UNIVERSIDAD DE SALAMANCA

Departamento de Informática y Automática

Facultad de Ciencias

Cálculo de caminos óptimos para manipuladores articulados

Miguel Ángel Gutiérrez García

Julio 2004

D. **Vidal Moreno Rodilla**, Prof. Titular de Ingeniería de Sistemas y Automática de la Universidad de Salamanca y Da. **Belén Curto Diego**, Pfa. Titular de Ingeniería de Sistemas y Automática de la Universidad de Salamanca

CERTIFICAN:

Que el trabajo de investigación que se recoge en la presente memoria, titulada **Cálculo de caminos óptimos para manipuladores articulados** y presentada por D. **Miguel Ángel Gutiérrez García** para optar al grado por la Universidad de Salamanca, ha sido realizado bajo su dirección en el Area de Ingeniería de Sistemas y Automática del Departamento de Informática y Automática de la Universidad de Salamanca.

Salamanca, 15 de Julio de 2004

D. **Vidal Moreno Rodilla**
Prof. Titular de Universidad
Área de Ingeniería de Sistemas y Automática
Universidad de Salamanca

Da. **Belén Curto Diego**
Pfa. Titular de Universidad
Área de Ingeniería de Sistemas y Automática
Universidad de Salamanca

Índice general

Índice de figuras

1

Introducción

Desde que en la década de los 60 comenzó un nuevo campo en la tecnología con el desarrollo de los robots, éstos se han adaptado de forma óptima en algunos sectores como la industria del automóvil o la fabricación. Las tareas que éstos realizan en estos lugares son íntegramente repetitivas (soldadura, paletización, montaje) ejecutando secuencias de movimientos, que previamente han sido programadas por un operador. Estas acciones son realizadas de forma sistemática en un entorno de trabajo estructurado en el que todos los elementos trabajan de una forma totalmente sincronizada.

La simple utilización de los robots en este tipo de tareas los convierte en herramientas, que aunque pueden ser reprogramados para otra utilidad, no utilizan la gran versatilidad que disponen. Esta es la razón por la que los principales centros de investigación en Robótica han orientado desde hace tiempo sus esfuerzos

hacia los sistemas robóticos autónomos. Esto es, dotarlos de cierto grado de "inteligencia" de manera que sean capaces de solventar los problemas que puedan plantearse cuando deben realizar un cierto trabajo, de manera que el usuario tan sólo tenga que decir lo que se desea hacer y no cómo hacerlo, como se hacía con el modo de programación previo. Los robots autónomos son especialmente interesantes para diversas aplicaciones como agricultura, construcción, ayuda a discapacitados, fabricación, exploración espacial y submarina, etc. En estas aplicaciones no existe un conocimiento detallado del entorno, las tareas son poco repetitivas e incluso se realizan en entornos peligrosos o en los que no es posible el acceso a personas (ambientes contaminados).

Conseguir que los robots sean autónomos conlleva que tengan que tomar decisiones y ejecutarlas. Esta necesidad de tener que ejecutar una decisión tomada anteriormente lleva consigo ejecutar una serie de movimientos para conseguirlo, y, por lo tanto, para lograrlo se requerirá una capacidad de planificar sus movimientos, siendo en esta última parte donde se va a centrar este trabajo.

Esta memoria esta estructurada en cinco apartados. En la introducción se presentarán una serie de aspectos generales a tener en cuenta a la hora de intentar conseguir robots autónomos. Así como se plantean también los objetivos de este trabajo.

En el siguiente, se hace una revisión con mayor detalle de los aspectos más relacionados con el proceso en sí del cálculo de caminos. En él se describen diversas técnicas existentes.

A continuación, en el capítulo tercero, se describen y clasifican los distintos algoritmos de búsqueda existentes.

En el capítulo cuarto se describen de manera detallada los algoritmos y que técnicas que se van a implementar en este trabajo basándonos en su aplicación en el caso sobre el que vamos a trabajar.

En el capítulo quinto se presenta la herramienta de simulación desarrollada para poder estudiar los algoritmos implementados. Se explica el manejo de la herramienta y como interpretar los resultados obtenidos.

En el capítulo sexto se presentan los resultados obtenidos al probar los algoritmos implementados sobre distintos mapas y destinos. Sobre estos resultados se obtendrán las conclusiones del trabajo, presentadas en el último capítulo.

1.1. Estructura de tareas

A un ser humano, aparentemente, la capacidad de planificar sus propias acciones no le plantea gran dificultad puesto que, en buena parte, la interacción con su entorno se realiza de forma inconsciente. Esta inconsciencia se torna en desconocimiento a la hora de intentar duplicarla en un programa de ordenador que se encargue de controlar un robot. Para llevar a cabo la emulación de actividades realizadas por el ser humano se necesitan desarrollar tecnologías como el razonamiento automatizado, la percepción y el control. Para dar una idea de la complejidad del tema, en los siguientes apartados se van a comentar las disciplinas y las tareas implicadas más directamente en el desarrollo de robots autónomos, tomando como base el diagrama de la figura 1.1.

Cuando un robot realiza una tarea ejecutando secuencias de movimientos en un espacio de trabajo ocupado por objetos, es necesario, por un lado estudiar su cinemática y por otro, disponer de una descripción del entorno y de la descripción geométrica del robot, lo que constituye el objetivo de los apartados siguientes.

1.1.1. Tipos de robots

Para poder comenzar con cualquier tipo de estudio sobre los robots y sobre todo si se pretenden realizar cálculos de caminos, es fundamental conocer cuales son sus componentes y cómo esta ordenada su estructura. Por esa razón es conveniente antes de comenzar realizar un breve análisis estos en cuanto a su estructura geométrica.

Una primera y simple clasificación de los robots en función de sus características mecánicas es:

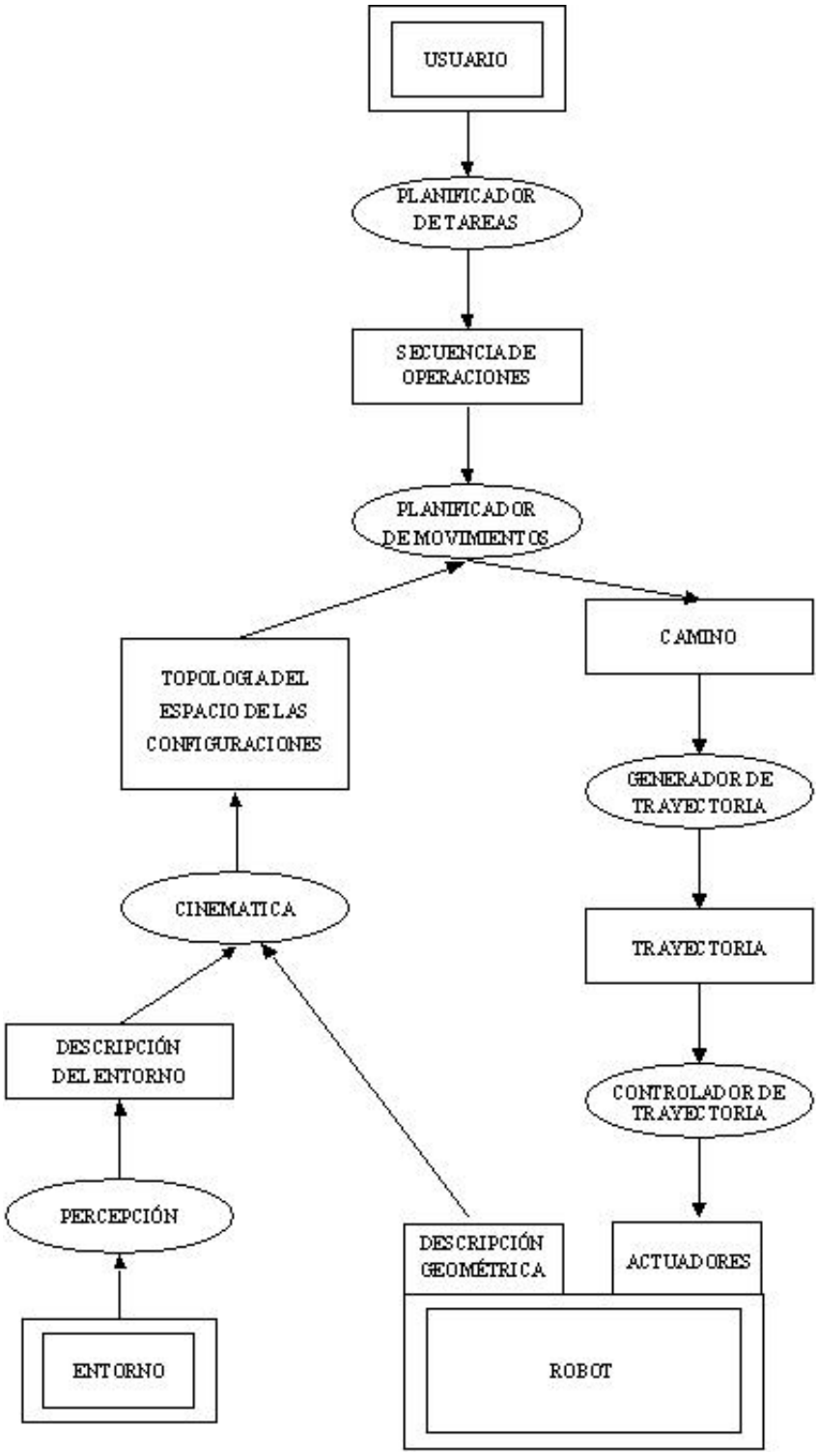


Figura 1.1: Tareas en la planificación de trayectorias

- Fijos o manipuladores: en los que la base del robot está fija en un punto del espacio en el que se encuentra.
- Móviles: en éstos la base no está fija y se puede mover con una cierta libertad por el espacio (aunque puede existir algún tipo de restricción).

En lo que concierne a los robots fijos están compuestos por varios elementos conectados por articulaciones en forma de una cadena cinemática. Las articulaciones pueden ser de dos tipos, de revolución que permiten una rotación entre dos elementos, y prismáticas que permiten un movimiento lineal relativo entre los dos elementos. El número de articulaciones determina los grados de libertad del manipulador. Los manipuladores deben tener seis grados de libertad independientes: tres para determinar la posición del efector final y otros tres para determinar su orientación. De otra forma si fueran menos no se podría llegar a cualquier posición de espacio con cualquier orientación y si fueran más se podría alcanzar esa posición y orientación de infinitas formas, en cuyo caso se trataría de robots con cinemática redundante.

El espacio de trabajo del manipulador es el conjunto total de puntos que pueden ser alcanzados por el efector final según el manipulador ejecuta todos sus posibles movimientos. Este espacio de trabajo vendrá determinado tanto por la geometría del manipulador (tipos de articulaciones, tamaño y forma de los elementos) como por las restricciones que existan en las articulaciones (una articulación de revolución en general tendrá una amplitud de giro de menor de 360^0). El espacio de trabajo se suele subdividir en dos partes, el espacio de trabajo alcanzable que es el conjunto de puntos accesibles y el espacio de trabajo hábil que es el conjunto de aquellos puntos que pueden ser alcanzados con cualquier orientación del efector final.

Los manipuladores se pueden clasificar siguiendo varios criterios, tales como la geometría, estructura cinemática, la aplicación para la que son diseñados, el tipo de control, etc. Una de las clasificaciones más comunes es la que se refiere al tipo de las tres primeras articulaciones. Según esta clasificación existen diversos tipos: articulados (RRR), esféricos (RRP), SCARA (RRP), cilíndricos (RPP) y cartesianos (PPP).

El otro tipo de robots es el de los robots móviles, que son aquellos en los que la base no esta fija, tiene cierta libertad para moverse por el espacio, ya sea en el plano o en el espacio tridimensional. Estos robots pueden estar únicamente formados por la base o también por una estructura similar a la de los manipuladores con la única excepción y complicación respecto a su controlabilidad de que la base no está fija.

1.1.2. El espacio de las configuraciones

El espacio de las configuraciones aparece con la intención de encontrar un espacio de manera que se pueda hacer una representación del robot como un único punto. Para definirlo es clave definir previamente lo que se entiende por una configuración. Una configuración es una especificación de forma única la posición de todos los puntos del robot. Por lo tanto estará directamente relacionada con la cinemática del robot concreto y de esta forma con los grados de libertad del mismo. El espacio de las configuraciones será entonces el espacio C formado por todas las posibles configuraciones del robot.

Con objeto de aclarar e concepto, supóngase que el robot es un disco que puede moverse y girar libremente en el plano con unas determinadas dimensiones [0,a] x [0,b], tal y cómo aparece en la figura 1.2. Para describir una configuración arbitraria del robot para un determinado punto del robot hay que especificar las coordenadas de ese punto, con respecto a un sistema de referencia fijo. Por tanto, una configuración vendría dada por una tupla de dimensión dos (x,y). Su espacio de las configuraciones estaría formado por todos los pares de coordenadas (x,y), donde x e y son los números reales que pueden tomar valores en un intervalo cuyas dimensiones dependen de las del espacio de trabajo, así como de la geometría del robot.

Otro ejemplo es el del manipulador planar, que dispone de dos elementos rígidos unidos por articulaciones de revolución con ejes de giro paralelos como se muestra en la figura 1.2. Para describir una configuración arbitraría del robot se necesita conocer el valor de los ángulos girados por las articulaciones θ_1 y θ_2. Por lo tanto una configuración vendrá determinada por la tupla de dimensión dos

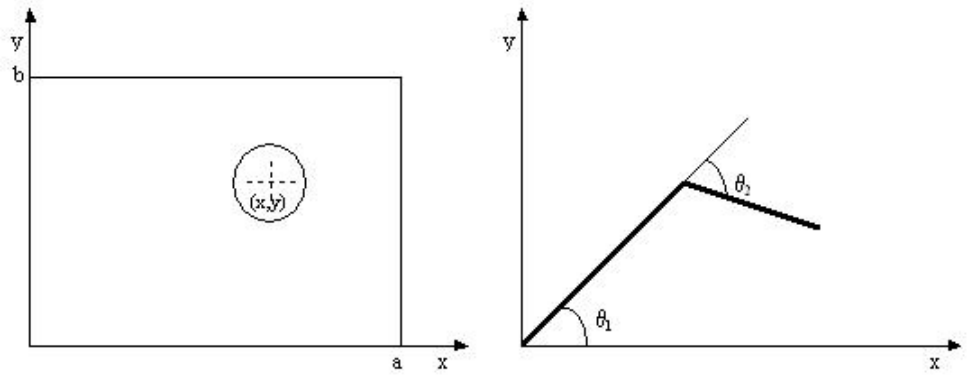


Figura 1.2: Robot móvil y robot articulado en el plano

(θ_1,θ_2). Así, el espacio de las configuraciones estará formado por todos los pares de coordenadas (θ_1,θ_2) donde θ_1 y θ_2 son números reales dentro del intervalo [0,2π).

En el capítulo de planificación se realizará un análisis más detallado del concepto del espacio de las configuraciones introduciendo una definición geométrica.

1.1.3. Percepción

Los manipuladores más implantados en la industria desarrollan su trabajo en un entorno perfectamente estructurado, donde se conocen de antemano las posiciones donde se encuentran los objetos que tienen que manejar o que forman parte del mobiliario industrial. Sin embargo, los robots autónomos necesitan disponer de información del entorno con el cual deben interaccionar. Por ejemplo, si se trata de un robot móvil encargado de transportar objetos por un pasillo es necesario que disponga de la situación de los elementos de su entorno, que lógicamente se modifica. También es interesante disponer de manipuladores capaces de adecuarse a entornos cambiantes.

En este sentido, la capacidad sensorial del robot se encarga de realizar las labores de percepción de información y juega un papel fundamental en el desarrollo de los sistemas robóticos inteligentes. En principio, hay una gran variedad de sensores que son utilizados en Robótica, pero los de visión ocupan un papel

fundamental. A partir de la información recogida por ellos se puede llegar a disponer de una descripción del entorno, en concreto de todos aquellos puntos del espacio de trabajo que están ocupados por los objetos.

En este trabajo se partirá de una información exacta y detallada del entorno que deberá ser sido proporcionada con anterioridad o que podrá generarse a través de la herramienta de simulación desarrollada.

1.1.4. Planificación de tareas

Como se pude observar en la figura 1.1, la planificación de tareas se encuentra en el nivel superior de esta jerarquía de disciplinas implicadas en la creación de robots autónomos. Un planificador de tareas se puede ver como un bloque lógico con una entrada y una salida. La entrada sería una descripción de la tarea que el usuario desea que ejecute el robot y la salida sería el conjunto de operaciones necesarias para su realización.

Por ejemplo, supóngase que el robot es un manipulador que se encuentra en una planta de fabricación de circuitos electrónicos. La entrada al planificador de tareas sería "montar el circuito X". La salida dividiría la tarea en las siguientes operaciones:

1. Coger la tarjeta vacía y colocarla en un soporte,
2. recoger cada uno de los componentes necesarios y colocarlos en las posiciones debidas,
3. transportar la tarjeta a la línea de embalado.

El paso 2 se subdividirá a su vez en otros pasos más sencillos para cada uno de los componentes (recoger el componente, colocarlo en su posición y soldarlo).

A pesar de la sencillez del ejemplo, en general, la etapa de planificación de tareas puede ser sumamente compleja, porque la tarea en si misma lo sea o porque se deseen introducir mejoras adicionales, como pueden ser la tolerancia a fallos, la optimización del tiempo de ejecución, la integración con el resto de los componentes de la línea de fabricación, etc.

1.1.5. Planificación de movimientos

Se ha mostrado como el planificador de tareas ha desglosado una tarea compleja en un conjunto de operaciones más sencillas, donde cada una de éstas consiste en recoger una pieza de un determinado punto del espacio y colocarla en otro. El planificador de movimientos que será la tarea en la que se centrará fundamentalmente este trabajo, y se encarga de encontrar un camino que conecte el punto origen y el punto destino sin que se produzcan colisiones con los obstáculos que ocupan parcialmente su entorno de trabajo, o devolver un fallo si este camino no existe.

Por ejemplo, para la primera operación del montaje de la tarjeta electrónica, las entradas al planificador de movimientos serán la posición A donde se encuentra la tarjeta vacía y la posición B del soporte donde debe ser colocada. Su salida sería la secuencia de movimientos que le permiten desplazarse desde A hasta B, sin que choque con los objetos que se encuentran en su entorno de trabajo o viole alguna de las restricciones que el robot mismo impone sobre su movimiento.

