos que unen el segmento que tienen en común las dos celdas.
- *(primer punto)...(último punto):* En el formato que se muestra arriba aparecen representados todos los puntos de unión.

Cada línea del fichero es leída y almacenada en una variable de tipo *struct*. Un *struct*, o estructura, se define como un tipo de dato compuesto que permite almacenar un conjunto de datos de diferente tipo, estos datos pueden ser números enteros, caracteres, vectores, otras estructuras, etc [9]. Cada nodo del grafo es una estructura y están almacenadas en una clase, *class*. Una clase es un nuevo tipo de dato que suele ser usado para crear objetos y que crea una consistencia lógica que establece una relación entre sus miembros; cuando se declara una variable clase se está creando un objeto [9].

```
class Grafo{
    public:

        struct nodo{
            int celda;
            int ocupado;
            int num_vertices;
            int num_x;
            int num_y;
            int num_adyacentes;
            float x[1000];
            float y[1000];
            vector<Adyacente> ady;

        };
        float CalculoCentroCelda(nodo a, float *xcentro, float *ycentro);
        float CalcularPeso(nodo o, nodo d, float *peso);
        float PuntoMedio(nodo o, nodo d, float *xmedio, float *ymedio);
        float PuntosInterseccionRectas(nodo o, nodo d, float *xinterseccion, float *yinterseccion);

};
```

Figura 3.2: Clase y estructura definidas para crear el grafo a partir de la descomposición de celdas.

En la Figura 3.2 se puede observar la clase *Grafo* definida para este programa. Dentro de la clase se define una estructura denominada *nodo* que recoge la información de cada nodo del grafo en sus diferentes variables. Además la clase *Grafo* permite almacenar funciones. Estas funciones tienen como parámetros de entrada datos pertenecientes a la clase.

Leída y almacenada toda la información obtenida en el fichero de texto, se realiza el proceso real de creación del objeto grafo. En este proceso se crean las aristas uniendo de dos en dos las celdas que son adyacentes.

Para crear las aristas se hace uso de una función perteneciente a la librería Boost enfocada en los grafos. La función es *pair <int, int>*, y se encarga de establecer parejas de celdas creando una arista para unirlas. Para que esta función se pueda aplicar entre dos celdas se tienen que cumplir dos requisitos muy importantes; el primero es que tienen que ser celdas adyacentes entre ellas, se tiene que comprobar si la celda con la que se va a emparejar pertenece al vector de adyacentes y, de este modo, el robot puede pasar de una a otra. El segundo requisito es que ninguna de las dos celdas tiene que estar ocupada, es decir, las dos tienen que estar libres de obstáculos para que el robot pueda circular por ellas. Para saber si una celda está libre se debe comprobar que en la variable *ocupado* tiene un 0. En caso de que una de las dos celdas esté ocupada, o incluso las dos, se ignoran y se siguen creando parejas con otras celdas adyacentes.

Conforme se van creando las parejas entre celdas adyacentes, se calcula el peso de las aristas. El peso de una arista se ha definido como la distancia métrica entre el centro de la celda de origen y el centro de la celda de destino, aplicando la función:

$$peso = \sqrt{(y_d - y_o)^2 + (x_d - x_o)^2} \tag{3.1}$$

siendo el subíndice d, la celda de destino, y el subíndice o, la celda de origen.

Para llevar a cabo el cálculo del peso se crea una función llamada *CalcularPeso(nodo o, nodo d, float *peso)* que toma como entrada dos *structs* de nodos adyacentes y devuelve como salida el peso calculado como se indica en la ecuación 3.2.

Es importante tener en cuenta que el peso asignado a cada arista es muy determinante a la hora de calcular trayectorias. En este Trabajo de Fin de Grado, el peso ha sido calculado como la distancia métrica entre los centros de dos celdas adyacentes, sin embargo, también se puede calcular de muchas otras maneras. Un ejemplo sería calcular el peso como la distancia entre el punto de entrada a una celda y el punto de salida de dicha celda; estos puntos pueden ser los puntos medios de los segmentos que unen dos regiones adyacentes.

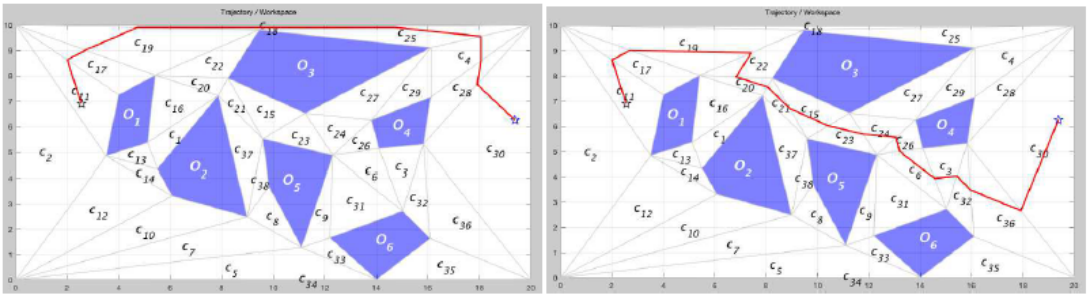

Figura 3.3: Dos trayectorias obtenidas dependiendo del peso de las aristas.

En la Figura 3.3 se puede observar como cambia la trayectoria de un robot dependiendo del peso asignado a cada arista. En la figura de la izquierda el peso está calculado como la distancia métrica entre los centros de dos celdas, y en la figura de la derecha está calculado como la distancia entre el punto de entrada a una celda y el punto de salida de la misma.

Construido el grafo, se pasa a la siguiente parte de la programación, la aplicación del algoritmo Dijkstra.

### 3.1.2. Parte 2: Aplicación Algoritmo Dijkstra

Para explicar la aplicación del algoritmo Dijkstra, se toma como ejemplo el mapa de la Figura 3.4, un mapa dividido en ocho celdas de las cuales dos están ocupadas por un obstáculo.

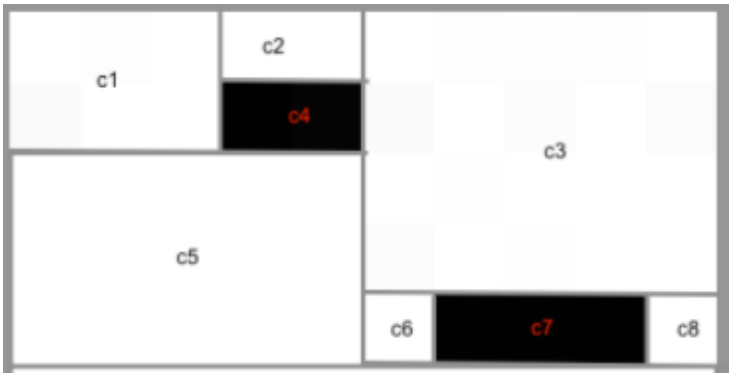


Figura 3.4: Ejemplo de un mapa dividido en ocho celdas.

Este algoritmo tiene, como parámetros de entrada, el objeto grafo, construido previamente con la función *pair <int, int>*, la cual crea aristas entre nodos adyacentes. También tiene como entrada el nodo de origen ,$s$, desde el cual se calcula la trayectoria más corta. Otros parámetros de entrada son el vector de los pesos de las aristas ,$w$, y el vector con todos los nodos del grafo ,*nodes*. Como parámetros de salida se obtiene el mapa de predecesores, un vector que guarda el nodo predecesor de menor distancia de cada nodo del grafo. También como salida se obtiene el mapa de distancias, un vector que almacena la distancia más corta para llegar a cada nodo.

Se ha de tener en cuenta que el algoritmo Dijkstra calcula el camino más corto que, desde un nodo origen, pasa por todos los nodos del grafo. Sin embargo, el objetivo de este proyecto es que el autómata se desplace a una posición final, en este caso, el último nodo del grafo. Para simplificar la programación, se ha creado un vector denominado *secuencia*, que almacena las celdas, o nodos, por las que tiene que pasar el robot para alcanzar la celda final que, en el ejemplo de la Figura 3.4 es la celda 8.