Un punto de vista incorrecto y muy extendido es el que considera que la planificación de movimientos consiste esencialmente en detectar colisiones. Además, se ocupa de calcular caminos libres de colisiones entre obstáculos móviles, de coordinar el movimiento de varios robots, de planificar movimientos para empujar y deslizar objetos con el fin de lograr relaciones exactas entre estos, de planificar la manera de coger los objetos de forma estable, etc.

1.1.6. Control de movimientos

El resultado obtenido por este planificador de movimientos es un camino que especifica la secuencia continua de configuraciones libres que el robot debe recorrer para alcanzar la configuración final. La tarea fundamental del controlador en tiempo real es lograr que el robot realice esta secuencia de movimientos, es decir sigue el camino generado, actuando sobre él. En concreto, dado un camino, el controlador tiene que encontrar la función temporal que define los pares a

aplicar a los actuadores del robot en cada instante de tiempo y aplicarlos.

Normalmente esta etapa se puede dividir en dos pasos. El primero, denominado generación de trayectorias, consiste en definir el perfil de velocidad a lo largo del camino. Este paso se puede realizar antes de la ejecución del movimiento. El segundo, denominado seguimiento de la trayectoria, consiste en calcular los pares que se deben aplicar a los actuadores en cada momento para realizar el movimiento deseado. En este paso se utiliza, directa o indirectamente la dinámica del robot para calcular el par que tiene que ser aplicado a cada actuador. Si la dinámica utilizada por el controlador fuera un modelo perfecto, no sería necesario disponer de un sistema realimentado. Sin embargo, debido a diferentes perturbaciones e inestabilidad, es necesario disponer de sensores para determinar la desviación entre el estado deseado y el estado actual del robot. Mientras se está ejecutando el movimiento, el controlador calcula los pares que tienden a eliminar esta desviación.

La figura 1.3 muestra la relación entre el planificador de movimientos, el generador de trayectorias, el controlador y el robot. El camino calculado por el planificador de movimientos es la entrada al generador de trayectorias, que determina la dependencia temporal de los parámetros que definen una configuración. La salida del generador de trayectorias, las configuraciones deseadas q_d como una función del tiempo, se introduce en el controlador. Este calcula los pares P que han de ser aplicados a cada actuador, a partir de la dependencia entre la configuración actual q_s, medida por los sensores, y la configuración deseada q_d.

La arquitectura presentada en la figura 1.3 es clásica, ya que separa la etapa de planificación de movimientos de la etapa de control. Su ventaja al dividir el problema es decidir un movimiento en varias etapas claramente definidas.

1.2. Objetivos del trabajo

Una vez que se han establecido las técnicas implicadas en el desarrollo de los robots autónomos, se van a presentar los objetivos perseguidos en este trabajo

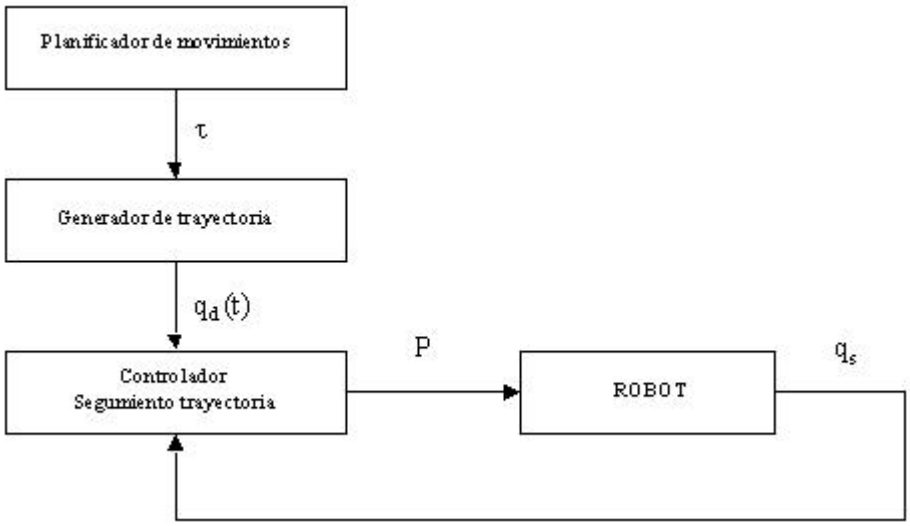


Figura 1.3: Controlador de movimientos

de investigación.

Así, se plantea que de forma general los esfuerzos se van a centrar en el problema de la planificación de movimientos de estructuras robóticas. Como se ha comentado anteriormente, el objetivo planteado es la obtención de una secuencia de posiciones libres de colisiones que conectan una configuración inicial con una final. Aunque, como se ha comentado, existen trabajos que realizan esta tarea en el espacio de trabajo, nos planteamos la utilización del espacio de las configuraciones. En este sentido, técnicas de la Inteligencia Artificial como son las estrategias de búsqueda ocuparán sin duda un papel fundamental. Por esta razón se plantea un objetivo inicial como es la evaluación de diferentes algoritmos que implementan dichas estrategias, realizando para ello la implementación de los más usados y optimizándolos para el caso particular del cálculo de caminos óptimos libres de colisiones para robots articulados en este caso. Se evaluarán los algoritmos válidos atendiendo tanto a entornos estáticos como a dinámicos.

Como se analizará posteriormente un elemento central en las estrategias de búsqueda es la utilización de información (denominada habitualmente heurística) acerca del problema concreto de la planificación. Se constituye así el siguiente objetivo particular con la propuesta de heurísticas para el guiado de estos algo-

ritmos tomando como información de partida las distancias al punto de destino así como la existencia de los obstáculos. Se planteará, para este fin, la utilización del concepto de campos de potencial.

Con objeto de evaluar el comportamiento de estos algoritmos, se plantea el desarrollo de un entorno gráfico que permita generar diferentes entornos de trabajo y la posterior visualización de los caminos libres de colisiones planificados. Esta visualización deberá plantearse tanto en el espacio de trabajo como en el espacio de las configuraciones.

Como se ha señalado, se considerará como caso de estudio un robot articulado de dos articulaciones que es una estructura que aparece en buena parte de los trabajos de investigación relacionados. Será necesario, para dotar de validez a los trabajos realizados permitir variar las características básicas del robot. Se completan así los principales objetivos del presente trabajo de investigación.

2

Planificación de caminos

En este capítulo se desarrollará una descripción de las técnicas de planificación más importantes que han sido propuestas. Para ello se comenzará con el problema básico de planificación de movimiento con el objetivo de proporcionar una idea general y definir de manera formal el espacio de las configuraciones. A continuación se expondrá una clasificación de los diferentes métodos de planificación con sus principales características y ventajas.

Para que cualquier tipo de manipulador articulado realice un movimiento se tiene que llevar a cabo una serie de actuaciones sobre sus articulaciones. De esta forma, al moverse cada una de sus articulaciones, la estructura total del robot realizará un movimiento. Por lo tanto, el aspecto fundamental a la hora de realizar un movimiento radica en variar sus articulaciones. El estudio de cómo afecta el movimiento de las articulaciones en el movimiento del robot

se realiza mediante la cinemática directa, la cual proporciona la posición de todos los puntos del robot para cada configuración. Dado que cada configuración determina de manera única el estado de cada una de las articulaciones, se puede trabajar sobre el espacio de las configuraciones, que es el espacio formado por todas las configuraciones posibles y en el que se puede representar el robot como un único punto en vez de toda su estructura. Por esa razón se desarrolla a continuación la definición formal del espacio de las configuraciones.

2.1. Problema básico

En primer lugar considérese el problema básico en el que el robot solamente está formado por un elemento rígido. Sea A el robot en una cierta posición y orientación representado como un subconjunto compacto (cerrado y acotado) del espacio euclideo W=$\Re^N$, N=2 ó 3, que a partir de ahora se denominará espacio de trabajo, y los obstáculos como los subconjuntos cerrados de W, B_1, ..., B_m. Sean también F_A y F_W los sistemas de coordenadas cartesianas asociados a A y W respectivamente. F_A es un sistema de coordenadas que se mueve con A, mientras que F_W es fijo. Por definición como A es rígido, cualquier punto a de A tiene una posición fija respecto a F_A. Pero, sin embargo, la posición de A respecto F_W depende de la posición y orientación de F_A respecto de F_W.

Una configuración del robot A es una especificación de la posición y orientación de cada punto del robot respecto del sistema de coordenadas fijo F_W. Por lo tanto una configuración q de A es una especificación de la posición τ, y la orientación θ de F_A respecto F_W. El espacio de las configuraciones de A es el espacio C de todas las configuraciones de A. El subespacio ocupado por A en W con una determinada configuración q se denotará por A(q), y de la misma manera, el punto a de A en la configuración q se denotará por a(q).

Una configuración se va a poder describir por una lista de parámetros reales. Por ejemplo, la posición τ se puede describir como el vector de N coordenadas desde el origen de F_A en F_W. La orientación θ se puede describir como una matriz NxN cuyas columnas sean las componentes, en F_W, de los vectores uni-

tarios de F_A. Entonces q=(τ,θ) se puede representar de forma única por una lista de N(N+1) parámetros. Esta representación, sin embargo, es redundante puesto que la matriz que describe la orientación tiene que tener las columnas ortonormales y el determinante +1. Por lo que C es tan solo un subespacio de $\hat{\mathrm{A}}^{N(N+1)}$. Una vez que se disponga de una estructura robótica determinada se puede reducir de forma considerable la dimensión de C.

Puesto que ya se ha definido el concepto de espacio de las configuraciones lo que queda ahora es la representación de los obstáculos del espacio de trabajo en éste.

Para todo B_i, $i = 1, .., q$, en el espacio de trabajo W se representa en C la región $CB_i = \{q \in C \mid A(q) \cap B_i \neq \phi\}$ que se denomina C-obstáculo. La unión de todos los conjuntos $\cup_{i=1}^{m} CB_i$ se llama región de C-obstáculos, y el subconjunto $C_{libre} = C \mid \cup_{i=1}^{m} CB_i = \{q \in C \mid A(q) \cap (\cup_{i=1}^{m} B_i) = \phi\}$ se llama espacio libre.

En el caso más complejo en el que sea un robot articulado, el robot A estará formado por varios elementos rígidos A_1, ..., A_p conectados mediante articulaciones de revolución o prismáticas. El espacio de configuraciones del robot sería entonces la composición de los espacios de configuraciones de cada uno de los elementos. Pero, sin embargo, las restricciones impuestas por las articulaciones sobre los posibles movimientos de cada uno de los objetos provocan que el espacio de configuraciones se convierta en un subespacio del total. Este subespacio se va a poder parametrizar, en general, por el desplazamiento lineal o el ángulo de giro de cada articulación.

2.1.1. Ruta libre de colisiones

Una vez que se ha introducido el espacio de las configuraciones se puede definir el concepto de *ruta libre de colisiones* sobre este espacio, que determina la ruta seguida por cada uno de los puntos del robot en el espacio de trabajo.

La noción de continuidad es fundamental en la definición de una ruta. Su formación requiere la definición de una topología en C. Una forma clásica de

hacerlo es especificar una función distancia: $d : CxC \rightarrow \Re$ y permitir que la topología en C sea la inducida por la función d. De forma más intuitiva, la distancia entre dos configuraciones q y q' debería disminuir y tender a cero cuando las regiones A(q) y A(q') se vayan acercando y tiendan a coincidir. Una función distancia simple es aquella que se define como:

$$d(q, q') = max_{a \in A} ||a(q) - a(q')||$$

donde $||x - x'||$ representa la distancia euclídea entre dos puntos cualesquiera x y x' de $\Re^N$.

Una ruta en C desde la configuración $q_{inicial}$ hasta la configuración q_{final} viene dada por la función continua:

$$\tau : [0, 1] \rightarrow C \text{ donde } \tau(0) = q_{inicial} \text{ y } \tau(1) = q_{final}.$$

La continuidad de τ significa que:

$$\forall s_0 \in [0, 1], lim_{s \rightarrow s_0}(max_{a \in A} ||a(\tau(s)) - a(\tau(s_0))||) = 0$$

con s tomando valores en el intervalo [0,1].

Por lo tanto una ruta libre de colisiones entre dos configuraciones libre $q_{inicial}$ y q_{final} es una función continua $\tau : [0, 1] \rightarrow C_{libre}$ con $\tau(0) = q_{inicial}$ y $\tau(1) = q_{final}$. Se dice además que dos configuraciones pertenecen a la misma componente conectada de C_{libre} si y sólo si están conectadas por un camino libre.

2.2. Métodos de planificación locales y globales

Una primera y amplia clasificación de las técnicas de planificación se basa en la información utilizada a la hora de realizar la planificación. De esta forma, se pueden clasificar en técnicas locales o globales.

Las técnicas locales tan sólo tienen en cuenta una porción del espacio total en el que se realiza el movimiento. Así, se realiza el movimiento de una configuración a la siguiente de la ruta teniendo en cuenta las características de un reducido entorno a su alrededor. Estas técnicas tienen la ventaja de que tienen una reducida carga computacional pero tienen grandes desventajas como:

- En numerosos casos no se alcanza una solución óptima.
- Pueden dar lugar a oscilaciones en pasillos estrechos.
- En algunos casos no se encuentra la solución al llegar a un lugar del que estas técnicas no permiten salir, como puede ser el caso de las técnicas basadas en potenciales explicadas en el apartado siguiente, y la aparición de mínimos locales.

Las técnicas globales tratan de encontrar la ruta completa teniendo en cuenta la conectividad de todo el espacio libre. Al considerar todo el espacio libre la complejidad computacional es muy elevada con la consiguiente lentitud. Pero sin embargo, tiene la ventaja de que se puede conseguir una ruta óptima.

2.3. Técnicas de planificación globales

Existe un gran número de métodos para resolver el problema de la planificación, pero una gran cantidad de ellos no son capaces de resolver el problema en su generalidad. Por ejemplo, algunos métodos requieren que el espacio de trabajo sea bidimensional o que los obstáculos sean poligonales.

De cualquier forma, todos estos métodos, a pesar de sus diferencias se pueden clasificar en tres grandes grupos: mapas de carreteras, descomposición por celdas y por campos de potencial.

A continuación se va a realizar una breve descripción de cada uno de ellos.

2.3.1. Mapa de carreteras

Las técnicas basadas en un mapa de carreteras consisten en encontrar la conectividad del espacio libre para el robot mediante una malla de curvas monodimensionales que es lo que se denomina el mapa de carreteras. Una vez que el mapa de carreteras (roadmap) R se ha calculado la planificación se reduce a conectar las configuraciones inicial y final con puntos de R y buscar en R un camino entre estos puntos. De esta forma el camino así construido es el resultado de la concatenación de tres caminos: uno que conecta la configuración inicial con el mapa de carreteras, otro contenido en el roadmap y el último que conecta el roadmap con la configuración final.

Existen varios métodos basados en esta idea general pero los más importantes son el del grafo de visibilidad y el diagrama de Voronoi.

El método del grafo de visibilidad es uno de los primeros métodos que aparecen para realizar la planificación, el cual se aplica sobre espacios de configuraciones de dimensión dos y con obstáculos poligonales. El grafo está formado por las líneas rectas que unen los vértices de los obstáculos tales que no intersectan con el interior de alguno de los otros obstáculos. Por lo tanto se forma un entramado de rectas que cubren todo el espacio, y el camino resultante irá de vértice a vértice por el espacio desde la configuración inicial a la final.

En la figura 2.1 se puede observar el grafo de visibilidad para un espacio con objetos poligonales. En él, dos puntos han sido conectados mediante una camino formado por líneas rectas.

El otro método, el diagrama de Voronoi, es una particularización para espacios de configuraciones de dimensión dos de un método más general llamado retracción. El diagrama está formado por el conjunto de puntos del espacio libre cuya mínima distancia a los obstáculos se alcanza por lo menos en dos puntos de la frontera de estos.

En la figura 2.2 se puede observar un diagrama de Voronoi con obstáculos poligonales.

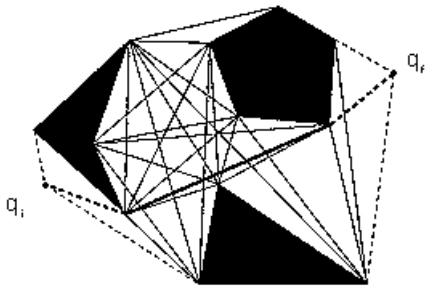


Figura 2.1: Grafo de visibilidad

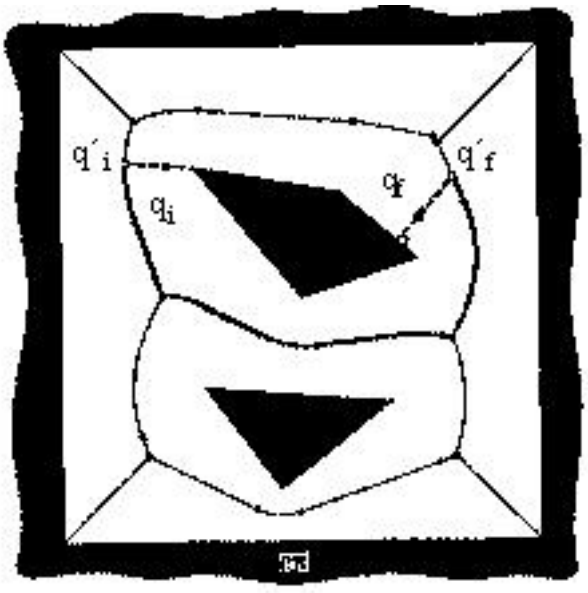


Figura 2.2: Diagrama de Voronoi

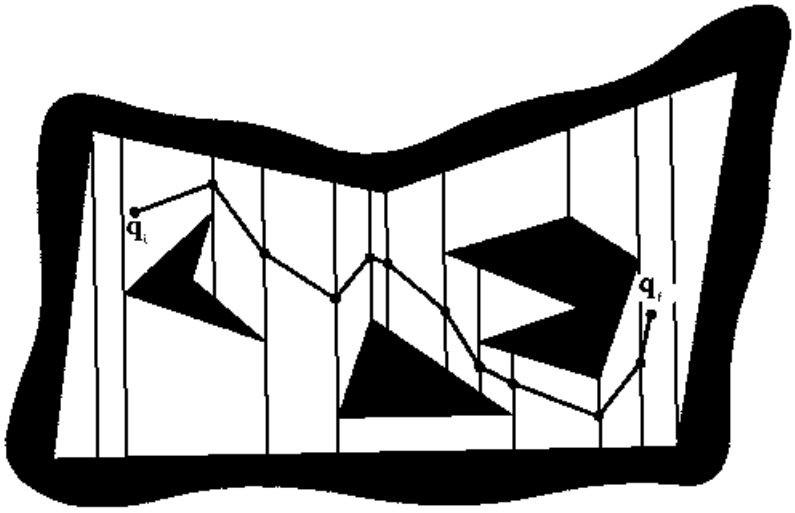


Figura 2.3: Descomposición por celdas de una región

2.3.2. Descomposición por celdas

Los métodos de la descomposición por celdas consisten en descomponer el espacio libre en regiones más pequeñas, llamadas celdas, de tal forma que un camino entre dos configuraciones en una misma celda se pueda generar fácilmente. La conectividad entre las distintas celdas se va a dar mediante un grafo de conectividad, de forma que el camino estará representado por la sucesión de celdas por las que se tiene que pasar para llegar a la configuración final.

Los métodos de descomposición por celdas se pueden separar en dos:

- Exactos: se descompone el espacio libre en celdas cuya unión es exactamente todo el espacio libre.
- Aproximados: se producen celdas de una forma determinada (p. e. rectangular) cuya unión está incluida estrictamente en el espacio libre. Este método fue introducido por [LP83] y Gouzenes lo implementó para manipuladores con articulaciones de revolución [Gou84].

Los métodos exactos aseguran el encontrar un camino si es que existe alguno, lo cual no aseguran los aproximados. Pero sin embargo, los aproximados van a

ser más fáciles de implementar y además la precisión se va a poder hacer mayor dependiendo del tamaño de las celdas.

En la figura 2.3 se muestra una descomposición por celdas exacta con la secuencia de celdas a seguir para conseguir la ruta.

2.3.3. Campos de potencial

Otra forma para encontrar el camino es discretizar el espacio de configuraciones en una fina malla regular de configuraciones y encontrar aquí el camino. Esta forma implica el utilizar heurísticas para guiar la búsqueda. Algunas de estas heurísticas se pueden interpretar como campos de potenciales. La introducción del concepto del campo de potencial en la planificación de movimientos se debe a Khatib [Kha86].

La interpretación física de los campos de potenciales puede entenderse como si existieran una serie de fuerzas, atractivas hacia la configuración final y repulsivas desde los obstáculos que generan estos potenciales y que guían al robot por la región libre hacia la configuración final. Estas fuerzas serían el gradiente negado de un potencial, por lo que se puede pensar en una superposición de los potenciales asociados a estas fuerzas.

Una posiblidad sería que el potencial repulsivo fuera igual al inverso de la mínima distancia a los obstáculos y el atractivo fuera igual al cuadrado de la distancia a la configuración final. Sin embargo se pueden definir numerosos tipos de potenciales para hacer la planificación, pero se presenta el problema de que según se sigue el sentido opuesto del gradiente se llegue a un punto en el que se alcance un mínimo local del potencial del que fuera difícil escapar. Este problema se puede resolver si se utilizan unos potenciales que no presenten ningún mínimo local salvo el final [Kod87], o si se utilizan algunos mecanismos muy potentes para escapar de los mínimos locales una vez que se han alcanzado [JB91].

En la figura 2.4(a) se puede observar un espacio de trabajo ocupado por dos obstáculos (triángulo y cuadrado). En la figura 2.4(b) se muestra el potencial atractivo hacia la configuración final q_f desde la configuración inicial q_i. En la

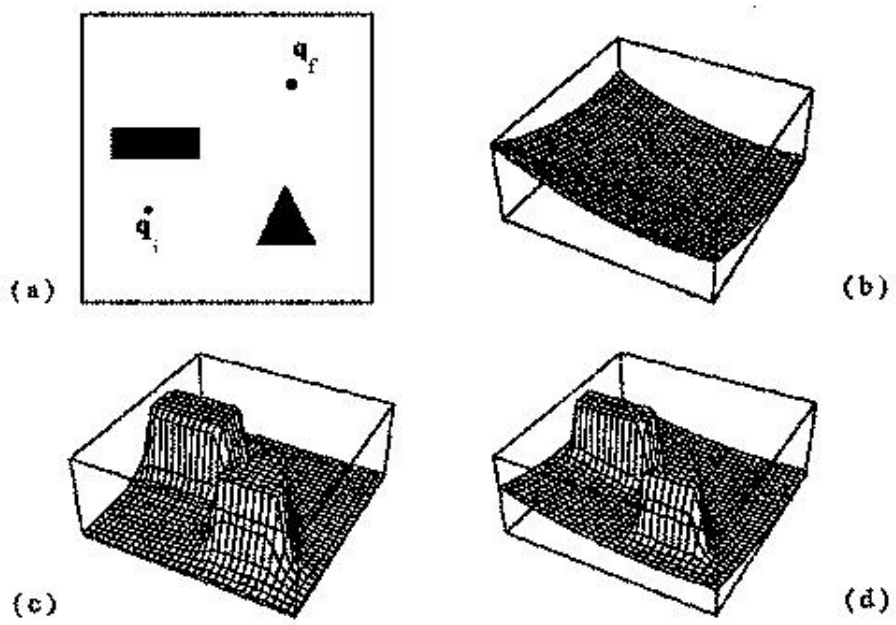


Figura 2.4: Campos de potencial

figura 2.4(c) el potencial generado por los obstáculos y, finalmente, en la figura 2.4(d) aparece el campo de potencial resultante de la superposición de ambos. Este campo de potencial generará un campo de fuerzas que guiará al robot en la búsqueda de un camino hacia la configuración final.

2.4. Revisión histórica

En este apartado se realizará una revisión histórica de los trabajos que utilizan alguna de las técnicas mencionadas en este apartado y que están directamente implicadas en el trabajo aquí presentado. En primer lugar el cálculo del espacio de las configuraciones, puesto que será el punto de partida para el cálculo de caminos en este trabajo. Posteriormente el empleo de campos de potenciales para el cálculo de caminos.

2.4.1. Espacio de las configuraciones

La idea de reducir el robot a un punto en un espacio apropiado fue introducida por Udupa [Udu77], aunque el término "espacio de las configuraciones" todavía no se había utilizado. Más tarde, Lozano-Perez [LP83] adoptó la noción de "espacio de las configuraciones" de la Mecánica y la popularizó en la planificación de movimientos. En algunos trabajos se calcula el espacio de las configuraciones mediante métodos geométricos tanto para obstáculos como robots con formas poligonal o poliédrica convexas [LP83] [LG86]. En otros se trata de hacer una representación en un bitmap, de tal manera que se pueden identificar "bloques elementales" que sean fácilmente transformados al espacio de las configuraciones de manera que combinados formen el espacio de configuraciones para formas más complicadas [WSN91].

En trabajos más recientes se propone calcular el bitmap del espacio de las configuraciones como la convolución del bitmap del espacio de trabajo con el del robot [Kav95] [BC97] [Rod97]. En estos se alcanzan unos tiempos de cálculo lo suficientemente pequeños como para poder utilizar el C-espacio en la planificación sin que esto suponga una carga adicional.

En otros trabajos se utilizan las propiedades geométricas de los obstáculos para la construcción del espacio de las configuraciones. Ejemplos los tenemos en: [HKL01], [WJ01], [Che96].

2.4.2. Campos de potencial

Uno de los primeros trabajos presentados sobre el uso de campos de potenciales para la planificación de caminos fue [HA88], donde se realiza un recorrido sobre las posibilidades que ofrece este método.

En [MOL+03] se utiliza el método de los campos de potenciales para representar el entorno del robot. El entorno es estático y de 2D. A cada uno de los obstáculos se les asigna un coeficiente de riesgo que se utiliza para aumentar el campo de repulsión entorno a algunos de los obstáculos. Se utiliza el algoritmo A* para la búsqueda del camino entre dos posiciones del mapa. También en en-

tornos 2D, en [EITE04] se utilizan los campos de potenciales para representar el entorno de un robot Nomad200, de forma que sea fácil encontrar caminos libres de obstáculos hasta las regiones no exploradas y construir el mapa del entorno.

En [GC02], [KLW89] y [Lee95] se proponen métodos que utilizan los campos de potenciales para realizar el cálculo de caminos para robots móviles en entornos dinámicos. En estos trabajos se propone una planificación para entornos dinámicos en los que se suponen conocidas la velocidad y aceleración de los obstáculos y el destino. Por ejemplo en [Lee95], el robot monitoriza el entorno a intervalos de tiempo regulares, y calcula, con los datos obtenidos al principio del intervalo, cuál será la zona del entorno ocupada por los obstáculos al final del mismo. Tras esto, se genera un campo de repulsión entorno al espacio que va a ser ocupado por los obstáculos y de atracción hacia el destino. En este trabajo, el robot se supone puntual y los obstáculos circulares. Un planteamiento muy similar se realiza en [GC02], salvo que no se monitoriza a intervalos el entorno del robot y los datos de la velocidad y aceleración de los obstáculos se suponen conocidos respecto al robot. Todos estos trabajos plantean su método de planificación sobre el espacio de trabajo y en entornos 2D.

Los métodos que utilizan los campos de potenciales sin mínimos locales tratan de encontrar un campo de potencial de manera que no presente ningún mínimo local salvo en la configuración final. Este método se utiliza en [JB91] y [WS00], donde se calcula el mapa de potenciales por medio de un mapa de carreteras previo calculado mediante un frente de ondas. En el método de planificación propuesto en [Rod97] se utilizan los campos de potenciales sin mínimos locales sobre el espacio de las configuraciones.

Se utilizan los campos de potenciales para la planificación de caminos para robots voladores (FFR) en entornos 3D estáticos en [Eln02], donde se utiliza una fuerza de atracción adicional para atraer al robot hasta una altura determinada. En entornos 3D dinámicos también se utilizan los campos de potenciales en [YK95] y [ZV97].

La técnica de los campos de potenciales también se utiliza combinada con modelos de fluidos para la planificación de caminos en [LB98] y [LL00]. Se

utiliza en mapas con distintos tipos de espacio libre (distintos tipos de suelo, por ejemplo) que pueden influir en el rendimiento del robot. El coeficiente de viscosidad del fluido se utiliza para controlar los caminos que ha de seguir el robot y las fuerzas del campo de potenciales sobre las partículas del fluido se utilizan para modelar la fricción entre el robot y el suelo.

Incluso se ha llegado a utilizar el método de los campos de potenciales en el encaminamiento de paquetes en redes de ordenadores en [BLR03] o modelos económicos como en [BDRT03].

3

Algoritmos de búsqueda

El problema de encontrar un camino en un espacio de tal forma que vaya desde una posición inicial a otra final en un principio puede pensarse que no requiere mucha inteligencia. Esto es, no es muy complicado para cualquier persona el decidir cómo se tiene que mover para llegar a un sitio determinado. Pero esta tarea que se realiza casi de forma inconsciente, para ser realizada por una máquina requiere técnicas propias de Inteligencia Artificial. Es más, el hecho de que se realice de forma inconsciente requiere una abstracción mayor para determinar cuáles son los elementos que se deben tener en cuenta.

Esta es la razón por la que a continuación se va a presentar el formalismo en el que están basados los diversos métodos para la búsqueda de solución dentro del problema de la búsqueda del camino que conecte una configuración inicial y una final. Posteriormente se presentarán las características de cada uno de ellos,

para poder seleccionar posteriormente el método más apropiado y determinar cuáles serán los criterios para decidir la dirección del movimiento.

3.1. Definición formal del problema

El primer paso será plantear la forma de abordar el problema. Este paso se debe realizar antes de comenzar a abordar un problema para determinar qué es lo que se pretende hacer y cuáles son las herramientas de las que se dispone. La definición del problema se realizará a través de una serie de pasos: definición de un ambiente de problema, definición de reglas, condiciones iniciales y objetivos.

- Definición de un ambiente de problema. Para ello se considera un conjunto de estados (el espacio de estados), de modo que cada uno de ellos defina de forma unívoca los elementos que afectan al problema. El espacio de estados será entonces la zona del espacio de configuraciones, puesto que por definición cada punto del espacio de las configuraciones del robot define la posición de todos los puntos del mismo.
- Definición de reglas. Estas se definirán de forma unívoca de modo que al aplicarlas a un estado, permitan alcanzar otros. Las reglas, en este caso, serán las posibles formas de pasar desde una configuración del robot a otra.
- Establecimiento de las condiciones iniciales. Se elegirá el estado que represente a la configuración inicial. Está claro que las condiciones iniciales vendrán representadas por la configuración de partida del robot.
- Definición de los objetivos, en el sentido de estimar qué estado o estados se desean alcanzar. El objetivo será simplemente alcanzar la configuración final.

Para que el proceso de búsqueda de la solución sea más efectivo y no se realice un excesivo cómputo cuando existe un elevado número de reglas, se debe definir una estrategia, que va a venir dada a través del concepto de sistemas de producción.

Los sistemas de producción nos van a proporcionar una separación de los componentes necesarios para poder hacer un desarrollo computacional de los problemas integrando los pasos fundamentales anteriormente citados. Estos componentes son:

- Base de datos global. Es una estructura central de datos en la que se tiene la descripción, de forma única, de la situación del problema a través del espacio de estados.
- Conjunto de reglas de producción. Estas operarán sobre la base de estados global. Cada regla de producción lleva asociada una precondición que debe ser satisfecha para su aplicación. Si se cumple la precondición se tendrá como resultado la modificación de la base de datos global. Además las reglas deben cumplir los requisitos siguientes:
  1. Todas las reglas acceden a la base de datos global.
  2. No pueden llamarse las unas a las otras.

  Las reglas de producción serán las posibles formas por las que se puede acceder desde una configuración a otra. En la precondición de las reglas tendríamos el test de colisiones.
- Sistema de control. Servirá para la elección de la regla más idónea en el caso de que existan varias aplicables. También tiene como misión que se llegue lo antes posible a la solución y por el camino más corto, si es que existen varios. El sistema de control lo que hará será determinar cuál de las reglas aplicables a una configuración proporciona otra configuración por la que se puede acceder a la solución.

Atendiendo a la información disponible sobre el problema en el proceso de búsqueda, los sistemas de control se pueden dividir en:

- Informados.
- No informados.

En los sistemas informados, el conjunto de información disponible se denomina heurística. Esta se representa a través de funciones de evaluación que

determinan los estados más idóneos a partir de los que se realiza la búsqueda. En este caso, en la función de evaluación se pueden incluir diferentes factores como la distancia a los obstáculos, coste en alcanzar esa configuración, etc.

3.2. Estrategias de búsqueda

Las estrategias de control, que definen como se va ha realizar el proceso de búsqueda, se pueden clasificar en:

- No tentativas o irrevocables. Serán aquellas caracterizadas por el hecho de que una regla seleccionada en un momento determinado no es reconsiderada posteriormente.
- Tentativas. En este caso se selecciona una regla aplicable, pero aún siendo aplicable se toman medidas con objeto de poder volver a este punto del proceso y aplicar otra distinta.

Las estrategias no tentativas se deben utilizar cuando se conoce con total seguridad cuál es la regla de selección que va a llevar a la solución. Como en este caso esta seguridad no va a existir nunca, se van a utilizar las estrategias tentativas, que se pueden clasificar a su vez en retroactivas (backtrack) y de exploración de grafos.

3.2.1. Backtrack

Mediante esta estrategia cuando se selecciona en un punto una regla a aplicar, se toman medidas para poder volver hasta este estado si se observa que esa regla no conduce a una solución. Los pasos dados que no conducían a la solución son olvidados y se selecciona otra regla de entre las aplicables a ese punto. Este tipo de estrategias son válidas tanto para sistemas sobre los que se dispone de información como para los que no, pero cuanto mayor sea la cantidad de información disponible para determinar cuáles de las reglas van a llevar a la solución, menor número de veces será necesario realizar un paso atrás (backtrack) y viceversa.

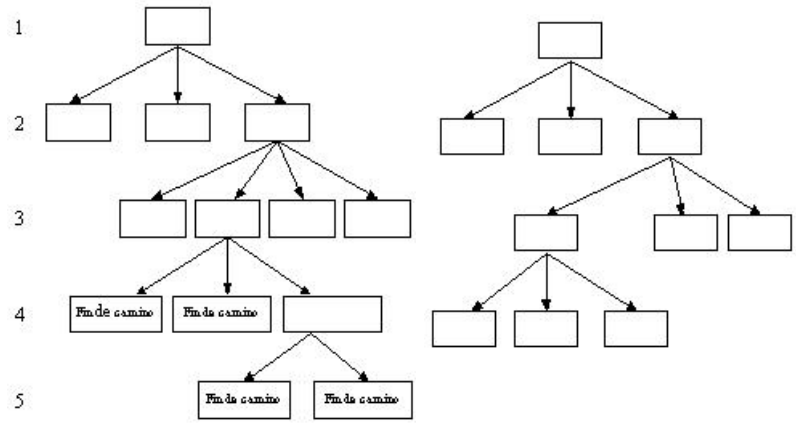


Figura 3.1: Paso atrás en backtracking

También hay que añadir que este método de búsqueda de solución está especialmente indicado para problemas que requieren sólo una pequeña cantidad de búsqueda. Las bases de datos a las que se debe aplicar son pocas. De igual forma, si se compara con la exploración de grafos (próximo apartado) es más sencilla de implementar y requiere mucho menor almacenamiento de información.

Los pasos atrás se realizan, como se indicó antes, cuando se dispone de la información suficiente que indique que la base de datos no conduce a una solución. Pero se tiene que, en general, no es posible tener la completa seguridad de cuándo sucederá esto y, por lo tanto, los pasos atrás se realizan en situaciones menos restrictivas, por ejemplo, cuando la secuencia tenga una longitud superior a una dada (límite de profundidad).

En la figura 3.1 se muestra cómo se realiza un paso atrás. En la parte de la izquierda se puede observar cómo se elige una regla de entre las aplicables, que posteriormente se comprueba que no conduce a solución puesto que se llega a lugares por donde no se puede continuar. Entonces, se produce un paso atrás, y en la parte de la derecha se elige otra regla de entre las aplicables olvidando todos los resultados anteriores.

3.2.2. Exploración de grafos

La exploración de grafos trata de resolver los problemas que se plantean en el backtrack con espacios de estados demasiado grandes. El principal problema es que no se puede asegurar con total seguridad que un estado no va a conducir a una solución, por lo que se puede producir un backtrack a una profundidad determinada que conduzca a no encontrar nunca una solución por estar ésta a mayor profundidad. Además también se presenta el problema de que al hacer un backtrack se olvida toda la información obtenida en ese intento. Otro problema es la pérdida de información al dar un paso atrás, puesto que se puede llegar a alcanzar por otro camino un estado que se hubiera alcanzado con anterioridad y que no condujera a solución, pero como esa información se ha perdido se tiene que volver a explorar.