En este ejemplo, aplicando la función del algoritmo Dijkstra, siendo el punto inicial la

celda 1 y, el punto final la celda 8, la secuencia de regiones obtenida para que el robot siga el camino más corto posible es $c1$, $c2$, $c3$ y $c8$. Una vez obtenido el camino, se tienen que calcular los puntos de cruce entre estas regiones.

### 3.1.3. Parte 3: Implementación de Algoritmos de cálculo de trayectorias

El problema a resolver en esta parte se encuentra en seleccionar el punto exacto que tiene que atravesar el autómata para cruzar de una celda a otra. Estos puntos se denominan puntos intermedios (*waypoints*). A continuación se explican los dos algoritmos desarrollados.

#### Algoritmo de cálculo de puntos medios

El primer algoritmo implementado consiste en calcular el punto medio del segmento común entre las dos celdas por las que pasa el robot. Esta opción es la más popular para el cálculo de los puntos de una trayectoria.

En la implementación de este algoritmo no hay una gran complejidad debido a que los puntos son hallados de manera muy simple y usando únicamente la partición de celdas del entorno en el que se está trabajando. Para llevar a cabo la implementación se parte de la información leída del fichero de texto y almacenada en la variable *struct nodo*. Como se puede apreciar en la Figura 3.2, esta estructura cuenta con un vector de clase Adyacente, definida como se muestra en la Figura 3.5.

```
class Adyacente{
    public:
        int numero;
        int puntosInterseccion;
        float puntos_x[1000];
        float puntos_y[1000];

};
```

Figura 3.5: Clase Adyacente

Esta clase ha sido creada para almacenar los datos del segmento que tienen en común dos celdas. En la variable *numero* se guarda el número de la celda adyacente que se está leyendo, la variable *puntosInterseccion* indica cuántos puntos tienen en común las dos celdas, y por último las variables *puntos x* y *puntos y* guardan las coordenadas $x$ e $y$ de los puntos en común respectivamente.

En la Figura 3.6 se puede observar un ejemplo de los puntos por los que pasaría el robot para ir desde la celda 1 hasta la celda 8. Suponiendo que la secuencia de regiones es $c1$, $c2$, $c3$ y $c8$, los puntos marcados en rojo son los puntos medios del segmento común entre dos regiones por las que tiene que cruzar el autómata.

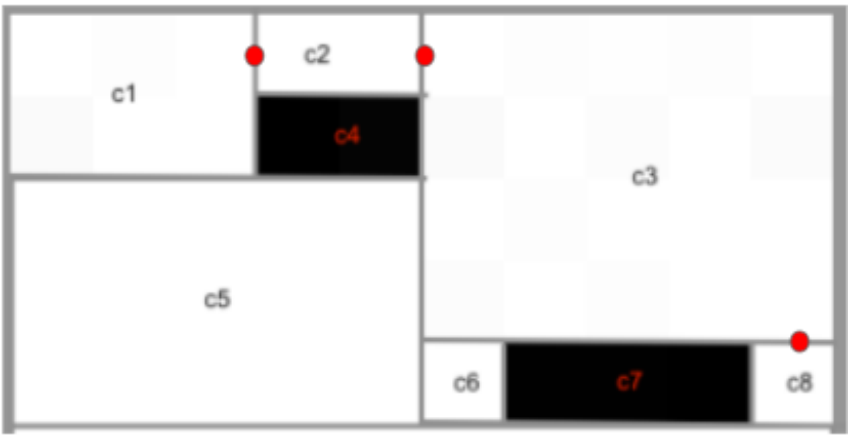


Figura 3.6: Cálculo trayectoria de un robot mediante puntos medios del segmento común.

### Algoritmo de unión de puntos centrales

Este algoritmo se basa en calcular la trayectoria a partir de la unión de dos rectas, que son el segmento de unión entre las dos celdas adyacentes y la recta que une los dos centros de ambas celdas. El punto de intersección de esas rectas es el punto por el que pasa el robot para cruzar de una región a otra. Para realizar el cálculo de ambas rectas, se utiliza la ecuación de la recta que pasa por dos puntos.

$$y = y_o + m(x - x_o) \tag{3.2}$$

siendo $m$ la pendiente de la recta

$$m = \frac{y_2 - y_1}{x_2 - x_1} \tag{3.3}$$

En la Figura 3.7 se puede observar un ejemplo de cálculo de los puntos de intersección entre las dos rectas. Suponiendo la misma secuencia de regiones que en el apartado anterior $c1$, $c2$, $c3$ y $c8$, las rectas rojas representan las rectas de unión de los centros de las dos celdas adyacentes, y los puntos negros, los puntos por los que el robot debe cruzar de una región a otra.

En la implementación de este algoritmo se ha encontrado una dificultad. La pendiente de la recta que pasa por dos puntos cuyas coordenadas $x$ son iguales, no se puede calcular. En estos casos, siendo $x1$ la coordenada de ambos puntos, la recta que une esos dos puntos es $x = x1$.

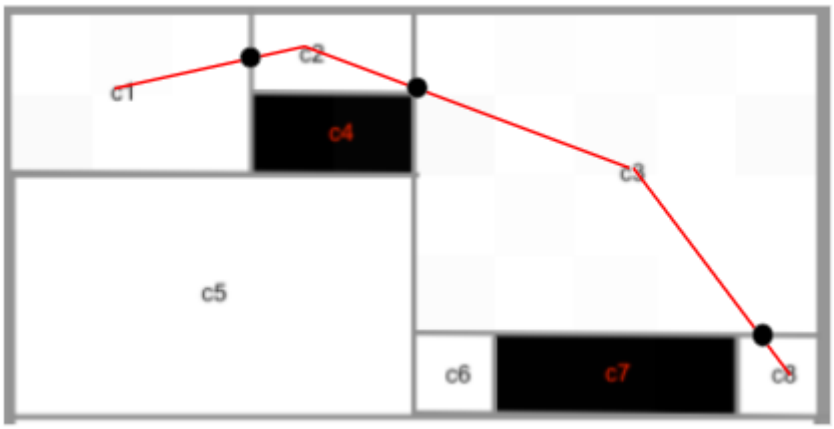


Figura 3.7: Cálculo trayectoria de un robot mediante la unión de los puntos centrales.

# Capítulo 4

# Evaluación y comparación

El objetivo del trabajo es la implementación de dos algoritmos que calculen los puntos de la trayectoria de un robot de manera que, el cálculo sea optimizado, es decir, el robot recorra la menor distancia posible para llegar a un punto final sin colisionar con ningún obstáculo. Los dos algoritmos que han sido implementados para llevar a cabo este cálculo han sido explicados en la sección 3.2.3 y son el algoritmo de cálculo de puntos medios y el algoritmo de unión de puntos centrales.

En este capítulo se hace una evaluación del funcionamiento de estos dos algoritmos, así como una comparación entre ambos. Para poder evaluarlos y compararlos, se han empleado dos métricas. La primera métrica es la distancia total recorrida, que mide en metros la longitud total que recorre el robot y, la segunda métrica es el tiempo de ejecución, que mide en segundos el tiempo que tarda el ordenador en ejecutar cada algoritmo.

## 4.1. Obtención de resultados para diferentes mapas

El proceso de obtención de los resultados comienza con un mapa que ha sido previamente dividido en celdas, al cual se le aplica el algoritmo de planificación de trayectorias para encontrar el camino de menor recorrido que ha de seguir el robot para llegar a un destino.

La información de cada celda obtenida en la descomposición viene dada en un fichero de texto, en el formato que ha sido explicado en la sección 3.2.1. A partir de esta información, se procede a la creación del grafo mediante la función *pair <int, int>* de la librería Boost, que se encarga de crear aristas entre las celdas adyacentes. Se ha de comprobar que las celdas que se van uniendo mediante aristas no están ocupadas por un obstáculo, debido a que entonces el autómata no puede cruzar por ellas. A cada arista se le asigna un peso mediante la fórmula de la ecuación 3.1.

Al grafo creado a partir de la descomposición, se le aplica el algoritmo de planificación de trayectorias, el algoritmo Dijkstra, mediante el cual se obtiene la secuencia de regiones por las que tiene que pasar el robot para alcanzar la posición final. A partir de esta secuencia de regiones, para obtener los puntos exactos de la trayectoria del robot, se aplican los dos algoritmos que han sido implementados. Estos algoritmos calculan los puntos de la trayectoria del robot, de manera que el cálculo es optimizado.

Para la aplicación del primer algoritmo se llama a la función:

*PuntoMedio(nodo o, nodo d, float *xmedio, float *ymedio).*

Esta función tiene como parámetros de entrada el nodo de origen y nodo de destino. El nodo de origen es la celda en la que está posicionado el robot, mientras que el nodo de destino es la celda a la que tiene que cruzar para seguir su trayectoria. Por otro lado, esta función devuelve como parámetros de salida dos variables de tipo *float* que representan las coordenadas $x$ e $y$ del punto de cruce entre las dos celdas. La variable de tipo *float* es una variable numérica que admite parte decimal.

Para la aplicación del segundo algoritmo se llama a la siguiente función:

*PuntosInterseccionRectas(nodo o, nodo d, float *xinterseccion, float *yinterseccion).*

Esta función tiene como parámetros de entrada, al igual que la función del algoritmo de puntos medios, la celda de origen y la celda de destino. Como parámetros de salida se obtienen dos variables de tipo *float* que representan las coordenadas del punto de cruce entre regiones. Este punto es calculado como la intersección entre el segmento común de las dos celdas y la recta que une los centros de ambas. Esta intersección se calcula siguiendo la fórmula de la ecuación 3.2.

Los resultados han sido obtenidos al ejecutar los dos algoritmos en ROS (Robotic Operating System), un software para trabajar con robots. Se han usado tres mapas con diferentes características, a partir de los cuales se han calculado las trayectorias. Para cada mapa se obtienen dos trayectorias, una con cada algoritmo implementado. Para realizar el cálculo de la distancia media y el tiempo de ejecución medio se han calculado 20 caminos diferentes. A continuación se muestran los resultados obtenidos.

Se denomina algoritmo 1 al algoritmo de puntos medios, que calcula el punto medio del segmento que tienen en común dos celdas adyacentes y, algoritmo 2 al algoritmo de unión de puntos centrales, que calcula la intersección entre el segmento en común y la recta que une los dos centros de las celdas.

### 4.1.1. Primer mapa

Para este primer ejemplo se toma como punto de partida el mapa de la Figura 4.1, un mapa de dimensiones 20x17. Este mapa se caracteriza por tener pocos obstáculos, de manera que el robot tiene más facilidad para moverse sin colisionar con ninguno. El resultado tras la descomposición de celdas es un mapa dividido en 40 celdas, de las cuales 4 están ocupadas por un obstáculo. Las celdas ocupadas no pueden formar parte de la trayectoria final del robot.

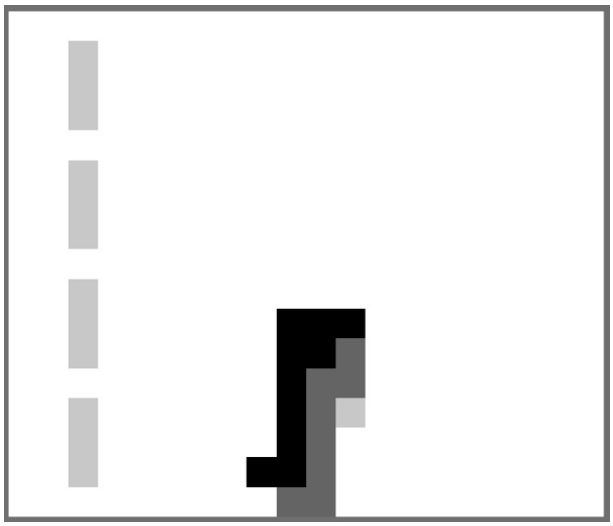

Figura 4.1: Mapa original 20x17.

Tras la aplicación del algoritmo Dijkstra, se ha obtenido la secuencia de regiones suponiendo que el punto inicial es el centro de la primera celda, y el punto final es el centro de la última celda, en este caso la celda 40. Teniendo en cuenta que el eje de coordenadas está situado en la esquina inferior izquierda, los resultados obtenidos tras aplicar ambos algoritmos son los mostrados en la tabla 4.1.

| Resultados de los dos algoritmos | | |
|---|---|---|
| Punto | Algoritmo 1 | Algoritmo 2 |
| 1º punto | (8, 7) | (6.95, 7) |
| 2º punto | (9, 8.5) | (9, 8.5) |
| 3º punto | (11, 8.5) | (11, 8.5) |
| 4º punto | (13, 9) | (13, 9) |
| 5º punto | (13, 11) | (13, 11) |
| 6º punto | (14, 12.5) | (14, 12.875) |

Tabla 4.1: Resultados tras la aplicación de los algoritmos al primer ejemplo.

Los puntos calculados por ambos algoritmos son bastante similares, sin embargo, se pueden apreciar algunas diferencias que hacen que las distancias recorridas sean diferentes. En la tabla 4.2 se muestran los resultados tras calcular la distancia total recorrida por el robot para ir desde la celda 1 hasta la última celda y el tiempo de ejecución de los dos algoritmos.

| | Algoritmo 1 | Algoritmo 2 |
|---|---|---|
| Distancia total recorrida (m) | 9.67 | 10.73 |
| Tiempo de ejecución (s) | 0.01 | 0.01 |

Tabla 4.2: Resultados del cálculo de la distancia recorrida y el tiempo de ejecución para el primer mapa.

Sin embargo, para poder hacer afirmaciones sobre qué algoritmo de los dos es mejor usar en cada caso, se han aplicado estos dos algoritmos a 20 trayectorias, con posiciones iniciales diferentes. En la tabla A.1 del Anexo 1 se muestran los resultados obtenidos al calcular las distancias recorridas por el robot, en metros, y el tiempo que tarda el ordenador en ejecutar los dos algoritmos, en segundos, para 20 caminos diferentes. En la tabla 4.3 se han calculado las medias de estos parámetros.

| | Distancia(m) Algoritmo 1 | Distancia(m) Algoritmo 2 | Tiempo(s) Algoritmo 1 | Tiempo(s) Algoritmo 2 |
|---|---|---|---|---|
| Media | 10,47 | 10,75 | 0,00385 | 0,00425 |

Tabla 4.3: Resultados del cálculo de la media de la distancia recorrida y el tiempo de ejecución para el primer mapa.

### 4.1.2. Segundo mapa

Para este segundo ejemplo se usa el mapa de la Figura 4.2. Este mapa, de dimensiones 22x22, es más grande que el mapa de la Figura 4.1. Se caracteriza por poseer una cantidad considerable de obstáculos que van a dificultar la trayectoria del robot. Tras la descomposición de celdas el mapa queda dividido en 130 regiones, de las cuales 42 están ocupadas por un obstáculo.

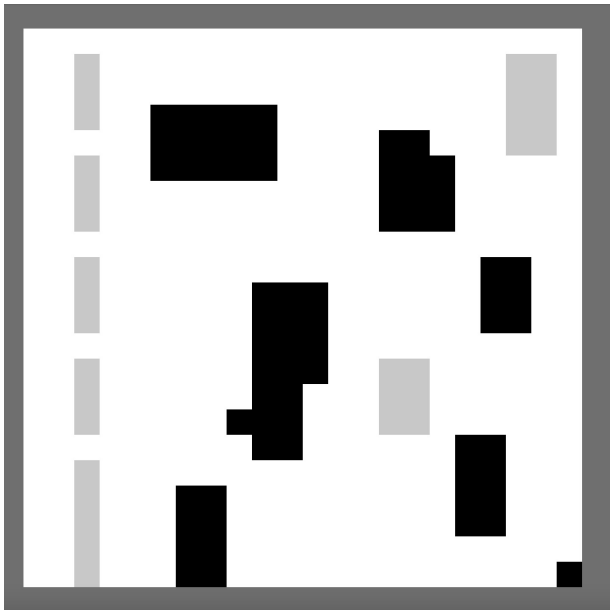

Figura 4.2: Mapa original 22x22.