Lo que se va a tratar de hacer es el generar un grafo (conjunto de nodos unidos entre si por un conjunto de arcos). En particular va a ser un grafo dirigido que tiene la particularidad de que los arcos van dirigidos de un nodo a otro, es decir, con un sentido. Así cuando se llegue desde el nodo n_i al nodo n_j, se dirá que "n_i es antecesor de n_j" o que "n_j es sucesor de n_i".

De esta forma se van a poder identificar los nodos con las bases de datos producidas y los arcos con las aplicaciones de las reglas. Así se va a formar un grafo partiendo del nodo origen (estado inicial) hasta que se verifique la condición de terminación (un nodo sea el estado final). La forma de generar el grafo va a ser teniendo en cuenta una heurística (que dependerá del problema concreto a tratar) que ordene los nodos del grafo. De esta forma no se va a olvidar nada de lo explorado y se van a explorar enormes porciones del grafo.

Cuando existe solución, el resultado es un árbol (como parte de un grafo) construido a base de nodos y apuntadores que contiene algún nodo objetivo.

Tómese el ejemplo de un grafo con sus apuntadores correspondientes que aparece en la figura 3.2. En ellas se representan los nodos mediante puntos que están conectados entre si por un conjunto de flechas (arcos). Para cada nodo, salvo el etiquetado con s, se puede observar que existe otra flecha (el apuntador),

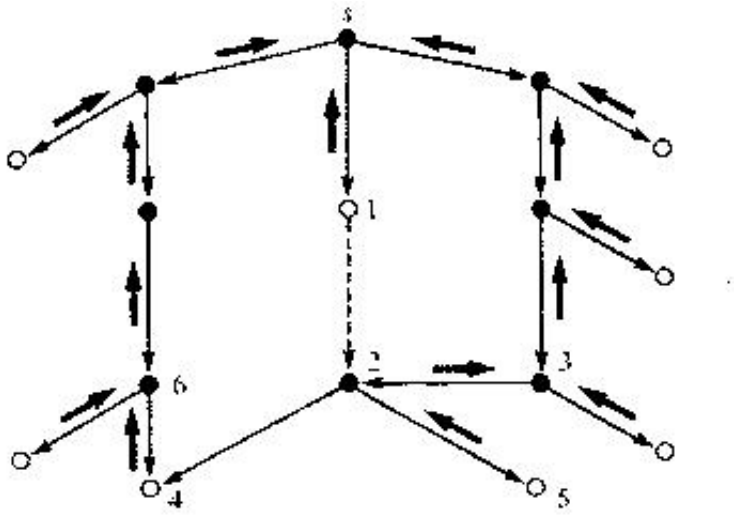


Figura 3.2: Ejemplo de grafo de búsqueda

con trazo más grueso, que lo conecta a uno sólo de sus antecesores.

En los siguientes apartados se van a estudiar dos algoritmos de exploración de grafos como son los algoritmos A* y D*.

3.3. El algoritmo A*

El algoritmo A* es un algoritmo informado de búsqueda en grafos, que usa información del problema para dirigir su búsqueda y no tener que explorar todos los nodos para llegar al objetivo.

Los nodos del grafo de búsqueda son las posibles configuraciones del robot y los arcos nos indican qué configuraciones son accesibles desde la configuración actual y a qué coste. El coste de alcanzar un configuración que produzca la colisión del robot con un obstáculo debe ser infinito.

Este algoritmo se basa en construir un árbol de búsqueda, en el que se van explorando los sucesores de cada nodo, eligiendo los sucesores que vamos a explorar en función del conocimiento que tenemos del problema. El objetivo final es construir una lista de nodos que recorriéndola nos lleve desde el nodo

inicial hasta el nodo final con el coste menor (camino óptimo). Por lo tanto, debemos definir una función que aplicada a cada nodo nos diga la posibilidad de que dicho nodo forme parte del camino óptimo. En función de este valor podremos dirigir nuestra búsqueda.

Esta función es la suma de dos factores:

$f = g + h$

g : coste desde el nodo origen

h : heurística o conocimiento del problema

El término h representa el conocimiento que disponemos del problema (heurística) y basándonos en él vamos a dirigir nuestra búsqueda. En el caso que nos ocupa, el valor de h va a ser igual a la distancia desde el nodo actual al nodo destino. Cuanto menor sea este valor más cerca estaremos del nodo destino. El valor de g es el valor que representa el esfuerzo de llegar al nodo actual. Ese valor va a ser la suma del coste de llegar al padre de dicho nodo más el coste de pasar del nodo padre al nodo actual.

Las propiedades fundamentales del algoritmo A* son:

- A* siempre termina en grafos finitos
- A* es completa en grafos finitos
- A* devuelve la solución óptima
- A* es admisible

El algoritmo A* nos garantiza que si existe un camino posible entre la configuración de origen y la de destino lo va a encontrar, si existen varios nos va a devolver el óptimo y devolverá fallo si no existe.

Dado que el algoritmo A* es admisible, la función heurística debe ser admisible, es decir, debe ser una función que no sobreestime la distancia entre el nodo actual y el nodo destino:

$$DistanciaReal(NodoActual, NodoDestino) \geq h(NodoActual, NodoDestino)$$

Por ejemplo, siempre una ruta real entre dos ciudades es algo mayor y a lo sumo igual a la distancia en línea recta tomada de un mapa. Esta última distancia es una heurística admisible pues en todo caso es "optimista".

3.3.1. Pseudocódigo del algoritmo A*

A ESTRELLA(nodoini,nodofin)

```
1.  insertarAbiertos(nodoini)
2.  while(listaAbiertos != vacia)
3.     nodoactual = sacarAbiertos() /* nodo con menor f */
4.     if (nodoactual==nodofin) return CAMINO ENCONTRADO
5.     listasucesores = generarSucesores(nodoactual)
6.     for (listasucesores)
7.        g(nodosucesor) = g(nodoactual) + costealcanzarnodo
8.        if (enAbiertos(nodosucesor) Y f(nodosucesor)>nodomismaposicion)
9.             continue
10.       if (enCerrados(nodosucesor) Y f(nodosucesor)>nodomismaposicion)
11.            continue
12.       borraAbiertos(nodosucesor)
13.       borraCerrados(nodosucesor)
14.       f(nodoSucesor) = g(nodosucesor) + h(nodosucesor)
15.       insertarAbiertos(nodosucesor)
16.    insertarCerrados(nodoactual)
17. return ERROR
```

En este algoritmo hay dos estructuras de datos muy importantes para su funcionamiento:

- Lista de ABIERTOS, en la que incluimos los nodos que queremos expandir para la búsqueda de la configuración final. En uno de los puntos siguientes trataremos con detenimiento esta estructura, la más importante dentro de este algoritmo.

- Lista de CERRADOS, nodos que ya han sido explorados anteriormente. Los nodos incluidos en esta lista deben haber estado antes en la lista de ABIERTOS.

Para elegir el mejor nodo sucesor del nodo actual para continuar la exploración del mapa en busca del nodo destino (paso 3 del pseudocódigo), utilizamos el valor de f de los nodos incluidos en la lista de abiertos.

3.4. El algoritmo D*

El algoritmo A* supone que el coste asociado a cada arco es constante. Sin embargo, hay veces en las que el coste de atravesar un arco puede variar a lo largo del tiempo o incluso no es conocido. Si utilizamos el algoritmo A* para realizar búsquedas de caminos en grafos de este tipo estamos obligados a replantear al menos parte del camino elegido si ocurre un cambio en el coste de una o varias transiciones entre nodos.

En [Ste94] se presenta el algoritmo D* y se completa su definición en [Ste95], añadiendo la mejora de replantear sólo la parte del camino que sea necesaria en función de la localización actual de la búsqueda en el momento de ocurrir los cambios de coste en los arcos que unen los nodos.

Este algoritmo asegura la consecución del camino óptimo entre dos puntos de la siguiente forma: se genera el camino óptimo en base al coste actual asociado a cada arco. En el caso de cambios en el coste de la transición entre nodos, se replantea el camino óptimo desde la posición actual hasta la posición final.

Al igual que el algoritmo A*, el algoritmo D* utiliza una serie de funciones que van a guiar la búsqueda del camino óptimo entre dos nodos:

- una función $t(X)$ que indica si el nodo X nunca ha estado en la lista de abiertos (NUEVO), si está en ese momento en la lista de abiertos (ABIERTO) o si ya ha salido de la lista de abiertos (CERRADO).
- una función $h(G, X)$ que indica el coste de alcanzar el nodo final G desde

el nodo actual X.

- una función $k(G, X)$ que indica el menor valor de h para un determinado nodo X desde que entró en la lista de abiertos.

Una variación en el coste de un arco no sólo afecta a ese arco, sino que va a afectar a otros muchos arcos, ya que lo que está cambiando es el coste final de todos los caminos que pasan a través de ese arco. Para transmitir estos cambios podemos clasificar los nodos en dos grupos, en función del valor de la función h para ese nodo:

- RAISE, nodos en los cuales *h(nodo)>k(nodo)*. A través de este tipo de nodos el algoritmo transmite los incrementos de coste del camino. Estos nodos se encuentran fuera del camino óptimo, ya que el valor de h para ese nodo supera el valor mínimo actual (k).
- LOWER, nodos en los que *h(nodo)=k(nodo)*. Estos nodos transmiten el decremento de coste del camino. Estos nodos van a formar parte del camino óptimo, ya que el valor de h coincide con el valor de h mínimo en ese momento (k).

La propagación de esta información se produce cuando se eliminan nodos de la lista de abiertos. Cada vez que se elimina un nodo de abiertos se expande en busca del camino hasta el nodo inicial, pasando el coste del nodo a sus vecinos. Los nuevos nodos son incluidos en la lista de abiertos para proseguir con la búsqueda.

- Cuando se incrementa el coste de atravesar un arco, el nodo afectado por ese cambio se incluye en la lista de abiertos y el incremento de coste se transmite mediante estados RAISE a lo largo de todos los caminos que contienen dicho arco. Cuando ese estado RAISE entra en contacto con nodos vecinos con un coste menor que el suyo, esos estados LOWER son insertados también en la lista de abiertos. A través de estos nodos se va a decrementar el coste de los nodos que anteriormente habían sufrido un aumento del coste siempre que sea posible.

- Si el coste de pasar de un nodo a otro disminuye, esa reducción se propaga mediante nodos de tipo LOWER a través de todas las secuencias de nodos que incluyan el arco, siempre que el coste de dichos nodos se pueda decrementar. De esta forma redirigimos la búsqueda.

3.4.1. Pseudocódigo del algoritmo D*

El pseudocódigo que vamos a comentar es el de la primera versión (ambos pseudocódigos son muy parecidos) y es el siguiente:

D ESTRELLA(nodoini,nodofin)

```
1.  insertarAbiertos(nodoini)
2.  while(estado(nodofin) != CERRADO)
3.      X=nodoMejor()
4.      if(X==NULL) return NO ENCONTRADO
5.      k_old=getKMIN()
6.      borra(X)
7.      if(k_old<h(X)) /* Estado RAISE */
8.        Por cada vecino Y de X
9.          if(h(Y)≤ k_old  y h(X)>h(Y)+c(Y,X))
10.           b(X)=Y
11.           h(X)=h(Y)+c(Y,X)
12.     else if(k_old==h(X)) /* Estado LOWER */
13.       Por cada vecino Y de X
14.         if((t(Y)==NUEVO) o (b(Y)==X y h(Y)≠h(X)+c(X,Y)) o
15.               (b(Y)≠X y h(Y)>h(X)+c(X,Y)))
16.           b(Y)=X
17.           insertar(Y,h(X)+c(X,Y))
18.     else
19.       Por cada vecino Y de X
20.         if((t(Y)==NUEVO) o (b(Y)==X y h(Y)≠h(X)+c(X,Y)))
21.           b(Y)=X
22.           insertar(Y,h(X)+c(X,Y))
```

```
23.          else
24.             if(b(Y)≠X y h(Y)>h(X)+c(X,Y))
25.                insertar(X,h(X))
26.             else if((b(Y)≠X) y (h(X)>h(Y)+c(X,Y)) y t(Y)==CERRADO
27.                    y h(Y)>k_old)
28.                insertar(Y,h(Y))
29.     return getKMIN()
```

Esta función comienza sacando de la lista de abiertos el nodo con menor valor de k. El resto de la función se encuentra dividida en tres grandes partes en función del valor de h para el nodo elegido comparado con k_{old}.

- Si el nodo extraído de la lista de abiertos es un estado LOWER ($k(X) = h(X)$) (líneas de la 10 a la 15 del pseudocódigo) el coste del camino que pasa por ese nodo es óptimo siempre que $h(X)$ sea igual que k_{old}. Dentro de esta condición if se examinan los nodos vecinos al nodo elegido para ver si se puede reducir el coste del camino. Además se asigna valor al coste del camino de los nodos nuevos. Los cambios en el coste del camino son propagados a los nodos vecinos que pertenezcan a caminos hacia el nodo de destino que pasen por dichos nodos y el nodo elegido en las primeras líneas del algoritmo. En los casos que sea necesario, los apuntadores que se usan para construir el camino se redirigen. Todos los nodos a los que se le asigna un nuevo coste son colocados en la lista de abiertos con la función `insertar`. De esta forma propagan dichos cambios de coste a sus vecinos.

- Si el nodo extraído de la lista de abiertos es un nodo de tipo RAISE (líneas de la 6 a la 9 del pseudocódigo) el coste de alcanzar el nodo final no es el óptimo. Antes de que X propague los cambios en el coste del camino, sus vecinos óptimos son examinados ($k_{old} < h(X)$) con el objetivo de intentar reducir su valor de h.

- En la última parte del if (líneas de la 16 a la 26 del pseudocódigo) que divide la función en tres, los cambios en el coste del camino son propagados a los estados nuevos y a los descendientes directos en el camino, de la misma forma que para los nodos de tipo LOWER. Si X puede permitir

disminuir el coste de un nodo que no es descendiente directo, se inserta nuevamente en la lista de abiertos para volverlo a expandir más adelante. Por último, si el coste de X se puede reducir mediante un hijo subóptimo, este hijo es insertado en la lista de abiertos. En este caso la actualización del coste se pospone hasta que el hijo tenga un coste óptimo.

3.5. La lista de abiertos

Uno de los pasos más críticos en los dos algoritmos de búsqueda que hemos presentado es la búsqueda del nodo con menor f dentro de la lista de abiertos. En una primera solución podríamos insertar los nodos en la lista sin ningún control. En este caso, el tiempo de inserción en la lista es mínimo, pero debemos recorrer la lista entera para asegurarnos de que elegimos el nodo con menor f. En función de la longitud de la lista de abiertos esta búsqueda puede ser muy lenta. Esto puede solucionarse si ordenamos los nodos en función de la f al incluirlos en la lista de abiertos.

3.5.1. Lista ordenada

Si utilizamos una lista ordenada para almacenar la lista de abiertos, se consume tiempo de cómputo en la inserción de nodos, pero disminuye el tiempo de cómputo a la hora de sacar el nodo con menor f, pues ya no se necesita recorrer la lista completa, basta con sacar el primer nodo de la lista ordenada, pues siempre va a ser el nodo con menor f.

A simple vista puede parecer que simplemente hemos desplazado el coste en tiempo desde la extracción a la inserción de los nodos en la lista. Sin embargo, esto no es así. En el peor de los casos el tiempo necesario para insertar de forma ordenada va a coincidir con el de la búsqueda del nodo con menor f en el caso de la lista no ordenada. No sólo eso, sino que normalmente va a ser bastante menor. Si nos fijamos en el funcionamiento de los algoritmos A* y D*, siempre vamos a elegir, de entre todos los nodos vecinos al nodo actual, el que menor f tenga. Ese nodo pasará a ser el nodo actual, exploraremos sus nodos vecinos

y elegiremos el de menor f, etc. A menos que existan grandes obstáculos, los nodos que vamos incluyendo tendrán valores de f cada vez más pequeños, por lo que habrá que insertarlos en los primeros puestos de la lista de abiertos. Esto hace que el tiempo de inserción en la mayoría de los casos sea también bastante pequeño.

Los tiempos de inserción pueden empezar a crecer si existen obstáculos grandes en el camino, ya que el coste g de los nodos empieza a crecer (y por tanto el valor de f para dichos nodos también), y esto hace que empiecen a pasar a zonas más alejadas del inicio de la lista. Una solución sencilla a este problema es calcular el valor medio de f de todos los nodos de la lista. A la hora de insertar un nodo, si su valor de f es menor que la media se comienza a buscar por el principio de la lista, y si su valor supera la f media se empieza a buscar por el final de la lista.

Otra solución es usar el algoritmo de ordenación *QuickSort*. El funcionamiento de este algoritmo es relativamente sencillo: se compara la f del nodo a insertar con la f del nodo que se encuentre en el medio de la lista. Si la f es menor se decartan los elementos de la lista que se dejan a la derecha y se repite el proceso con la mitad de la lista que hemos dejado a la izquierda, y si es mayor se descartan los elementos de la lista que se dejan a la izquierda y se repite el proceso con la mitad derecha de la lista. Esto se repite de manera recursiva hasta que las listas de la izquierda y de la derecha no contengan elementos, se compara y en función del resultado el nodo que queríamos insertar se coloca en la lista.

INSERTARQS (elemento,lista)

```
1.  if (tamanolista==0)
2.      lista[actual]=elemento
3.      return
4.  if(felemento<lista[mitadlista])
5.      INSERTARQS(elemento,mitadlistaizq)
6.  if(felemento>lista[mitadlista])
7.      INSERTARQS(elemento,mitadlistader)
```

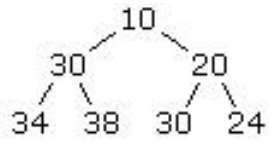


Figura 3.3: Ejemplo de estructura montículo binario

3.5.2. Montículo binario

Un método muy similar al Quick Sort consiste en usar la estructura de datos montículo binario. Este tipo de estructura de datos se basa en un detalle que hasta ahora no se había tenido en cuenta: para la lista de abiertos no se necesita que toda la lista esté ordenada, basta con que el primer nodo de la lista tenga el menor valor de f. El resto de la lista puede estar completamente desordenada.