Siendo el punto inicial la primera región, es decir, la celda número 1, y el punto final la celda número 130, los resultados obtenidos tras aplicar el algoritmo de puntos medios y el algoritmo de unión de puntos centrales son los mostrados en la tabla 4.4.

| Resultados de los dos algoritmos | | |
|---|---|---|
| Punto | Algoritmo 1 | Algoritmo 2 |
| 1º punto | (4, 2) | (4, 2) |
| 2º punto | (5, 2) | (5, 2) |
| 3º punto | (7, 2) | (7, 2) |
| 4º punto | (8, 2) | (8, 2) |
| 5º punto | (10, 2.25) | (10, 2) |
| 6º punto | (12, 3.5) | (12, 3.16) |
| 7º punto | (13, 4) | (13, 3.5) |
| 8º punto | (13, 5) | (13,5) |
| 9º punto | (13, 7) | (13, 6.5) |
| 10º punto | (14, 10) | (14, 10) |
| 11º punto | (15, 12) | (15, 11.75) |
| 12º punto | (16, 12) | (16, 12) |
| 13º punto | (18, 12) | (18, 12) |
| 14º punto | (19, 12) | (19, 12) |
| 15º punto | (20, 15) | (20, 15) |
| 16º punto | (20, 18) | (20, 18) |

Tabla 4.4: Resultados tras la aplicación de los algoritmos al segundo ejemplo.

Como se puede observar en los resultados, Los puntos calculados por ambos algoritmos son prácticamente iguales, únicamente diferenciándose en la coordenada $x$ del primer punto de cruce. Por ello, la distancia recorrida varía muy poco al aplicar un algoritmo u otro, como se muestra en la tabla 4.5.

| | Algoritmo 1 | Algoritmo 2 |
|---|---|---|
| Distancia total recorrida (m) | 28.18 | 28.13 |
| Tiempo de ejecución (s) | 0.031 | 0.032 |

Tabla 4.5: Resultados del cálculo de la distancia recorrida y el tiempo de ejecución para el segundo mapa.

Para poder comparar ambos algoritmos, se procede a seguir el mismo proceso que para el mapa anterior y se calculan 20 caminos con puntos de inicio distintos. Los resultados de estos cálculos se muestran en la tabla A.2 del Anexo 1. Para decidir qué algoritmo es mejor aplicar en este segundo mapa, en la tabla 4.6 se han hecho los cálculos de la media de la distancia recorrida por el robot en los caminos calculados y la media del tiempo de ejecución de cada algoritmo.

| | Distancia(m) Algoritmo 1 | Distancia(m) Algoritmo 2 | Tiempo(s) Algoritmo 1 | Tiempo(s) Algoritmo 2 |
|---|---|---|---|---|
| Media | 16,94 | 16,89 | 0,0268 | 0,0279 |

Tabla 4.6: Resultados del cálculo de la media de la distancia recorrida y el tiempo de ejecución para el segundo mapa.

### 4.1.3. Tercer mapa

Para este ejemplo se parte del mapa de la Figura 4.3. Este mapa es más pequeño que los anteriores, de dimensiones 10x7, y no posee muchos obstáculos pero al ser un mapa pequeño, abarcan gran parte del espacio.

Figura 4.3: Mapa original 10x7.

Se han calculado los puntos de la trayectoria de un robot, siendo la posición inicial la celda 1 y, la posición final, la última celda. Los resultados obtenidos tras aplicar el algoritmo de puntos medios y el algoritmo de unión de puntos centrales son los mostrados en la tabla 4.7.

| Resultados de los dos algoritmos | | |
|---|---|---|
| Punto | Algoritmo 1 | Algoritmo 2 |
| 1º punto | (1, 0) | (1, 0) |
| 2º punto | (4, 0) | (4, 0) |
| 3º punto | (5.5, 0) | (5.5, 0) |
| 4º punto | (5.5, 1) | (5.5, 1) |
| 5º punto | (6, 2) | (6, 2) |
| 6º punto | (8, 2) | (8, 2) |
| 7º punto | (17, 14) | (17, 14) |

Tabla 4.7: Resultados tras la aplicación de los algoritmos al tercer ejemplo.

Como se puede observar en los resultados, la trayectoria del autómata obtenida por ambos algoritmos es exactamente la misma. Al ser un mapa pequeño en el que los obstáculos ocupan gran parte del espacio, ambos algoritmos obtienen caminos muy parecidos, o iguales en este caso. La distancia recorrida por el robot para ir desde la primera celda hasta la última, es la misma calculada por los dos algoritmos. Sin embargo, para poder saber cuál de los dos calcula un mejor camino en la mayor parte de los casos, se han calculado 20 caminos con puntos de inicio o puntos de fin diferentes. Los resultados se muestran en la tabla A.3 del Anexo 1. En la tabla 4.8 se pueden observar los resultados al calcular la distancia media y el tiempo de ejecución medio de 20 trayectorias diferentes.

| | Distancia(m) Algoritmo 1 | Distancia(m) Algoritmo 2 | Tiempo(s) Algoritmo 1 | Tiempo(s) Algoritmo 2 |
|---|---|---|---|---|
| Media | 6,30 | 6,37 | 0,0016 | 0,00205 |

Tabla 4.8: Resultados del cálculo de la media de la distancia recorrida y el tiempo de ejecución para el segundo mapa.

## 4.2. Comparación de los algoritmos

Tras haber evaluado los dos algoritmos en los tres mapas mostrados, se puede observar que, tanto el algoritmo de cálculo de puntos medios como el algoritmo de unión de puntos centrales calculan puntos bastante similares. No hay un gran cambio de la trayectoria al usar un algoritmo u otro, sin embargo, aunque la diferencia sea pequeña, si que hay

un cambio en las distancias calculadas por cada algoritmo. En la Figura 4.4 se pueden observar las distancias medias de 20 trayectorias distintas calculadas por cada algoritmo para los tres mapas. El algoritmo 1 es el algoritmo de puntos medios, y el algoritmo 2 el de unión de puntos centrales.

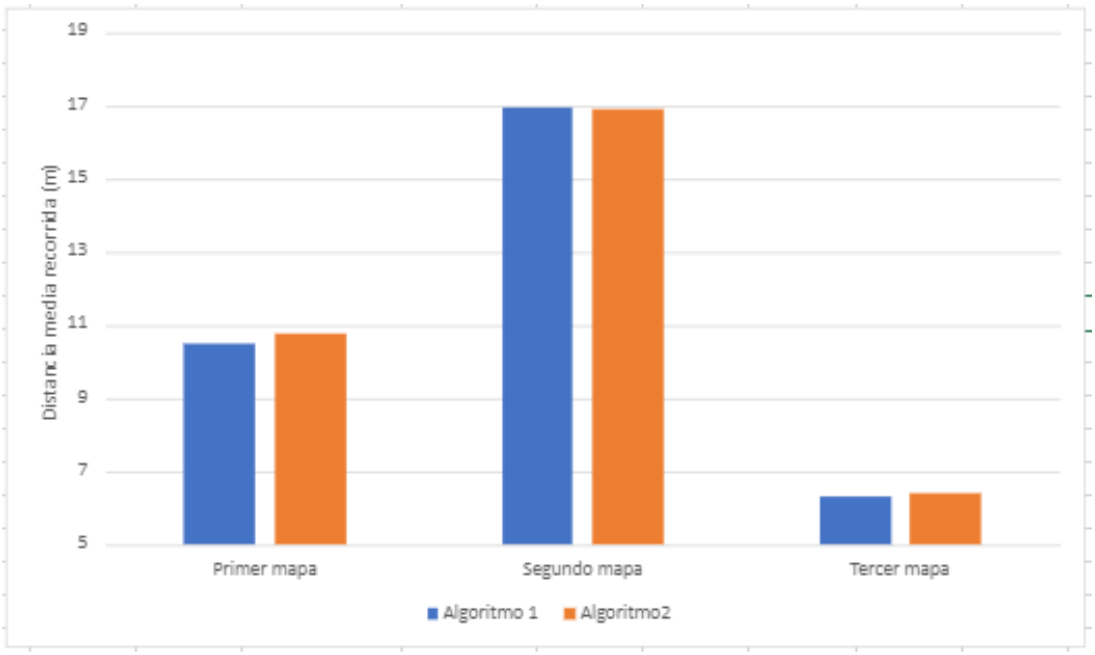


Figura 4.4: Comparación de distancias medias calculadas por los dos algoritmos.

Se puede observar que, para el primer mapa, el algoritmo que calcula una menor distancia es el algoritmo 1, sin embargo, para el segundo mapa, aunque la diferencia es muy pequeña, es el algoritmo 2 el que calcula la menor distancia. Con estos resultados se puede sacar la conclusión de que para mapas grandes con pocos obstáculos, como el de la Figura 4.1, el algoritmo que calcula la menor distancia es el de cálculo de puntos medios. Sin embargo, para mapas grandes que tienen muchos obstáculos, como el de la Figura 4.2, el algoritmo que calcula la menor distancia es el de unión de puntos centrales. Estas deferencias se pueden basar en el resultado de la descomposición de celdas, ya que para mapas con pocos obstáculos se generan un menor número de regiones que tienen mayores dimensiones, pero para mapas con un gran número de obstáculos se generan más celdas y con dimensiones más pequeñas. Al calcular las trayectorias para un número menor de celdas de mayores dimensiones el algoritmo de puntos medios calcula una menor distancia que cuando se trata de celdas con menores dimensiones.

En el caso del tercer mapa, las distancias calculadas por un algoritmo u otro varían muy poco. No obstante, se puede apreciar una ligera diferencia que hace que el algoritmo que calcule la menor distancia sea el algoritmo de puntos medios.

En cuanto al tiempo de ejecución de ambos algoritmos, se puede observar en la Figura 4.5 una comparación de los tiempos medios que tarda el ordenador en ejecutar cada

algoritmo para los tres mapas evaluados.

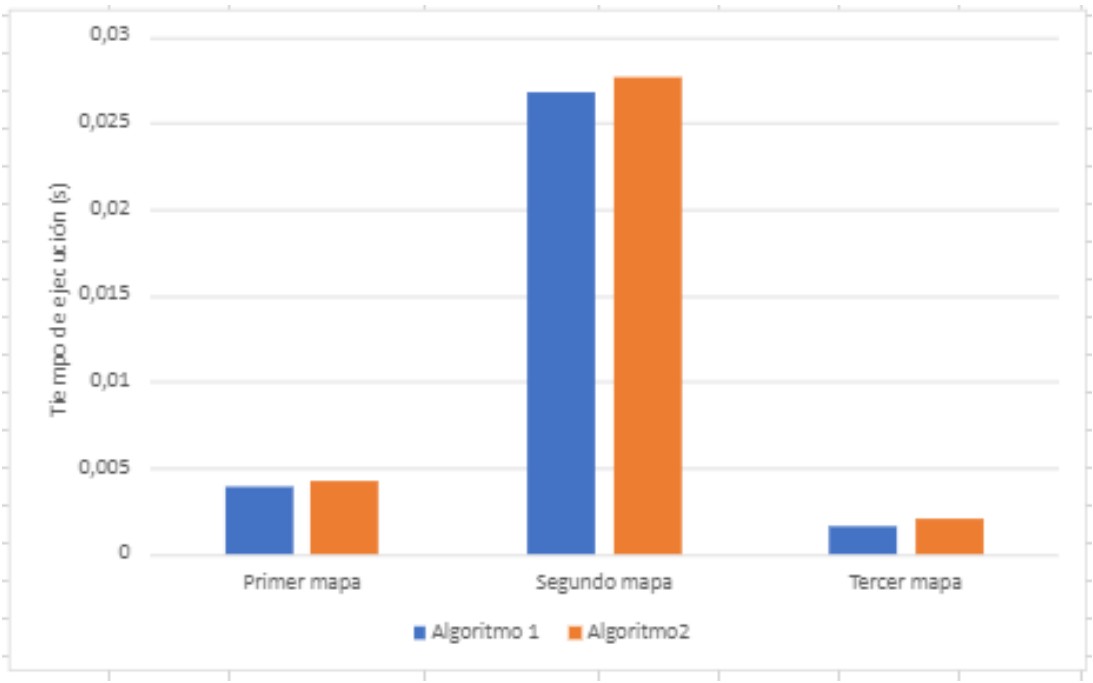


Figura 4.5: Comparación de la media de los tiempos de ejecución de los dos algoritmos.

Se puede observar que en los tres casos, el tiempo que tarda el ordenador en ejecutar el algoritmo 1 es inferior al tiempo que tarda en ejecutar el algoritmo 2. Esto puede ser porque el algoritmo 1, que es el que calcula los puntos medios del segmento común de dos celdas adyacentes, es más sencillo de implementar que el algoritmo 2. El algoritmo 1 únicamente calcula el punto medio de un segmento, mientras que el algoritmo 2 calcula dos rectas y, la intersección de ambas.

# Capítulo 5

# Conclusiones

Planificar una trayectoria de forma óptima consiste en calcular el camino más corto desde un punto inicial hasta un destino. Para realizar este cálculo existen diversos algoritmos ya implementados, entre ellos el algoritmo Dijkstra. Este algoritmo proporciona, a partir de un mapa dividido en celdas, la secuencia de regiones más corta para alcanzar una posición final. Sin embargo, para calcular los puntos exactos que forman la trayectoria de un robot, se necesitan implementar otros algoritmos.

El objetivo de este proyecto es la implementación de dos algoritmos que calculan de forma optimizada los puntos intermedios de una trayectoria que ha de seguir un autómata sin colisionar con ningún obstáculo. Para conseguir el objetivo, se han creado dos algoritmos que calculan de manera diferente los puntos de la trayectoria de un robot, el primer algoritmo calcula los puntos medios del segmento que tienen en común dos celdas adyacentes, mientras que el segundo calcula la intersección entre la recta que une los centros de las dos celdas y el segmento común de ambas. La elección del algoritmo que se debe usar en cada caso se basa en la distancia total recorrida por el robot. Si la distancia recorrida es menor, el tiempo que tarda el robot en llegar al destino final se reduce y, por lo tanto, el coste del proyecto es menor. Tras haber comprobado en el capítulo 4 su correcto funcionamiento, se puede afirmar que el objetivo ha sido alcanzado.

Al evaluar y comparar los dos algoritmos para diferentes mapas se obtienen las siguientes conclusiones:

- Para mapas grandes y con pocos obstáculos, el algoritmo que calcula una menor distancia es el de cálculo de puntos medios. Para mapas grandes y con muchos obstáculos, ocurre lo contrario, el algoritmo que calcula una menor distancia es el de unión de puntos centrales. Esta diferencia puede ser debida al resultado de la descomposición de celdas, ya que, cuando el mapa posee muchos obstáculos, se generan un gran número de celdas de pequeñas dimensiones, sin embargo, cuando el mapa presenta pocos obstáculos, se generan un menor número de celdas y son de

dimensiones más grandes. Para mapas pequeños, se ha comprobado que el algoritmo que calcula una menor distancia es el algoritmo de puntos medios. Sin embargo, en todos los casos, las trayectorias calculadas por ambos algoritmos son muy similares.

- El tiempo que tarda el ordenador en ejecutar cada algoritmo es bastante parecido, sin embargo, para todos los casos evaluados el tiempo de ejecución es inferior al aplicar el algoritmo de cálculo de puntos medios. Esto se debe a que su implementación es más sencilla ya que sólo calcula el punto medio del segmento común entre dos celdas, mientras que el algoritmo de unión de puntos centrales calcula dos rectas y la intersección de ambas.