En una estructura montículo binario, el nodo con menor f está en la parte alta del montículo. Ese nodo tendrá dos hijos con un valor de f mayor o igual que el nodo padre. Estos a su vez tendrán dos hijos con mayor o igual valor de f, y así sucesivamente. Un ejemplo aparece en la figura 3.3.

Como podemos ver en el ejemplo, en la parte más alta se encuentra el nodo con menor f. El resto de la estructura no está ordenada. Lo importante es que a la hora de sacar de la estructura ese primer nodo se reorganice el resto de tal manera que en la parte más alta se encuentre el nodo con menor f de los nodos restantes.

Otra de las grandes ventajas de esta estructura de datos es que se puede almacenar en una matriz unidimensional, con lo que será más fácil trabajar con ella. El elemento que se encuentra en lo alto del montículo se sitúa en la primera posición de la matriz (3.4), sus dos hijos en las dos posiciones siguientes, sus cuatro hijos en las cuatro posiciones siguientes, etc.

A continuación se describirán los algoritmos para los hijos de un nodo, para insertar un nodo y para eliminarlo.

- Para localizar los hijos de un nodo determinado basta con multiplicar la posición del nodo dentro de la matriz por dos. En esa posición y en la

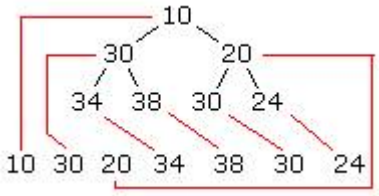


Figura 3.4: Estructura montículo binario en forma matricial

```
10 30 20 34 38 30 24 17
10 30 20 17 38 30 24 34
10 17 20 30 38 30 24 34
```

Figura 3.5: Inserción en una estructura montículo binario en forma matricial

siguiente encontraremos los hijos del nodo.

- Para insertar un nodo en la matriz se debe colocar al final. Después se compara con su padre, situado en la posición $\frac{nodosenlista}{2}$, y si $f(nodopadre) > f(nodohijo)$ entonces se intercambian. Este proceso se repetirá hasta que nos encontremos en una posición en la que $f(nodopadre) < f(nodohijo)$. Si se repite el proceso sobre el ejemplo previo se obtiene el resultado de la figura 3.5.

- Para eliminar un elemento (en el caso que nos ocupa siempre será el primero) el mecanismo es muy similar salvo que la ordenación se realiza al revés. Cuando se elimina el primer elemento se mueve el último hasta la primera posición. Después se compara el valor de la f del que ahora es el primer nodo con la de sus dos hijos. En el caso de que el padre tenga la menor f, se finaliza. En caso contrario el nodo padre intercambia su posición con el nodo que tenga menor f. En la figura 3.6 se muestra el resultado de aplicar el proceso descrito para eliminar el nodo con valor de $f = 10$ que se encontraba en la cima del montículo.

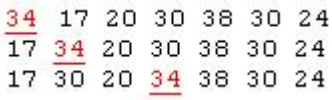


Figura 3.6: Eliminación en una estructura montículo binario en forma matricial

3.6. Revisión de trabajos que utilizan algoritmos de búsqueda en planificación

Existen multitud de trabajos que utilizan distintos algoritmos de búsqueda para la planificación de caminos. En este apartado vamos a hacer un repaso tanto de trabajos sobre el tema como de los distintos algoritmos que pueden usarse para la búsqueda.

En [Tro96], Karen I. Trovato realiza un estudio de los algoritmos A* y D* en el caso de un brazo robótico y un automóvil. En este trabajo se realiza un estudio del coste de los algoritmos de manera teórica y se plantean distintos criterios mediante los cuales guiar la búsqueda del camino. En el caso del robot planar articulado los criterios planteados son:

- la distancia euclidea entre dos configuraciones. El criterio de la distancia en el espacio de las configuraciones será el criterio que se utilizará en este trabajo.
- minimizar el movimiento del extremo móvil del robot.
- minimizar la potencia necesaria para mover el robot desde la configuración inicial hasta la final. Las masas de los dos elementos del robot se suponen distintas.
- en base al tiempo empleado en mover el robot desde la la configuración inicial hasta la final. Se supone que uno de los elementos del robot es más lento que el otro.

En [GL02] se utiliza el algoritmo A* sobre un espacio de búsqueda representado utilizando árboles BSP (Binary Space Partitioning). En este trabajo, la construcción del mapa de búsqueda mediante un árbol BSP es la parte de

la planificación que lleva la mayor parte del tiempo de cómputo, ya que es un tipo mapa que contiene mucha información. Este tipo de representación del entorno depende de la geometría de los obstáculos. Esto hace que esta técnica de planificación sea muy difícil de usar en entornos de más de tres dimensiones.

En [AB04] se utiliza el algoritmo A* sobre mapas representados de manera jerárquica, de forma que la búsqueda del camino se realiza en varios niveles. Este método es muy útil cuando las dimensiones del espacio de búsqueda son muy grandes. El método que se propone consiste en dividir el mapa inicial en distintos submapas más pequeños, que a su vez pueden ser divididos en otros más pequeños, etc. Una vez dividido el mapa en submapas de menor tamaño, se realiza la búsqueda del camino óptimo de forma individual sobre cada uno de los mapas, y los caminos obtenidos se unen para obtener un único camino. El rendimiento es hasta 10 veces mayor que si la búsqueda se realizara directamente sobre el mapa original. Este método tiene como desventaja que no se obtiene un camino total óptimo, ya que no basta con que los subcaminos que lo forman sí lo sean.

En [AAM93] Maciejewski propone una planificación de caminos libres de obstáculos basado en las características geométricas de los obstáculos en el espacio de las configuraciones. Mediante este método se consigue un buen rendimiento, pero es un método muy difícil de aplicar y de un muy bajo rendimiento en entornos de más de tres dimensiones, lo que limita el método a robots articulados de dos o tres grados de libertad, ya que las dimensiones del espacio de las configuraciones están directamente relacionadas con los grados de libertad del robot articulado. Un aspecto a destacar de este trabajo es el estudio que se realiza del espacio de las configuraciones y su relación con el espacio de trabajo, que utilizaremos en este trabajo en varios apartados como base de la explicación del método de planificación propuesto.

También son bastante comunes los trabajos que realizan la planificación mediante redes neuronales, como en [PK98] y [YM00]; o algoritmos genéticos como en [NA97]. Estos algoritmos son muy útiles en el caso de trabajar con espacios de configuraciones con muchos mínimos locales.

Otros trabajos utilizan modelos de fluidos para la planificación de caminos, como [LB98] y [LL00]. Con este tipo de técnicas podemos utilizar mapas con distintos tipos de espacio libre para poder simular distintos rendimientos del robot en cada uno de ellos.

4

Técnica de planificación de caminos propuesta

En este apartado se detalla como se han aplicado las técnicas comentadas en los apartados anteriores en el caso de un robot planar articulado.

Comenzaremos explicando como se genera el espacio de las configuraciones, ya que sobre este mapa se va a realizar el cálculo de caminos.

Tras la generación del mapa de búsqueda se pasará comentar los pasos necesarios para la búsqueda de caminos óptimos mediante los algoritmos A* y D* en el caso concreto que nos ocupa en este trabajo.

4.1. Descripción del problema

El problema de la búsqueda de caminos en robots articulados puede definirse como encontrar un movimiento continuo del robot que le lleve desde la configuración inicial hasta la configuración final sin colisionar con los obstáculos del entorno.

El caso particular sobre el cual vamos a probar el método de planificación propuesto es el de un robot articulado con dos grados de libertad como el de la figura 4.1. El robot se mueve en un espacio de trabajo bidimensinal, por lo que el problema consiste en mover el extremo del robot de un punto a otro del espacio de trabajo.

El primer paso de la planificación propuesta consiste en trasladar el problema al espacio de las configuraciones, pues el cálculo del camino libre de colisiones se simplifica porque únicamente es necesario planificar los movimientos de un punto y no de los dos elementos que constituyen el robot. En el apartado 4.3 se detallará cómo se proyectan los obstáculos en el espacio de las configuraciones.

Como segundo paso se utilizará un algoritmo de búsqueda para buscar el camino óptimo entre la configuración inicial y la final. Para ello utilizaremos los algoritmos de búsqueda A* y D* adaptándolos a este caso particular. Esto se trata con detalle en el apartado 4.4.

Los algoritmos propuestos utilizan información del problema (heurística) para dirigir la búsqueda del camino óptimo libre de obstáculos. Vamos a utilizar campos de potenciales para guiar dicha búsqueda. La generación del campo de potenciales en el espacio de las configuraciones se detalla en el último apartado.

4.2. Cinemática de un robot planar articulado

Para poder calcular las dos posibles configuraciones que puede adoptar el robot para alcanzar el punto de destino en el espacio de trabajo debemos utilizar cinemática inversa. Si utilizamos la notación que aparece en la figura 4.1, θ_1 y

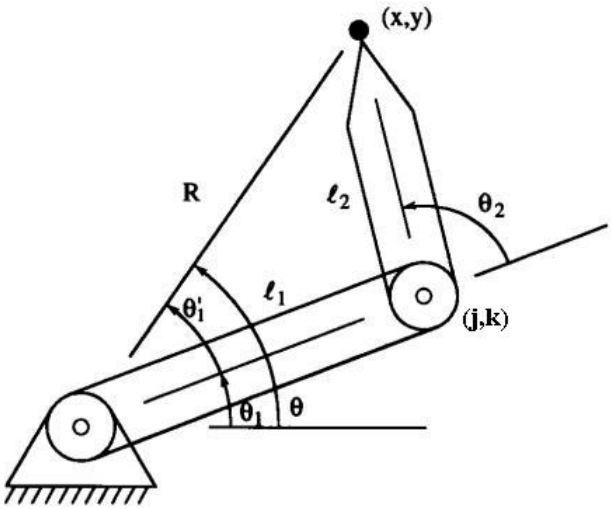


Figura 4.1: Cinemática para un brazo articulado con dos grados de libertad

θ_2 vendrán dados por:

$\theta_1 = \theta + \theta_1' = \theta \pm cos^{-1} \frac{R^2+l_1^2-l_2^2}{2l_1R}$

$\theta_2 = \pm cos^{-1} \frac{R^2-l_1^2-l_2^2}{2l_1l_2}$

donde

$R = \sqrt{x^2+y^2}$

$\theta = tan^{-1} \frac{x}{y}$

y l_1 y l_2 son las longitudes de los elementos 1 y 2 del robot, respectivamente.

Utilizando estas expresiones podemos calcular las dos posibles configuraciones codo arriba y codo abajo (signo $\pm$) que puede adoptar el robot para alcanzar el punto de destino. Dado que en este trabajo nos interesa el camino óptimo y que a priori no podemos asegurar cuál de los dos posibles configuraciones conduce al camino más corto, debemos calcular los dos caminos posibles y compararlos. Esto hará que el cálculo del camino sea más lento pero aseguramos que el camino elegido es el más corto de los dos. En los apartados siguientes se describe cómo se realiza la búsqueda del camino entre dichas configuraciones.

Una vez obtenido el camino, dispondremos de una serie de configuraciones

para el robot, de forma que si va pasando de una a otra en orden realizará un movimiento que llevará el extremo del robot desde el punto origen al punto destino del espacio de trabajo sin colisionar con los obstáculos que le rodean.

Si se quiere obtener la posición del robot en el espacio de trabajo para cada una de las configuraciones que forman el camino (en la base del robot se sitúa el sistema de referencia fijo), por lo que los cálculos a realizar, basándonos en la notación empleada en la figura 4.1 son:

$$j = l_1 cos\theta_1 \qquad x = l_2 cos\theta_2$$

$$k = l_1 sen\theta_1 \qquad y = l_2 sen\theta_2$$

4.3. Cálculo del espacio de las configuraciones

El cálculo del espacio de las configuraciones fue planteado de forma matemática por Maciejewski en [AAM93]. En la figura 4.2 podemos ver la transformación entre un obstáculo puntual en el espacio de trabajo 4.2(b) y el mismo obstáculo en el espacio de las configuraciones 4.2(a). La relación entre las dos partes del robot es $l_2 = l_1$.

En base a la geometría de los obstáculos en el espacio de las configuraciones podemos dividirlos en dos: obstáculos del A al E y obstáculos del F al I. El primer grupo de obstáculos podemos distinguir dos partes: una primera línea punteada que representa la posible colisión del primer elemento del robot con el obstáculo y una segunda línea no punteada que representa la colisión del segundo elemento del robot con el obstáculo. Este primer grupo de obstáculos tienen la peculiaridad de cumplir que $R_{obst} \leq l_1$. En el segundo grupo de obstáculos, sólo el segundo elemento del robot puede colisionar con algún obstáculo. La forma de estos obstáculos en el espacio de las configuraciones es la de una S invertida que se hace más pronunciada a medida que el radio R_{obst} se aproxima a l_1. En el obstáculo E se produce la transición entre los dos grupos, ya que en ese obstáculo $R_{obst} = l_1$.

En la figura 4.3 podemos ver la tranformación entre líneas en el espacio de

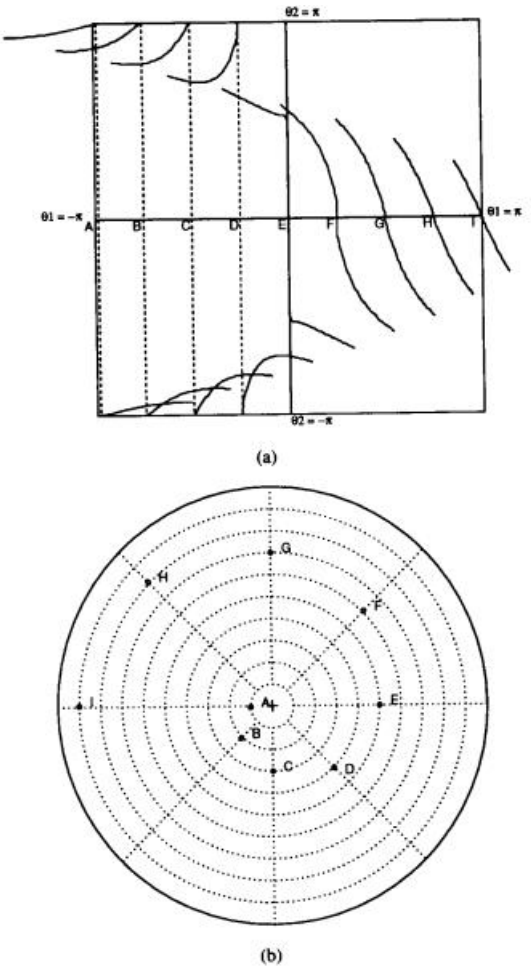


Figura 4.2: Transformación de un obstáculo desde el espacio de trabajo y al espacio de las configuraciones.

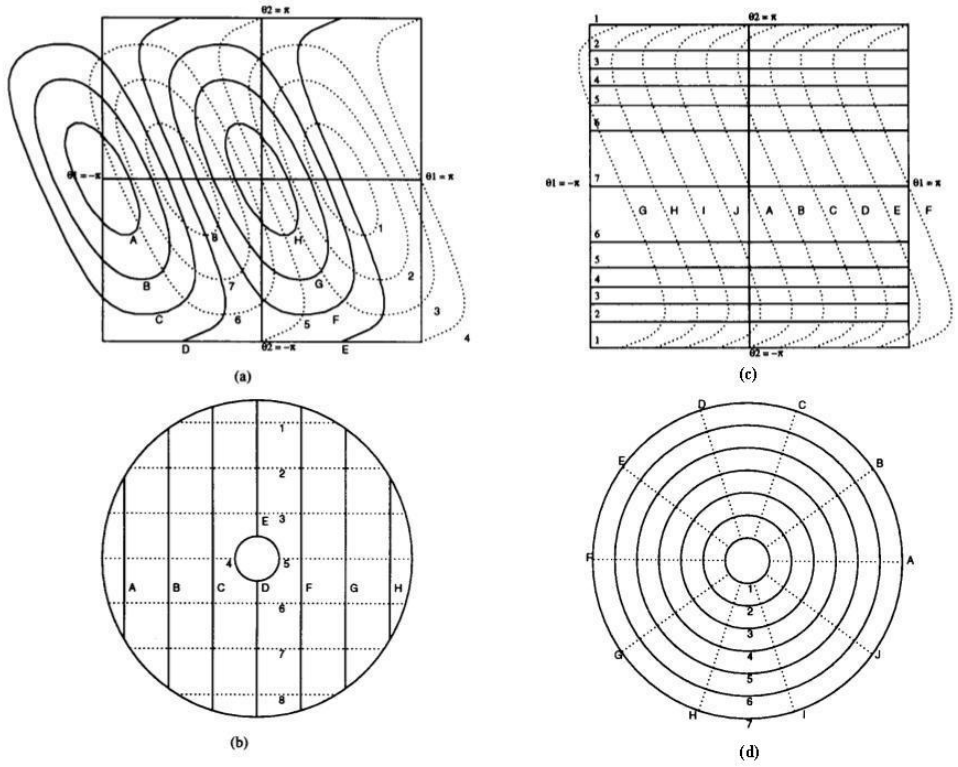
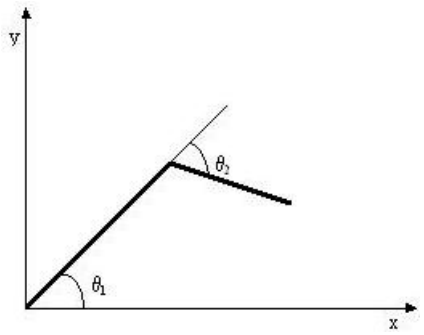


Figura 4.3: Transformación entre el espacio de trabajo y el espacio de las configuraciones.

trabajo y las curvas asociadas a ellas en el espacio de las configuraciones. En 4.3(a) podemos ver el espacio de las configuraciones correspondiente al espacio de trabajo 4.3(b) expresado en coordenadas cartesianas y en 4.3(c) el espacio de las configuraciones correspondiente al espacio de trabajo 4.3(d) expresado en coordenadas polares. La relación entre las dos partes del robot es $l_2 = \frac{3}{4}l_1$.

Para describir una configuración arbitraría de un robot planar articulado se necesita conocer el valor de los ángulos girados por las articulaciones θ_1 y θ_2. Por lo tanto, una configuración vendrá determinada por la tupla de dimensión dos (θ_1,θ_2). Así, el espacio de las configuraciones estará formado por todos los pares de coordenadas (θ_1,θ_2) donde θ_1 y θ_2 son números reales dentro del intervalo [0,2π).