Se ha de tener en cuenta que los algoritmos se han aplicado únicamente sobre tres mapas. Para sacar conclusiones más precisas se necesitaría probarlos sobre una cantidad considerable de mapas con diferentes características. También hay que resaltar que las distancias medias han sido calculadas en base a 20 trayectorias distintas en cada mapa, no obstante, si se quisiese hacer un análisis completo, se deberían calcular las distancias para todas las trayectorias posibles.

En este Trabajo Final de Grado se han creado únicamente dos algoritmos de cálculo de puntos intermedios, sin embargo, usando diferentes métodos para realizar el cálculo, se pueden implementar otros algoritmos distintos que puedan reducir más la distancia total recorrida. Han sido creados para su posterior integración en ROS (Robotic Operating System), un software para trabajar con robots que actualmente no cuenta con ningún algoritmo capaz de calcular estos puntos.

# Apéndice A

# Anexos

## A.1. Resultados obtenidos al aplicar los algoritmos en tres mapas diferentes

| Celda inicial | Distancia (m) Algoritmo 1 | Distancia(m) Algoritmo 2 | Tiempo(s) Algoritmo 1 | Tiempo(s) Algoritmo 2 |
|---|---|---|---|---|
| 1 | 9,66 | 10,72 | 0,003 | 0,003 |
| 3 | 13,9 | 13,8 | 0,005 | 0,005 |
| 5 | 7,86 | 8,18 | 0,003 | 0,004 |
| 2 | 4 | 4,13 | 0,003 | 0,003 |
| 4 | 11,9 | 11,76 | 0,009 | 0,009 |
| 6 | 11,36 | 11,74 | 0,003 | 0,003 |
| 7 | 10,36 | 10,68 | 0,004 | 0,005 |
| 9 | 12,4 | 12,26 | 0,003 | 0,004 |
| 11 | 16,46 | 16,47 | 0,003 | 0,004 |
| 13 | 13,41 | 13,26 | 0,003 | 0,004 |
| 15 | 14,41 | 14,26 | 0,005 | 0,005 |
| 17 | 16,41 | 16,46 | 0,003 | 0,004 |
| 18 | 15,41 | 15,26 | 0,007 | 0,007 |
| 20 | 16,41 | 16,76 | 0,003 | 0,004 |
| 23 | 17,82 | 17,88 | 0,004 | 0,004 |
| 25 | 5,86 | 6,18 | 0,003 | 0,003 |
| 26 | 3,8 | 5,62 | 0,003 | 0,003 |
| 28 | 1,8 | 3,52 | 0,003 | 0,004 |
| 33 | 3,04 | 3 | 0,004 | 0,004 |
| 36 | 3,16 | 3,09 | 0,003 | 0,003 |

Tabla A.1: Resultados del cálculo de la distancia recorrida y el tiempo de ejecución para 20 caminos diferentes del primer mapa.

| Celda inicial | Distancia(m) Algoritmo 1 | Distancia(m) Algoritmo 2 | Tiempo(s) Algoritmo 1 | Tiempo(s) Algoritmo 2 |
|---|---|---|---|---|
| 1 | 28,17 | 28,11 | 0,039 | 0,04 |
| 5 | 25,17 | 25,11 | 0,022 | 0,022 |
| 11 | 24,17 | 24,12 | 0,025 | 0,025 |
| 17 | 21,74 | 22,07 | 0,023 | 0,023 |
| 21 | 22,65 | 23,07 | 0,026 | 0,027 |
| 25 | 21,56 | 21,48 | 0,023 | 0,023 |
| 30 | 21,56 | 21,48 | 0,028 | 0,028 |
| 35 | 22,23 | 22,12 | 0,038 | 0,039 |
| 42 | 19,18 | 18,6 | 0,027 | 0,027 |
| 54 | 14,56 | 14,48 | 0,026 | 0,027 |
| 63 | 11,16 | 11,66 | 0,025 | 0,025 |
| 68 | 11,54 | 11,54 | 0,025 | 0,025 |
| 74 | 18,56 | 18,57 | 0,022 | 0,027 |
| 76 | 17,94 | 18 | 0,028 | 0,028 |
| 84 | 15,34 | 15,16 | 0,027 | 0,027 |
| 95 | 15,95 | 15,4 | 0,028 | 0,028 |
| 101 | 13,56 | 12,98 | 0,022 | 0,023 |
| 110 | 7,26 | 7,24 | 0,025 | 0,025 |
| 117 | 3 | 3,2 | 0,034 | 0,034 |
| 126 | 3,5 | 3,502 | 0,023 | 0,031 |

Tabla A.2: Resultados del cálculo de la distancia recorrida y el tiempo de ejecución para 20 caminos diferentes del segundo mapa.

| Distancia(m) Algoritmo 1 | Distancia(m) Algoritmo 2 | Tiempo(s) Algoritmo 1 | Tiempo(s) Algoritmo 2 |
|---|---|---|---|
| 8,62 | 8,62 | 0,001 | 0,002 |
| 5,62 | 5,62 | 0,002 | 0,002 |
| 9,12 | 9,24 | 0,002 | 0,002 |
| 4,11 | 3,74 | 0,002 | 0,002 |
| 5,5 | 5,5 | 0,002 | 0,002 |
| 3,11 | 2,8 | 0,002 | 0,002 |
| 4,52 | 4,23 | 0,002 | 0,003 |
| 11,11 | 11,5 | 0,002 | 0,002 |
| 12,23 | 12,45 | 0,001 | 0,002 |
| 13,23 | 13,6 | 0,001 | 0,002 |
| 2 | 2 | 0,002 | 0,002 |
| 4,11 | 4,1 | 0,001 | 0,002 |
| 4,61 | 4,97 | 0,002 | 0,002 |
| 3,12 | 3,1 | 0,001 | 0,002 |
| 7,61 | 7,97 | 0,001 | 0,002 |
| 9,11 | 9,52 | 0,002 | 0,002 |
| 2,12 | 2,13 | 0,002 | 0,002 |
| 1 | 1,054 | 0,002 | 0,002 |
| 13,23 | 13,73 | 0,001 | 0,002 |
| 2 | 1,625 | 0,001 | 0,002 |

Tabla A.3: Resultados del cálculo de la distancia recorrida y el tiempo de ejecución para 20 caminos diferentes del tercer mapa.

## A.2. Código implementado para la realización del trabajo

El código implementado en este trabajo se encuentra en la siguiente carpeta. En ella se encuentran los ficheros de texto de los tres mapas evaluados.

https://drive.google.com/drive/u/0/folders/1nuWnPRX-Ve2wj3CNSzKTeGYJUAiEZezj

## A.3. ROS: Robotic Operating System

En la actualidad existen muchos *frameworks* o lenguajes de programación específicos para robots. Algunos son específicos de un robot concreto y otros, sin embargo, son ge-

nerales. Este proyecto se ha llevado a cabo con uno de los *frameworks* más importantes, ROS (Robotic Operation System) [10].

*"El Sistema Operativo de Robot (ROS) es un marco flexible para escribir software de robot. Es una colección de herramientas, bibliotecas y convenciones que tienen como objetivo simplificar la tarea de crear un comportamiento robótico complejo y robusto en una amplia variedad de plataformas robóticas."* [11].

Esta definición, que se encuentra en la página oficial de ROS, quiere decir que este sistema operativo está dotado de librerías y herramientas que hacen que el trabajo con el robot sean más sencillo. Por ejemplo, tiene librerías que acceden a los sensores del robot, facilitando mucho la programación. Existen librerías generales para cualquier tipo de robots y otras que solo sirven para unos robots específicos. Además, en ROS se pueden incorporar otras librerías que previamente no están instaladas, de manera que se pueda trabajar con mas herramientas [10]. En este proyecto se hace uso de la librería Boost, que previamente no está instalada en el sistema operativo.

Una de las principales ventajas que tiene ROS es que, además de ser muy útil para trabajar con robots reales, es capaz de ejecutar los programas a través de simuladores [10]. Este trabajo se ha llevado a acabo sin el uso de robots reales, únicamente simulando los programas en el *frameworks*.

ROS fue desarrollado en 2007 por el Laboratorio de Inteligencia Artificial de Stanford para dar soporte a su proyecto de robot con inteligencia artificial. ROS no es un sistema operativo, sin embargo, tiene todos los servicios estándar que un sistema operativo puede tener como por ejemplo, el control de dispositivos de bajo nivel, el paso de mensajes entre procesos y el mantenimiento de paquetes. Está basado en grafos cuyos nodos son los responsables de recibir y mandar mensajes de diferentes sensores y actuadores. La librería está diseñada principalmente para un sistema UNIX(Ubuntu- Linux-) [12]. Para la elaboración del trabajo, como no se disponía de Linux, se ha trabajado desde una máquina virtual en la cual está instalado Ubuntu.

## A.4. Niveles de ROS

ROS está dividido en tres niveles diferenciados: el nivel de sistema de archivos, el nivel de computación gráfica y el nivel de la comunidad.

### A.4.1. Grafo computacional de ROS

El grafo de computación es el nivel más importante dentro de la arquitectura de ROS.

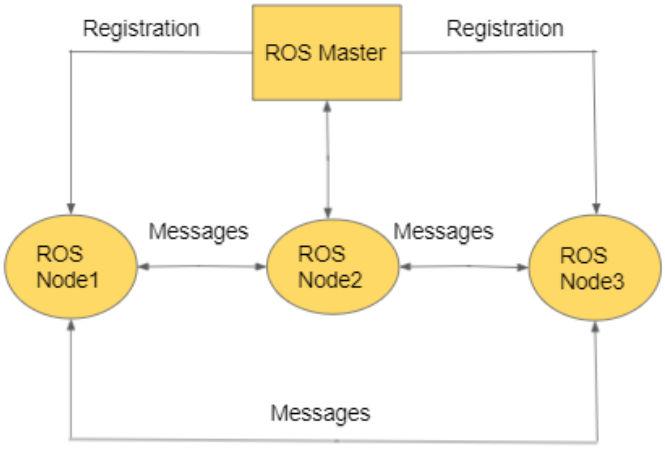


Figura A.1: Grafo computacional de ROS.

En la Figura 2.1. se puede ver como es el modelo del grafo computacional con los diferentes elementos que lo componen.

*ROS Nodes:* Los nodos son procesos en los que se realizan los cálculos, en cada uno de ellos se realiza una sola tarea. Son escritos con uso de bibliotecas como roscpp o rospy.

*ROS Master:* El master rastrea las direcciones IP de cada nodo, sin él los nodos no se podrían encontrar e intercambiar información entre ambos.

*ROS Topics:* Estos permiten que el nodo publicador y el nodo subscriptor intercambien información a través de mensajes. Cada tópico es capaz de comunicar un tipo de dato específico, por lo que, es importante que el mensaje intercambiado sea también de este tipo.

Para entender bien el funcionamiento de ROS, en la Figura A.2 2.2 se detalla el proceso de intercambio de información entre dos nodos.

En esta figura se puede observar que para que dos nodos intercambien información entre sí, diferentes elementos son necesarios. En primer lugar, los Topics sirven para que los mensajes puedan viajar de un nodo a otro y en segundo lugar los servicios, que son aquellos encargados del transporte bidireccional, es decir, están definidos por dos estructuras de mensajes, uno para la solicitud y otro para la respuesta. De esta forma los nodos pueden comunicarse entre sí, un nodo proveedor ofrece un servicio y un nodo cliente lo utiliza enviando una solicitud y esperando una respuesta [13].

El Máster es el encargado de que el intercambio de datos entre los nodos pueda llevarse a cabo. En él encontramos el servidor de parámetros, almacena los datos por claves. Y por último tendríamos las Bolsas (Bags) que sirven para guardar y reproducir mensajes de ROS.

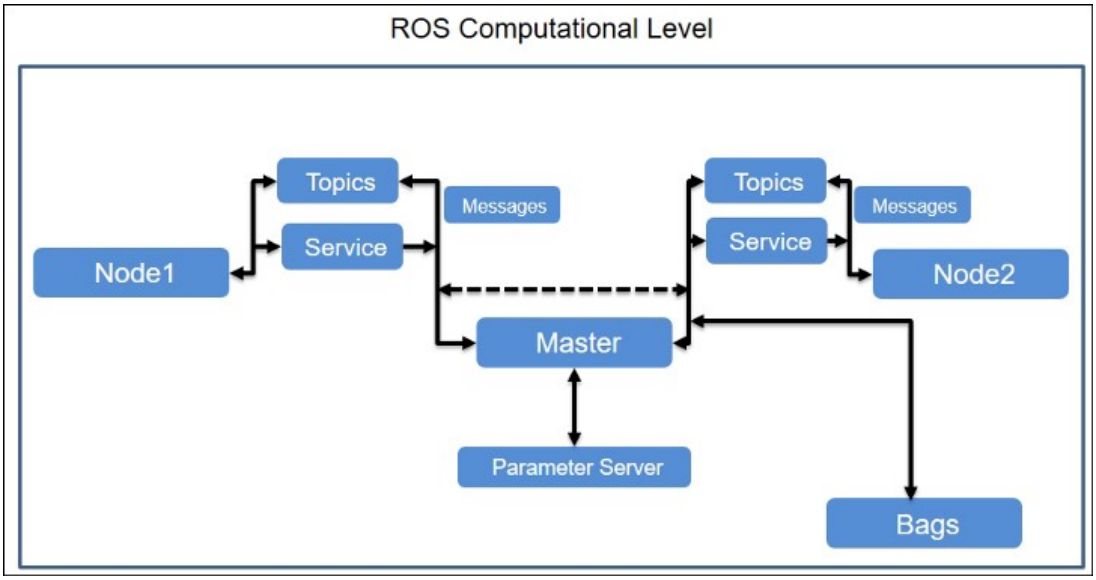


Figura A.2: Gráfico computacional ROS de 2 nodos que se comunican entre sí.

### A.4.2. Sistema de archivos

El sistema de archivos es lo que engloba los archivos necesarios para el correcto funcionamiento del software. Como elementos más importantes se pueden identificar los paquetes, que son la unidad básica de organización. Un paquete contiene toda la información necesaria para que el programa se pueda ejecutar, contiene, principalmente, la información de los nodos, bibliotecas, etc [13].

En este proyecto, se han creado dos paquetes diferentes, dentro del espacio de trabajo. Un paquete llamado *grafo*, que es el que contiene la información del nodo y es el paquete que se tiene que ejecutar para el desarrollo del programa. Y, otro paquete, llamado *grafolibrary*, que es aquel que contiene la biblioteca con las funciones que han sido creadas para que funcione correctamente el programa.

El sistema de archivos está compuesto también por un manifiesto, que es un fichero con información sobre un determinado paquete, como por ejemplo, su nombre, la versión, la descripción, etc.

ROS provee una serie de librerías y herramientas para que los encargados del software puedan desarrollar aplicaciones para robots. Algunas de las librerías más populares son roscpp y rospy. Roscpp es la más utilizada de ROS y es una implementación en c++ que sirve para interactuar rápidamente con servicios y parámetros,y rospy que tiene la misma función que roscpp pero usa lenguaje Python [13]. Sin embargo, en este trabajo se ha agregado a ROS la librería Boost que cuenta con una serie de funciones dedicadas a grafos, que es con lo que se está trabajando.

## A.5. la robótica móvil

La robótica es una técnica que aplica la tecnología al diseño y la construcción de aparatos, llamados robots, que realizan trabajos u oeraciones en sustitución de personas normalmente en ámbitos industriales [4]. Actualmente la robótica móvil se considera un área de tecnología avanzada; es un campo de investigación que se está desarrollando continuamente pero que aún queda mucho por investigar.Los productos de la robótica móvil se basan en aplicaciones de programación, inteligencia artificial y sirven de base para grandes avances de la industria [14].