El cálculo del espacio de las configuraciones se va a realizar utilizando el algoritmo planteado en [BC97] para un robot planar articulado. Este algoritmo optimiza el tiempo de cálculo empleado, lo que nos va a permitir generar de manera muy rápida el C-espacio cada vez que se produzca un cambio en el espacio de trabajo.

Este método se basa en el hecho de que la representación de los obstáculos en el espacio de las configuraciones puede verse como la convolución de dos funciones que describan al robot y a los obstáculos. En el caso que nos ocupa la representación se realiza en un sistema de coordenadas polares (r, φ) y las expresiones finales para el cálculo del Cespacio presentadas en [BC97] son:

$$F[CB_1] = \int (F[A_{1(0)}(r, \theta_1)]_\varphi F[B(r, \theta_1)]_\varphi) dr$$

$$F[CB_2] = \int (F[A_{2(0,\theta_2)}(r, \theta_1)]_\varphi F[B(r, \theta_1)]_\varphi) dr$$

donde A_1 y A_2 con los subconjuntos del espacio de trabajo que representan los dos elementos que forman el robot y B es el subconjunto del espacio de trabajo ocupado por los obstáculos.

Este método tiene dos ventajas respecto al utilizado en [AAM93]:

- Permite trabajar con obstáculos no puntuales, tengan la forma que tengan.
- Permite trabajar sobre representaciones en forma de bitmaps del espacio de trabajo.

4.4. Búsqueda del camino óptimo

Una vez generado el espacio de las configuraciones, debemos usar la información contenida en él para generar un camino libre de colisiones entre las configuraciones inicial y final. Para realizar este último paso vamos a utilizar los algoritmos de búsqueda A* y D*, que se detallaron en 3.3 y 3.4.

Para usar estos algoritmos en el caso concreto que nos ocupa, podemos ver nuestro espacio de configuraciones como un grafo en el que las posibles configuraciones de nuestro robot son los nodos que forman el grafo. Dentro de ese grafo habrá nodos (configuraciones) que no podremos alcanzar debido a la existencia de obstáculos. Nuestro objetivo es explorar este grafo para conseguir una sucesión de configuraciones que nos lleve de la configuración inicial a la final esquivando las configuraciones prohibidas.

La definición formal de una ruta libre de colisiones dada en 2.1 se ha realizado para espacios de configuraciones euclideos. Pero en este trabajo se planificará sobre un espacio discretizado en forma de mapa de bits (bitmap), que será la estructura de datos que representa el grafo sobre el cual efectuaremos la búsqueda.

Cuando un robot se encuentra en una configuración determinada, las posibles configuraciones que se pueden alcanzar desde ella son conocidas como configuraciones *vecinas*. Por lo tanto, los vecinos de un nodo son aquellos que se pueden alcanzar desde un nodo dado con un desplazamiento atómico. Desplazamiento atómico es el desplazamiento más pequeño con el que pasamos de una configuración a otra. Este desplazamiento lo marca el nivel de discretización que usemos para representar el entorno y los movimientos de nuestro robot.

Para esquivar las configuraciones prohibidas, debemos hacer que el algoritmo de búsqueda no elija esas configuraciones como posibles componentes del camino elegido como solución, o lo que es lo mismo, debemos impedir que estos nodos se incluyan en el conjunto de abiertos. Para ello debemos asignarles un coste infinito, lo que hará que valga lo que valga la función heurística para ese nodo no se elija nunca ese nodo como posible sucesor.

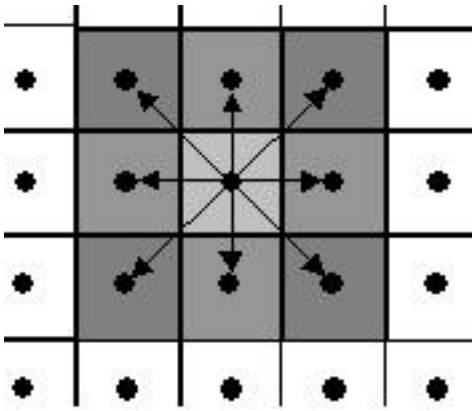


Figura 4.4: Configuraciones vecinas

La función f que dirige la búsqueda en el algoritmo A* es la suma de otras dos funciones: la función de coste g y la función heurística h. Como ya vimos en el apartado 3.3, la función de coste representa el esfuerzo que debe realizar el robot para pasar de una configuración a otra. Lógicamente, el esfuerzo para atravesar un obstáculo debe ser infinito. La función heurística en este caso va a ser la distancia al nodo destino desde el nodo actual.

Además, debemos tener en cuenta el campo de potenciales artificiales que hemos generado para alejar al robot de los obstáculos penalizando las configuraciones afectadas por dicho campo.

Para poder tener en cuenta en la función f todos los factores comentados debemos realizar un pequeña modificación sobre dicha función que propone el algoritmo:

$f = g + h + z$

g : coste desde el nodo origen

h : distancia al nodo destino (heurística)

z : valor del campo de repulsión para esa configuración

El valor de z dependerá de la proximidad de esa configuración a un obstáculo. En el caso de las configuraciones no afectadas por un campo de repulsión de un obstáculo el valor de z será 0. Este valor se asignará a cada configuración en función de cómo se genere el campo de potenciales. La generación del campo de potenciales se detalla en 4.5.

El coste g de pasar de una configuración a otra será constante y de valor 1, ya que el espacio libre se supone homogeneo, por lo que el pasar de una configuración a otra del espacio no ocupado por los obstáculos tendrá asociado el mismo coste se encuentren donde se encuentren dichas configuraciones.

El valor de h es la distancia al nodo destino. Una vez que se dispone del espacio de las configuraciones toda la información está incluida en él, por lo que la función distancia sólo dependerá del espacio de las configuraciones. Así, dadas las configuraciones p y q con coordenadas (x, y) y (s, t) respectivamente (consideradas como posiciones en el bitmap), en un espacio de las configuraciones discreto la definición de distancia puede ser definida de varias formas. En base al número de vecinos que se tienen en cuenta:

1. 4-distancia: los vecinos a distancia 1 con esta definición serán los 4-vecinos, y las distancias a una configuración q quedarán:

$$\begin{array}{ccccc} & & 2 & & \\ & 2 & 1 & 2 & \\ 2 & 1 & q & 1 & 2 \\ & 2 & 1 & 2 & \\ & & 2 & & \end{array}$$

2. 8-distancia: Los vecinos a distancia 1 con esta definición se llamarán los 8-vecinos, y las distancias a una determinada configuración q quedarán:

$$\begin{array}{ccccc} 2 & 2 & 2 & 2 & 2 \\ 2 & 1 & 1 & 1 & 2 \\ 2 & 1 & q & 1 & 2 \\ 2 & 1 & 1 & 1 & 2 \\ 2 & 2 & 2 & 2 & 2 \end{array}$$

En base a la manera de calcular el valor de la función distancia:

1. Distancia euclídea: $d(p, q) = \sqrt{[(x - s)^2 + (y - t)^2]}$.

2. Distancia Manhattan: $d(p, q) = |x - s| + |y - t|$.

En este trabajo se implementarán tanto la distancia euclidea como la distancia manhattan, pero sólo se utilizará la 8-distancia, al ser la más completa de las dos posibilidades y no existir ninguna restricción al respecto en el caso que nos ocupa.

En el caso del algoritmo D* necesitamos una función h que nos indique el coste de alcanzar el nodo final desde el nodo actual. Para poder aprovechar las funciones definidas para el algoritmo A*, basta con comenzar la búsqueda desde el nodo final hacia el nodo inicial y utilizar la función g definida anteriormente. De esta forma disponemos del coste desde el nodo actual hasta el nodo final. Si además deseamos incluir los potenciales de repulsión de los obstáculos la función queda como sigue:

$f = g + z$

g : *coste hasta el nodo final*

z : *valor del campo de repulsión para esa configuración*

Es hecho de utilizar funciones comunes nos permitirá una codificación de los algoritmos más rápida y sencilla.

Otra modificación importante va a ser la eliminación de la lista de cerrados en el algoritmo A*, ya que lo único para lo que vamos a utilizar la lista de cerrados es para comprobar si el nodo actual ya ha sido visitado. Esta estructura de datos será sustituida por la función t definida para el algoritmo D* que nos indicará si el nodo actual está sin visitar, abierto o cerrado.

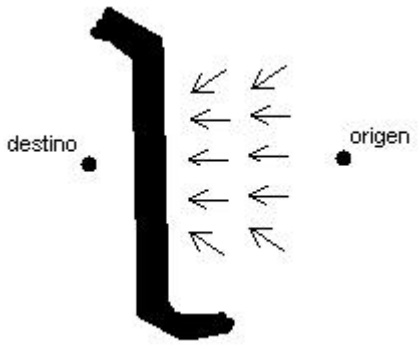
4.5. Cálculo del campo de potenciales

Una vez se ha calculado el espacio de las configuraciones, debemos generar el campo de potenciales que repelerá al robot de los obstáculos y lo atraerá hacia la configuración de destino.

Respecto a la parte del campo de potenciales que atrae al robot hacia la configuración de destino debemos utilizar alguna función capaz de indicar para un determinado nodo hacia que dirección es atraido el robot. El problema es

encontrar dicha función teniendo en cuenta que los algoritmos a desarrollar deben ser lo más generales posibles para ser usados en futuros trabajos, en distintas situaciones y con mapas de distintas dimensiones.

La función elegida va a ser la distancia al nodo destino. Es una función sencilla, rápida de calcular y válida para cualquier dimensión. Sin embargo, aporta poca información sobre el entorno, lo que puede provocar situaciones en las que el campo de potenciales nos atraiga hacia los obstáculos:



La función distancia elegida será utilizada por el algoritmo de búsqueda para guiar la búsqueda a la configuración destino.

Por otro lado, para generar el campo de repulsión que va a alejar al robot de los obstáculos, debemos marcar de alguna forma las configuraciones próximas a los obstáculos de forma que esa información pueda ser usada por los algoritmos que van a buscar la configuración de destino para alejarse de los obstáculos.

En este trabajo vamos a generar un campo de repulsión que pierda fuerza a medida que nos alejamos de los obstáculos. Para generar ese campo de repulsión, debemos marcar las configuraciones que rodean al obstáculo con el campo de repulsión más potente, las configuraciones que rodean a las configuraciones que hemos marcado anteriormente con un campo de repulsión una unidad menos potente, etc. El proceso se repetirá en base al alcance que se quiera dar al campo de repulsión. Un ejemplo del resultado esperado lo podemos ver en la figura 4.5. Un pseudocódigo para hacer esto puede ser el siguiente:

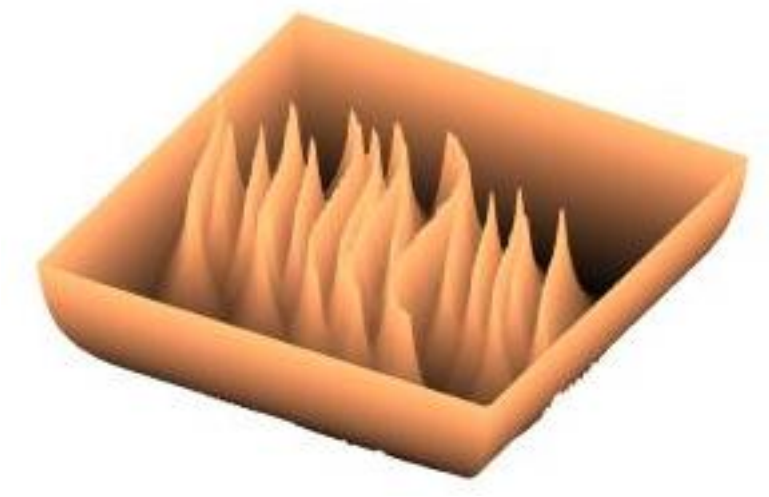


Figura 4.5: Campo de potenciales para robot planar articulado

```
1.  Desde i=1 hasta maxCampoRepulsion
2.     Para todos los puntos que rodeen al obstáculo a una distancia i
3.       Si el nodo esta dentro del mapa
4.         Si el nodo no es un obstáculo
5.           Si la fuerza de repulsión nueva es mayor que la existente
6.             nodoActual.pot = maxCampoRepulsion - i + 1
```

De esta forma añadimos más información sobre el entorno, indicando de esta forma la proximidad de un obstáculo y reduciendo el efecto del problema de atracción hacia los obstáculos anteriormente comentado.

Si pese a todo, la información que generamos del entorno nos dirige a un obstáculo, los algoritmos de búsqueda deberán ser capaces de redonducir la búsqueda en un tiempo razonable.

El objetivo, por tanto, es generar un mapa del entorno del robot en el mínimo tiempo posible, recayendo sobre el algoritmo de búsqueda elegido la tarea de reconducir la búsqueda en el caso de que la representación dada no nos ofrezca la información necesaria para esquivar un obstáculo.

5

Aplicación desarrollada

En este apartado se va a presentar la aplicación desarrollada para estudiar la técnica de planificación propuesta y estudiar los resultados obtenidos. En el primer apartado se muestran los diagramas de clases y de secuencia más importantes de la aplicación. En el segundo se comentan los detalles de implementación. En los distintos puntos de este apartado se comenta como se han solucionado distintos problemas encontrados a la hora de implementar la aplicación. En el tercero y último se presenta la interfaz de la aplicación y se explica como manejar e interpretar los resultados que se obtienen con la herramienta.

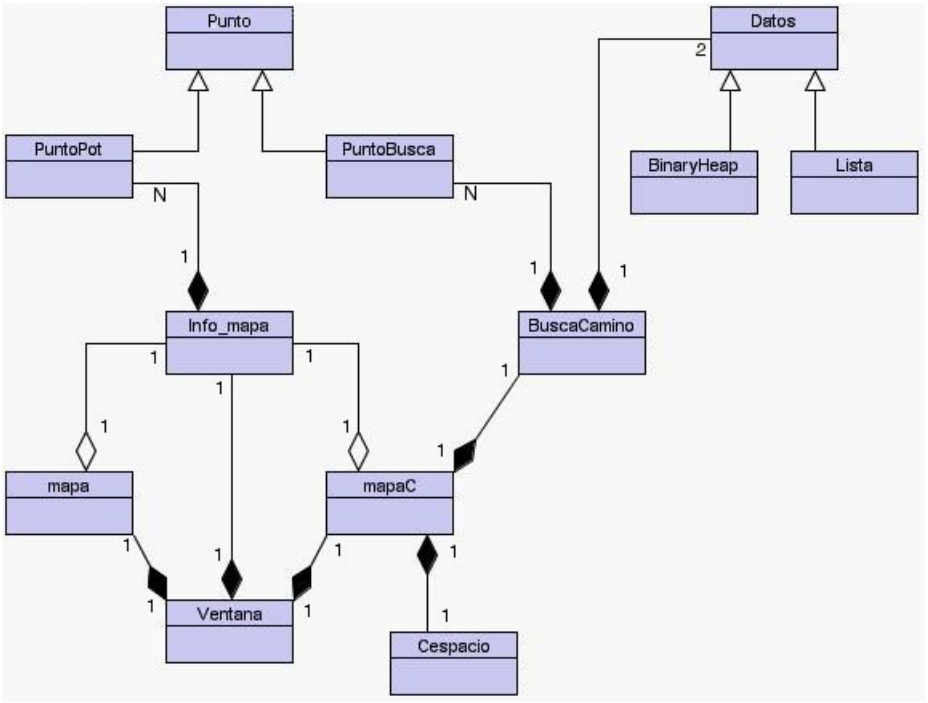


Figura 5.1: Diagrama de clases de la aplicación

5.1. Diseño de la aplicación

En este apartado se describen los distintos componentes y herramientas utilizados para el desarrollo de la aplicación, explicar la estructura interna de la aplicación, los objetos que la forman y la comunicación entre ellos.

5.1.1. Diagrama de clases

En este apartado vamos a comentar las clases que se han creado para este trabajo y la relación que existe entre ellas. En la figura 5.1 podemos ver el diagrama de clases de la aplicación, mediante el cual podemos ver la aplicación de forma estática: las clases que la forman y las relaciones existentes entre ellas.

Para almacenar la información correspondiente a cada uno de los nodos de los mapas de la aplicación disponemos de varias clases. Existe una clase `Punto` de la cual descienden otras dos clases: `PuntoPot` y `PuntoBusca`. La clase `PuntoPot`

se utiliza para construir el mapa con los campos de potenciales dentro de la clase `Info_mapa`, que a su vez utilizan las clases `mapa` y `mapaC`. La clase `PuntoBusca` dispone de los métodos y atributos necesarios para trabajar con los algoritmos de búsqueda A* y D*, es decir, los valores para ese nodo de las funciones f, g, h y k. Esto hace que la clase `PuntoBusca` pueda ser usada por ambos algoritmos de búsqueda, aunque se desaproveche memoria, ya que nunca se van a usar todos los atributos, sea cual sea el algoritmo que utilizemos. Los algoritmos de búsqueda están implementados en este trabajo en la clase `BuscaCamino`.

En la estructura de clases nos encontramos con la clase abstracta `Datos`. De ella descienden dos clases: `Lista` y `BinaryHeap`. Estas dos clases implementan las dos estructuras de datos en las que podemos almacenar las listas de abiertos y cerrados de nuestros algoritmos. La clase abstracta `Datos` se utiliza para cambiar en tiempo de ejecución el tipo estructura de datos que se usa para almacenar las listas de abiertos y cerrados. En el apartado 5.2.2 se describe esto con detalle.

Disponemos de dos clases para cada uno de los mapas que vamos a usar en la aplicación: la clase `mapa` para el espacio de trabajo y la clase `mapaC` para el espacio de las configuraciones. Estas clases no sólo tienen métodos y atributos para gestionar cada uno de los espacios sino que se encargan de la visualización de los mapas y del último camino recorrido. Cada una de estas clases tiene como atributo una referencia a una misma instancia de la clase `Info_mapa`. Esta clase va a contener las variables necesarias para la comunicación entre los dos mapas: último camino, posición actual del brazo robótico, nodos explorados para calcular el último camino, etc. Para comunicarse, un mapa cambia los valores de las variables que necesite y envía una señal al otro mapa para que actúe conforme a los nuevos valores.

La clase `Cespacio` se utiliza para obtener el espacio de las configuraciones en base al espacio de trabajo actual a través del método descrito en [BC97]. Desde la clase `mapaC` podemos acceder al espacio de trabajo actual ya que disponemos de una referencia a la clase `Info_mapa`. La obtención del espacio de las configuraciones deberá hacerse cada vez que el espacio de trabajo sea modificado por el usuario.