Además el miedo del ser humano a ser reemplazado en su totalidad por los robots ha ido desapareciendo con el tiempo. Esto es debido al gran impacto tecnológico, social y, sobre todo, económico que han tenido los robots en la sociedad.

## A.6. Los robots

Un robot es una máquina que es capaz de realizar tareas complejas, tomar decisiones y, actuar consecuentemente. Se llama robot móvil, cuando es capaz de desplazarse en cualquier ambiente dado. Este tipo de robots están provistos de patas, ruedas u orugas, como se puede apreciar en la Figura 2.1 que les facilitan el poder desplazarse de acuerdo a su programación. Estos son empleados, sobre todo, en el transporte de mercancías en cadenas de producción y almacenes. Estos robots son también muy útiles en la investigación de lugares de difícil acceso, como por ejemplo en la exploración espacial y las investigaciones submarinas [4].

### A.6.1. Clasificación de los robots

En general, se ha considerado que existen tres grandes tipos de robots: Los robots industriales, los médicos y los móviles.

*Los robots industriales:* Este tipo de robots son los más usados en tareas de alcance económico. Están formados por una estructura mecánica articulada que se mueve de acuerdo a su programación. Estos son capaces de mover cargas pesadas a elevadas velocidades y con una gran exactitud [4].

*Los robots médicos:* Estos robots tienen diferentes aplicaciones, algunos son usados para rehabilitación, como prótesis inteligente. Estos se diferencian del resto porque tienen la forma de la extremidad correspondiente. En este tipo de robot, las señales provienen de señales nerviosas o musculares. Dentro de este conjunto entran también aquellos robots desarrollados para ayudar en las intervenciones quirúrgicas de gran precisión o de alta

Figura A.3: Robot industrial Kuka [4].

complejidad. Un ejemplo de robot quirúrgico es el de la Figura 4.2, llamado robot Da Vinci y es el instrumento quirúrgico más sofisticado que existe. Es capaz de obedecer al cirujano, a la vez que opera con una mayor precisión y destreza que un ser humano y además, proporciona una clara visión de la anatomía del paciente [4].

Figura A.4: Robot quirúrgico Da Vinci [4].

*Los robots móviles:* Este tipo de robots son capaces de desplazarse en cualquier medio, están dotados de una plataforma mecánica que les permite, de manera automática, moverse en un determinado espacio de trabajo. Son usados normalmente para transportar cargas de un punto inicial a un estado final. Este proyecto consiste, como ya ha sido mencionado, en la implementación de algoritmos para calcular la trayectoria de un robot. El robot usado es de tipo móvil, ya que, se desplaza de forma automática. Las tareas de este tipo de robot suelen ser muy diferentes pero, normalmente, están usados en aquellas que son peligrosas para el ser humano, por ejemplo, en la manipulación de materiales explosivos, el mantenimiento de reactores nucleares, etc [4].

Estos robots son autónomos, no sólo porque son capaces de desplazarse sin intervención del ser humano, sino también porque están dotados de capacidades para percibir, planificar y actuar de forma autónoma, ya que muchas veces el robot se desenvuelve en entornos desconocidos. En este proyecto, en el entorno se encuentran una serie de obstáculos que el robot tiene que ser capaz de identificar y evitar.

Figura A.5: Robot móvil diseñado por Weston Robot y Agile X [4].

En la Figura 4.3, se muestra el robot móvil diseñado por Weston Robot y Agile X. Este robot ha sido creado en Singapur para la desinfección de lugares para combatir el COVID-19.

## A.7. El funcionamiento del robot

Dos de las características más importantes de los robots son la versatilidad y la autoadaptabilidad. La versatilidad es la capacidad que tiene el robot de realizar tareas diferentes o, de realizar una tarea de varias maneras distintas. La autoadaptabilidad es la propiedad que permite que el robot se mueva hasta alcanzar el objetivo final a pesar de los obstáculos que se puede encontrar en el camino. Para ello, está dotados de sensores que les hacen capaces de conocer el entorno en el que se mueven [4].

Un robot está compuesto por cuatro sistemas importantes relacionados entre sí; el sistema mecánico, sensorial y de control. El mecánico es aquel que permite que el robot se mueva, el sensorial es aquel que le permite el conocimiento del entorno en el que está trabajando, es decir, la identificación de obstáculos para poder evitarlos. El sistema de control es el cerebro del robot, en él se procesa la información obtenida del entorno y es el que manda la orden para actuar; este sistema está dotado de algoritmos de control [4].

Este trabajo se centra en el sistema de control, es decir, dotar al robot con algoritmos de control para que sea capaz de seguir una trayectoria para alcanzar una posición final.

### A.7.1. Autonomía de los robots

El objetivo de la robótica móvil es, claramente, conseguir la autonomía de dichos sistemas para que sean capaces de ejecutar una tarea determinada sin la necesidad de la intervención humana. Para conseguir dicha autonomía, los robots han de ser capaces de orientarse y elegir la ruta adecuada para desplazarse sin colisionar con obstáculos que pueden encontrarse en el medio. Además, hay veces que el evadir los obstáculos no es el único requisito, sino que encontrar la ruta más corta que emplee el menor tiempo posible, es también un requisito [4].

La navegación de los robots se puede dividir en tres grandes problemas, uno de ellos es el de la localización, es decir, conocer en todo momento dónde se encuentra el robot, cuales son sus coordenadas respecto a un sistema de referencia. El segundo problema es la planificación de tareas, esto consiste en decidir en qué orden se van a realizar las distintas tareas, es dónde más alto nivel de razonamiento se quiere y suele estar basado en inteligencia artificial. Finalmente, el último problema es la planificación del movimiento, que es donde entran los algoritmos de cálculo de trayectorias para planear el camino a seguir hasta alcanzar un punto final [14].

# Bibliografía


[1] A. Yandún and N. Sotomayor, "Planeación y seguimiento de trayectorias para un robot móvil."

[2] C. Mahulea, M. Kloetzer, and R. González, *Path Planning of Cooperative Mobile Robots Using Discrete Event Models*, 1st ed. IEEE Press Series on Systems Science and Engineering, Wiley., 2020.

[3] P. Frana and T. Misa, "An interview with edsger w. dijkstra," *Communications of the ACM*, vol. 53, no. 8, pp. 41–47, 2010.

[4] M. R. Tapia García and J. M. Dr. López Hernández, "Robótica móvil," *Revista de divulgación científica*, vol. 3, no. 2, pp. 2526–2530, 2017.

[5] A. Muñoz Cueva, "Generación global de trayectorias para robots móviles, basada en curvas betaspline," Proyecto de fin de grado, Universidad de Sevilla, 2014.

[6] F. Prieto Rodríguez, "Métodos de generación de trayectorias," Proyecto de fin de grado, Universidad de Sevilla, 2017.

[7] F. Gómez Bravo, A. Ollero, D. López, F. Cuesta, M. del Toro, P. Gil, and F. Real, "Rrt-d : Planificación distribuida de caminos basada en la información de una red de sensores wireless."

[8] "Boost : dijkstra shortest paths." [Online]. Available: https://www.boost.org/doc/libs/1_53_0/libs/graph/doc/dijkstra_shortest_paths.html

[9] M. Peña Basurto and J. M. Cela Espín, *Introducción a la programación en C*, 1st ed. EDICIONS UPC, 2000.

[10] R. Marras. (27 de Agosto de 2014) Frameworks para robots (i): Ros. [Online]. Available: https://jjromeromarras.wordpress.com/2014/08/27/frameworks-para-robots-i-ros/

[11] About ros. [Online]. Available: https://www.ros.org/about-ros/

[12] D. Ortego Delgado. (2017) Qué es ros. [Online]. Available: https://openwebinars.net/blog/que-es-ros/

[13] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "Ros: an open-source robot operating system," 2009.

[14] G. Bermúdez, "Robots móviles. teoría, aplicaciones y experiencias," in *Tecnura 10*, 2002, pp. 6–17.