Como ya se ha comentado en otros apartados del trabajo, la búsqueda del camino debe hacerse sobre el espacio de las configuraciones. Tras la obtención del camino, el brazo deberá adoptar en el espacio de trabajo las configuraciones que forman el camino hasta alcanzar la configuración de destino. Para poder calcular el camino óptimo disponemos de la clase `BuscaCamino`. En esta clase están implementados los algoritmos de búsqueda A* y D*. Esta clase crea un mapa de objetos del tipo `PuntoBusca` y realiza la búsqueda del camino óptimo sobre él. Además del mapa debemos disponer de dos instancias de `Lista` o de `BinaryHeap` para crear las listas de abiertos y cerrados necesarias para los algoritmos.

La clase `Ventana` es la encargada de generar el entorno gráfico de la aplicación y de trabajar con los ficheros en los cuales podemos almacenar los mapas. Como podemos ver en el diagrama de clases, la clase `Ventana` crea una instancia de cada uno de los mapas y se encarga de conectarlos mediante señales y slots. También se crea una instancia de la clase `Info_mapa` para la información compartida entre los mapas.

5.1.2. Diagramas de secuencia

En este apartado se va a explicar la interacción entre objetos para cada una de las operaciones que el usuario puede realizar en la aplicación mediante diagramas de secuencia. De esta forma podremos observar claramente la comunicación que existe entre los distintos objetos que forman la aplicación, especialmente entre los mapas.

En la figura 5.2 podemos observar la comunicación entre los distintos objetos que forman la aplicación cuando el usuario desea añadir un obstáculo al mapa de trabajo. Como podemos observar en dicha figura, cada cambio en el mapa del espacio de trabajo se refleja en el mapa del espacio de las configuraciones. Para calcular el nuevo espacio de las configuraciones usamos el objeto `Cespacio`.

En la figura 5.3 tenemos el diagrama de secuencia correspondiente a la búsqueda del camino entre dos configuraciones del brazo robótico. La búsqueda se realiza sobre el mapa del espacio de las configuraciones. Una vez calculada

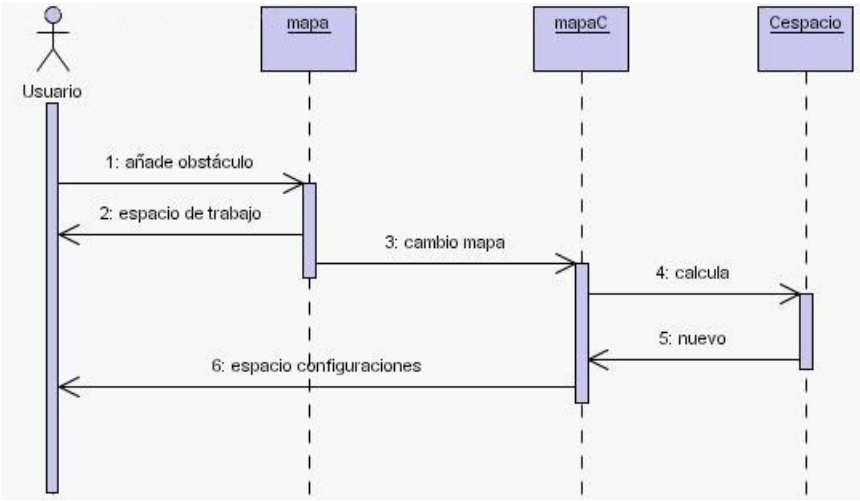


Figura 5.2: Diagrama de secuencia: añadir obstáculo.

la serie de configuraciones que va a permitir al robot moverse desde la configuración inicial a la configuración final evitando los obstáculos del entorno, se pinta el camino en el mapa del espacio de las configuraciones y se realiza una animación del brazo robótico de forma que recorra dicha serie de configuraciones en el espacio de trabajo. También se almacena el resultado de la búsqueda para no tener que volver a realizar los cálculos en el caso de tener que repintar alguno de los mapas.

5.2. Detalles de implementación

5.2.1. Comunicación entre mapas

De los dos mapas que tenemos en la aplicación sólo podemos actuar sobre el mapa del espacio de trabajo. El mapa del espacio de las configuraciones refleja los cambios que hagamos en el espacio de trabajo añadiendo obstáculos.

Ya hemos comentado que la librería gráfica utilizada para la implementación de la aplicación está basada en señales y slots. Este es el mecanismo que vamos a utilizar para implementar la comunicación necesaria entre los dos mapas de nuestra aplicación.

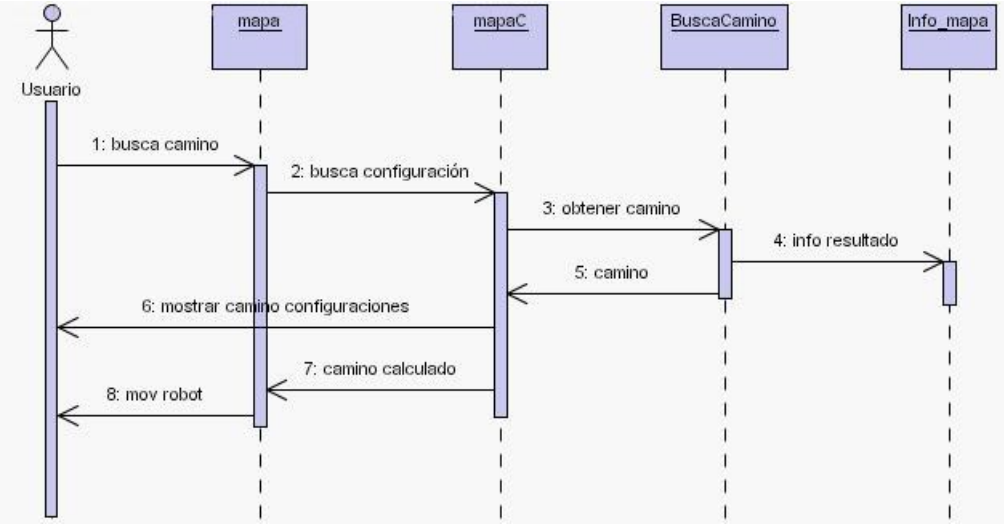


Figura 5.3: Diagrama de secuencia: búsqueda del camino óptimo.

Cada vez que añadimos un obstáculo o pedimos que el robot se desplace hasta otra posición debemos indicarlo en el espacio de trabajo. Esto se captura como un evento, y tras realizar las acciones oportunas, se envía una señal al mapa de las configuraciones para que este refleje los cambios oportunos. Esta señal incluye datos sencillos sobre el evento recogido y los cambios a efectuar en base a ese cambio.

Sin embargo, en el caso de que la acción deseada por el usuario sea la búsqueda de un camino entre dos configuraciones del robot, la comunicación tiene que ser en los dos sentidos, ya que una vez calculado el camino a seguir, el brazo deberá moverse pasando por las configuraciones que forman el camino elegido. En este caso no sería eficiente comunicar los mapas sólo mediante señales, debemos utilizar alguna zona de memoria común a los dos objetos para poder intercambiar tanta información.

La solución elegida ha sido implementar un objeto (`Info_mapa`) que contenga la información necesaria para los dos mapas y las variables que tienen que compartir: camino elegido, nodos explorados, posición del robot, etc. De esta forma los mapas se comunicarán mediante señales en el caso de que ocurra algún evento que suponga un cambio en los mapas o cada vez que alguno de los mapas cambie el valor de alguna de las variables que utilizan los mapas para

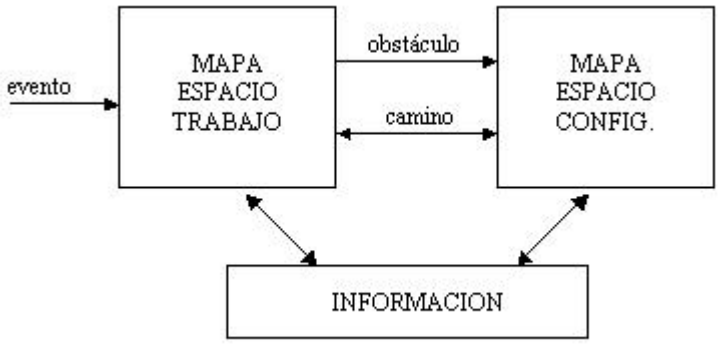


Figura 5.4: Comunicación entre mapas

intercambiar información y quiera que el otro mapa actúe en consecuencia. Todo esto podemos verlo en la figura 5.4.

De esta forma es fácil añadir más tipos de mapas a la aplicación. Sólo debemos implementar la clase correspondiente, instanciar un objeto que tenga como atributo una referencia al objeto Info_mapa y comunicar el nuevo mapa con el resto de los mapas mediante señales y slots.

5.2.2. Implementación de la lista de abiertos

A la hora de implementar los dos algoritmos de búsqueda de caminos utilizados en este trabajo, se han usado los dos tipos de estructuras de datos con una serie de características comunes.

En cualquiera de los dos casos nuestro objetivo es representar la lista de abiertos necesaria para la ejecución de los dos algoritmos. El mapa sobre el cual vamos a realizar la búsqueda es cuadrado y de un tamaño fijo en ejecución. Por lo tanto, el tamaño máximo de la lista de abiertos es conocido antes de empezar la búsqueda. Este dato es utilizado en las dos estructuras de datos para reservar la memoria necesaria para el peor de los casos antes de comenzar la ejecución. Aunque en la inmensa mayoría de los casos esto va a suponer un desaprovechamiento de la memoria del sistema nos va a evitar tener que reservar memoria cada vez que queramos insertar un nodo. Si tenemos en cuenta que una

petición de memoria al sistema es una llamada muy costosa en tiempo, en la implementación realizada para este trabajo de las dos estructuras de datos antes descritas se antepone la velocidad (sólo una petición de memoria) al espacio empleado en memoria. En el caso de disponer de poca memoria quizá sería interesante cambiar este enfoque.

La segunda característica que tienen en común las implementaciones de las dos estructuras de datos es el hecho de que ninguna trabaja con copias de los datos del mapa. Cada uno de los nodos que forman nuestro espacio de búsqueda es un objeto que almacena mucha información útil para la búsqueda. Si trabajamos con copias de dichos objetos en nuestra lista de abiertos, no sólo aumentará muchísimo el espacio ocupado en memoria por la lista, sino que a la hora de reordenar las listas el movimiento de información en memoria puede ser importante. La solución es no almacenar los datos en si, sino un puntero al nodo correspondiente. A través del puntero tendremos acceso a todos los métodos del objeto que representa al nodo, el tamaño de la lista de abiertos disminuirá considerablemente y será mucho menos costoso reordenar la lista en caso de que fuera necesario. Un ejemplo en el cual podemos ver todo esto es el constructor de la clase `BinaryHeap`:

```
BinaryHeap::BinaryHeap() :currentSize(0)
{
     array=(PuntoBusca **)calloc((TAMMAPAX*TAMMAPAY)+1,
                                 sizeof(PuntoBusca *));
}
```

Otro detalle a tener en cuenta es que las clases donde se implementan las estructuras de datos tienen los mismos métodos públicos. La primera razón para que esto sea así es evidente: las dos estructuras de datos se usan para lo mismo. Pero hay otra razón para que esto sea así: podemos utilizar una clase abstracta en la cual definimos los métodos públicos comunes como virtuales y hacer que las clases que implementan las estructuras de datos hereden de dicha clase abstracta. Si nos interesa intercambiar la estructura de datos para la lista de abiertos nos bastará con declararnos un puntero a la clase base, y en función de lo que necesitemos que apunte a un objeto de tipo lista o a un objeto de tipo

binary heap.

En el código:

```
Datos *listaAbiertos;
.............
if(tipoEstructDatos==0) listaAbiertos=new Lista;
else listaAbiertos=new BinaryHeap;
```

5.2.3. Búsquedas en paralelo

Como ya se comentó en el apartado 4.2, en el caso particular de un robot planar articulado de dos grados de libertad disponemos de dos posibles configuraciones para alcanzar el mismo punto del espacio de trabajo con el extremo del brazo robótico. Esto obliga a calcular ambos caminos, ya que no podemos segurar antes de calcularlos cuál de los dos va a ser el más corto.

En el caso de disponer de dos o más CPUs se pueden lanzar las dos búsquedas en hilos de ejecución distintos, de forma que cada búsqueda se realice en una CPU distinta. Al terminar las búsquedas se pueden comparar los resultados para devolver como resultado de la búsqueda el camino más corto. De esta forma, el tiempo de cómputo empleado en la búsqueda ambos caminos será ligeramente superior a la búsqueda de un único camino.

Tras esto, podemos optimizar aun más el cáculo en paralelo de ambos caminos utilizando la memoria compartida entre ambos hilos de ejecución. Como ya hemos comentado anteriormente, los algoritmos A* y D* generar árboles de búsqueda, con el nodo inicial en la raíz y el nodo final en una de las hojas. Aparte de esto, dentro de la función f que guía la búsqueda, el término g nos indica el coste de llegar a un nodo determinado desde el nodo inicial. En el apartado 4.4 ya comentamos que el coste de pasar de un nodo a uno de sus nodos vecinos era 1. Por lo tanto, en este caso particular, el valor de g para un determinado nodo coincide con la altura del árbol de búsqueda, o lo que es lo mismo, con la longitud del camino en un momento determinado de la búsqueda.

En el momento en el que un hilo de ejecución termine la búsqueda podemos

conocer el valor de g para ese nodo, o desde otro punto de vista, podemos calcular la longitud del camino encontrado. Dado que disponemos de memoria compartida, podemos comunicar al otro hilo dicha información, de forma que si la altura del árbol de búsqueda del hilo que no terminó es mayor que la del hilo que ya terminó podemos dar por terminada la búsqueda, ya que el resultado nunca va a ser mejor que el del otro hilo (el camino encontrado no va a ser más corto). Sin embargo, si es menor debemos continuar la búsqueda hasta encontrar el nodo destino o superar el valor de g del hilo que ya terminó.

Para comunicar los dos hilos de ejecución se ha utilizado esta estructura:

```
struct paralelo
{
   int fin, prof;
}
```

La variable `fin` nos indica si alguno de los dos hilos ha terminado y `prof` nos indica el valor de g del nodo destino para ese camino, o lo que es lo mismo, la profundidad en la búsqueda del camino.

Las modificaciones en los algoritmos para poder ejecutarse en paralelo son mínimas. Basta con almacenar la profundidad máxima de la búsqueda en una variable propia de cada uno de los hilos,

```
if(mi_profundidad<MapaBusca[(*it)->x()][(*it)->y()].g())
   mi_profundidad=(int)MapaBusca[(*it)->x()][(*it)->y()].g();
```

incluir dicha información en la zona de memoria compartida una vez terminada la búsqueda

```
if(info->fin==0) {info->fin++; info->prof=mi_profundidad;}
```

y comprobar en cada iteración si el otro hilo ha terminado, y si ha terminado comparar la profundidad del hilo que ha terminado con la la alcanzada hasta el momento por el hilo que no ha terminado. Un buen sitio para realizar la comprobación es el if donde se comprueba si la lista de abiertos está vacia.

```
if(listaAbiertos->vacia() ||
   (info->fin==1 && mi_profundidad>info->prof))
```

Uno de los problemas a los que suele enfrentarse un programador a la hora de implementar un programa que se ejecuta en distintos hilos de ejecución es el de coordinar el acceso concurrente a la información compartida, que en este caso es la estructura *paralelo*. En este caso particular no hace falta realizar un control mediante zonas de exclusión mutua, ya que el único caso en el que los dos hilos pueden intentar escribir a la vez en la zona de memoria compartida es que ambos hilos terminen a la vez. En ese caso la información contenida en la zona de memoria compartida ya no es importante y no se va a volver a utilizar.

5.2.4. Herramientas utilizadas en el desarrollo

La plataforma utilizada para el desarrollo es Linux Fedora Core 2, aunque la aplicación es completamente compatible con cualquier otra versión de Linux. Se ha elegido este sistema operativo porque es de libre distribución y dispone de muchas herramientas y bibliotecas de funciones (también de libre distribución) que serán de gran utilidad a la hora de desarrollar la aplicación.

El lenguaje elegido es el C++, no sólo por ser un lenguaje potente y muy utilizado, sino porque las librerías utilizadas para el desarrollo están pensadas para trabajar fundamentalmente con C++. Esta elección también nos permite diseñar e implementar nuestra aplicación utilizando orientación a objetos, con todas las ventajas que eso conlleva.

Se ha utilizado la librería gráfica Qt para crear la interfaz gráfica de la aplicación. Se ha utilizado la versión libre de esta librería. Es una librería fácil de usar, muy potente, muy extendida y multiplataforma, lo que permite portar la aplicación a otras plataformas como Microsoft Windows. Esta librería basa su funcionamiento en signals y slots. Cada vez que un objeto cambia de estado y este cambio puede ser interesante para el resto de los objetos emite una señal. Estas señales se pueden recoger mediante slots y de esta manera actuar en base al cambio del objeto que emitió la señal. Esto va a ser importante en el diseño de la aplicación para la coordinación entre los distintos mapas que forman la aplicación. Todo esto se estudiará con detalle más adelante.

Otra librería que vamos a utilizar es la fftw. Esta librería implementa las

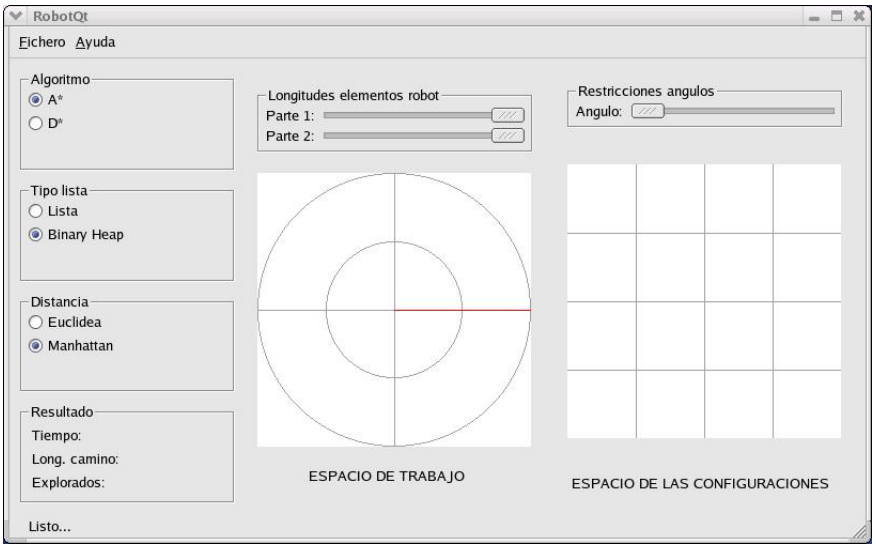


Figura 5.5: Aspecto de la aplicación desarrollada.

funciones necesarias para obtener la transformada rápida de Fourier. Esto es necesario para poder construir el espacio de las configuraciones mediante el método descrito en [BC97]. La versión utilizada es la 2.1.5. En el caso de esta librería la versión es muy importante, porque la última versión desarrollada de la librería fftw es la 3.0.1 y esta versión no es compatible con las anteriores.

5.3. Uso de la aplicación

En este apartado vamos a explicar como funciona la herramienta que hemos desarrollado para poder visualizar los resultados obtenidos por los algoritmos de búsqueda. Tras ejecutar la aplicación se nos presenta una ventana como la de la figura 5.5. En la ventana podemos observar cuatro partes diferenciadas:

1. En la primera englobamos todos los parámetros con los que podemos variar las condiciones de búsqueda: longitud de las partes del robot, restricciones en las articulaciones, tipo de algoritmo utilizado para la búsqueda, tipo de estructura de datos a usar por el algoritmo y el tipo de cálculo de distancia a emplear por el algoritmo; y un resumen del resultado obtenido: tiempo de ejecución, tamaño del camino encontrado y tamaño de la lista de abiertos.

2. El primer mapa (izquierda) que nos encontramos en la aplicación es el espacio de trabajo, donde podemos ver la posición actual del brazo del robot, obstáculos y el último movimiento realizado por el robot.

3. El segundo mapa (derecha) es el espacio de las configuraciones, donde podemos observar la representación del robot como un punto, el último camino encontrado y los nodos explorados para encontrarlo, la representación de los obstáculos que tenemos en el espacio de trabajo y el campo de potenciales de repulsión que generan los obstáculos.

4. Por último disponemos de un menú (Fichero) que nos va a permitir guardar el mapa con el que estamos trabajando y la posición del brazo en ese momento, abrir un escenario guardado anteriormente o salir de la aplicación.

Sólo podemos interactuar con el mapa de la izquierda, el que representa el entorno del robot. En él podemos picar con el botón izquierdo del ratón si queremos que el extremo del brazo del robot alcance la posición que le indiquemos, o con el botón derecho en el caso de que lo que queramos hacer sea incluir un obstáculo en el espacio de trabajo. En el espacio de las configuraciones podremos observar las representaciones de los obstáculos que incluimos en el espacio de trabajo y la posición del robot dada como los ángulos de las dos partes del robot.

Los obstáculos generan un campo de repulsión del robot para evitar que el robot se aproxime a ellos fácilmente. Este campo se representa en el campo de las configuraciones mediante el color azul. A medida que nos acercamos a los obstáculos la fuerza de repulsión es mayor. Esto se representa con la intensidad del color azul que es mayor a medida que nos acercamos a un obstáculo.

Las búsquedas se realizan en el espacio de las configuraciones y el resultado se refleja en el movimiento del brazo en el mapa de trabajo. Los nodos explorados aparecen coloreados en verde. Si los nodos explorados se ven afectados por una fuerza de repulsión causada por la proximidad de un obstáculo, quedan coloreados en color morado. La intensidad del color morado se corresponderá con la intensidad de la fuerza de repulsión ejercida por el obstáculo. De esta forma podemos ver claramente de que forma se ha ido explorando el árbol de posibles nodos y que parte del mapa se ha explorado hasta encontrar el camino

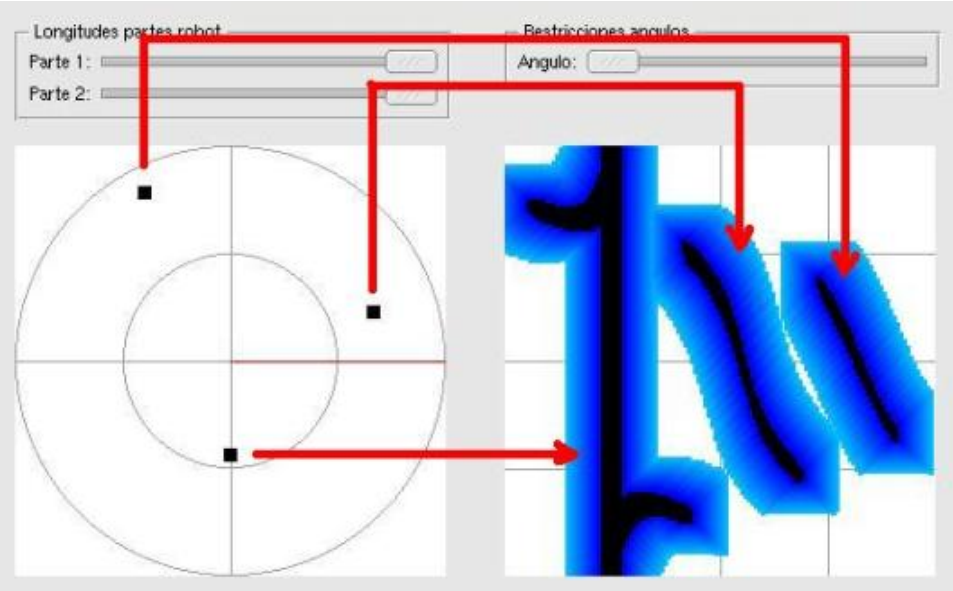


Figura 5.6: Relación entre obstáculos en el mapa de trabajo y el mapa de las configuraciones

óptimo. También podemos observar el camino óptimo entre la configuración de origen y la de destino pintado en rojo en el espacio de las configuraciones y las distintas posiciones por las que va pasando el brazo hasta llegar a la posición de destino en verde en el espacio de trabajo.

En la figura 5.7 podemos observar el movimiento del robot, tanto en el espacio de trabajo como en el de las configuraciones, los nodos explorados, etc.

Es importante tener en cuenta una peculiaridad del caso de un robot planar articulado: un punto determinado en el espacio de trabajo se corresponde con dos configuraciones distintas del brazo robótico: codo arriba y codo abajo. Por lo tanto, si queremos calcular el camino óptimo, debemos calcular los caminos desde la configuración de origen a las dos posibles configuraciones de destino. De los dos caminos que se calculan sólo se representa el más corto de los dos. Sin embargo, el tiempo que se muestra es el tiempo empleado en el cálculo de los dos caminos, ya que para podernos decidir debemos calcular ambos o al menos debemos calcular uno completo y explorar el camino hacia el otro destino posible hasta llegar a la misma profundidad en el árbol de búsqueda sin encontrar el

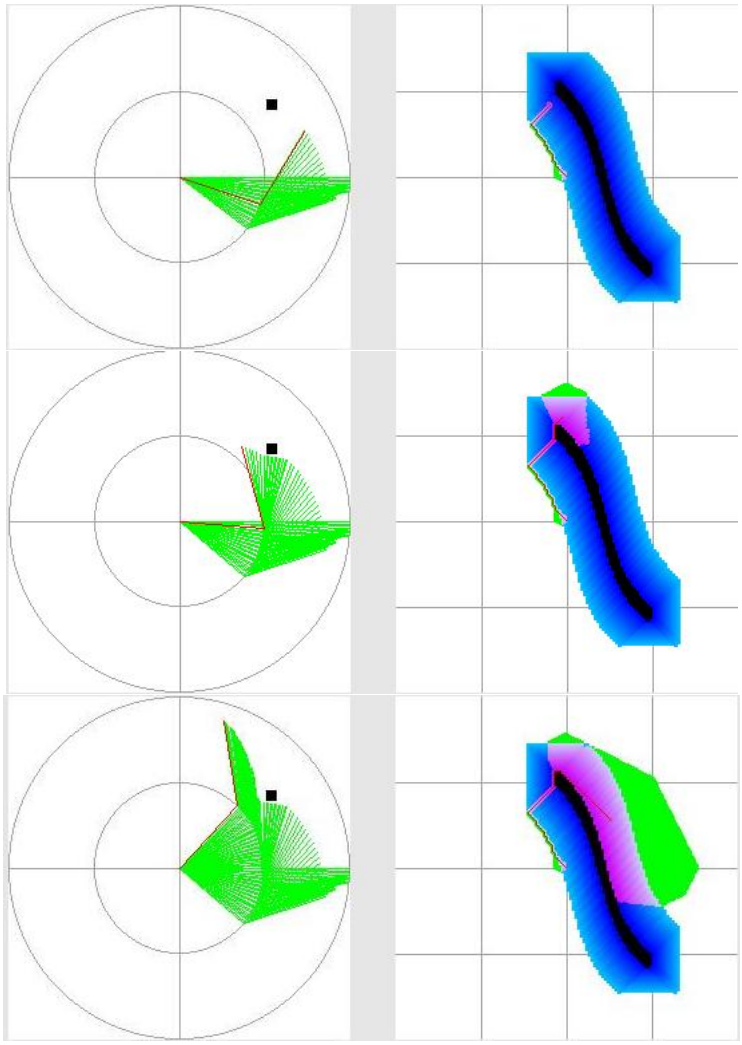


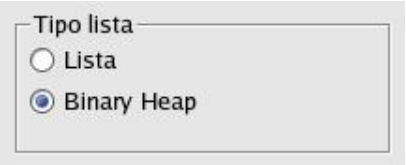
Figura 5.7: Movimiento del robot. Nodos explorados y camino elegido.

nodo destino. De esta forma nos podemos asegurar de que elegimos el camino más corto de los dos posibles.

5.3.1. Algoritmos de búsqueda

La aplicación nos permite seleccionar un algoritmo para realizar la búsqueda, lo que nos va a permitir comparar el rendimiento de los distintos algoritmos implementados y ver claramente las diferencias que hay entre ellos.

Otro de los parámetros que podemos configurar desde la interfaz gráfica es el tipo de estructura de datos elegida para albergar la lista de abiertos. Se han implementado dos posibilidades: una lista enlazada ordenada y un montículo binario. Dada la importancia de esta estructura de datos para los algoritmos elegidos para la búsqueda de caminos, en determinados casos puede ser importante el uso de una estructura de datos u otra.

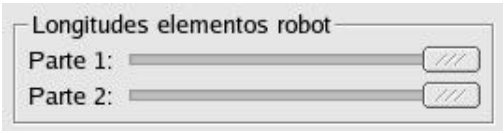


5.3.2. Cálculo de la distancia

Podemos modificar el método empleado para calcular la distancia de un determinado nodo hasta el nodo destino. Se han implementado la distancia Euclidea y la distancia Manhattan.

5.3.3. Modificación del robot

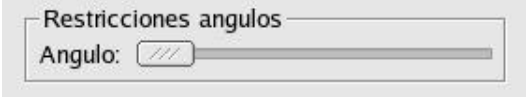
Otra posibilidad que nos ofrece la aplicación es cambiar el tamaño de cualquiera de los dos elementos que forman el robot. Esto nos va a permitir observar como afecta al mapa del espacio de las configuraciones el cambio de tamaño de los elementos que forman el robot.



En el caso de disminuir el tamaño de alguno de los elementos que forman el robot, podemos observar como el tamaño de los obstáculos en el espacio de las configuraciones se reduce, ya que al reducir el tamaño de los elementos del robot estamos reduciendo el número de configuraciones del robot que podrían ocasionar una colisión con alguno de los obstáculos. En el caso de aumentar el tamaño de uno de los elementos del robot, el efecto será el contrario, ya que aumentamos el número de configuraciones que producen colisión. Podemos observar esto en la figura 5.8, en la que se reduce el tamaño de ambas partes del robot (abajo) respecto de la de arriba. Como podemos observar, los obstáculos han cambiado de forma y han reducido su tamaño.

5.3.4. Restricciones de ángulos

La última posibilidad de la que disponemos es variar lo que se ha llamado en la aplicación "Restricciones de ángulos".



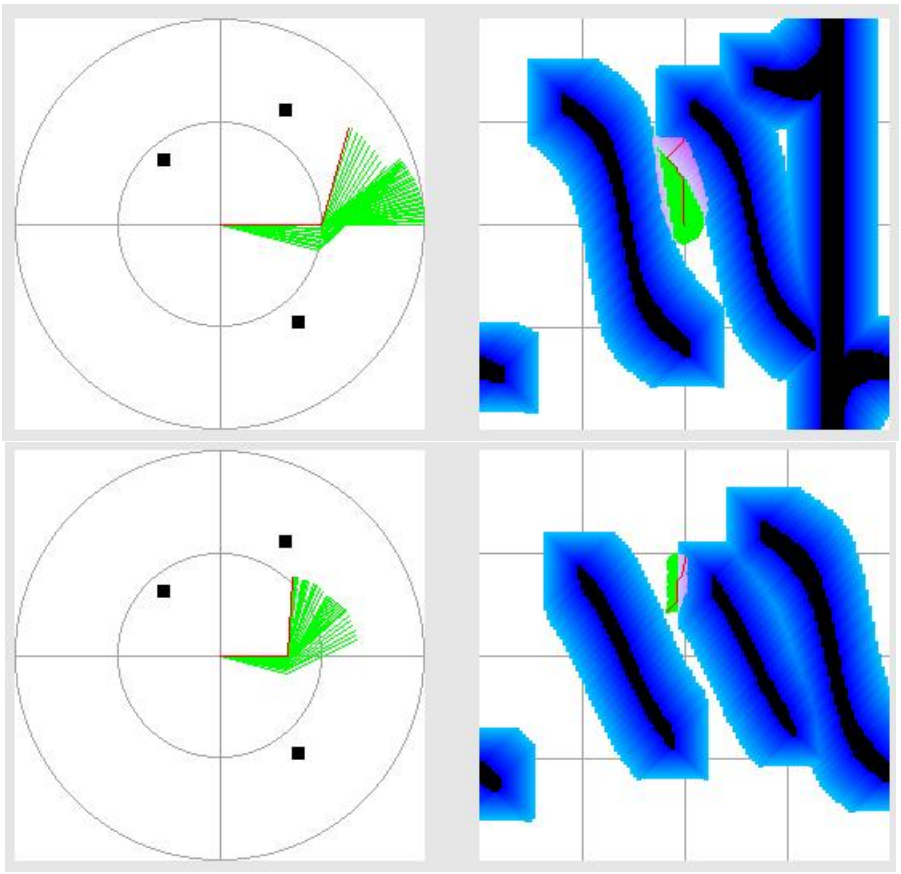


Figura 5.8: Cambios en el Cespacio en base a la longitud de los elementos del robot.

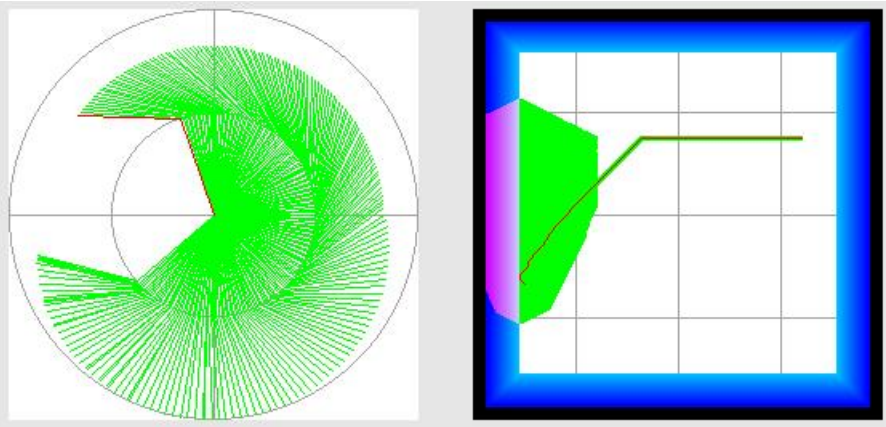


Figura 5.9: Restricciones de angulos de rotación para las articulaciones del robot.

Cuando trabajamos con un brazo robótico real, las partes que forman el brazo no pueden rotar 360^0 sobre el eje de giro, o dicho de otra forma, nunca pueden llegar a formar un ángulo de π o -π. Para poder trabajar en nuestra aplicación con esta restricción debemos convertir en obstáculos todas las configuraciones que en un robot real no podríamos alcanzar. En el caso del espacio de las configuraciones, estas configuraciones se encuentran en el borde del mapa. Mediante el movimiento de la barra de desplazamiento podemos restringir más o menos las posibilidades de giro de las partes del brazo.

Como podemos observar en la figura 5.9 el camino más corto entre las dos configuraciones está en dirección contraria pero no podemos alcanzar la configuración de destino de esta forma porque el brazo no puede girar lo suficiente, ya que ese camino pasa por configuraciones en las que la primera parte del brazo del robot forma un ángulo de π. Esto obliga al algoritmo a elegir un camino más largo pero que si sea alcanzable por el robot.

6

Resultados

6.1. Aplicación de pruebas

Para poder realizar medidas del rendimiento y comportamiento de los algoritmos implementados se ha desarrollado una aplicación que va a generar automáticamente mapas con distinto número de obstáculos y va a realizar búsquedas de caminos entre distintas configuraciones de inicio y de fin elegidas aleatoriamente. De esta forma dispondremos de muchos datos de búsquedas de caminos utilizando los algoritmos desarrollados sobre una gran variedad de situaciones, lo que nos permitirá realizar un estudio sobre dichos algoritmos.

La herramienta tiene dos modos de funcionamiento en base a los argumentos que se le pasen al programa:

- Un modo en el que pasamos al programa un mapa ya construido y la aplicación genera varios caminos usando configuraciones aleatorias de inicio y fin.

- Otro modo en el que la aplicación recibe como argumento el número de obstáculos que queremos y genera automáticamente varios mapas y varios caminos usando configuraciones aleatorias de inicio y fin.

Los resultados obtenidos se van a almacenar en ficheros de texto, de tal forma que se puedan interpretar fácilmente mediante alguna herramienta de generación de gráficas para poder relacionar e interpretar de manera sencilla los resultados obtenidos.

6.2. Análisis de rendimiento

Las pruebas de los algoritmos implementados se han realizado sobre una máquina con una sola CPU Pentium 4 a 2GHz, 225 Mb de memoria RAM y 512 kb de memoria caché. Utilizando el programa que se describe en 6.1, se han buscado 50 caminos distintos entre configuraciones de origen y destino elegidas de forma aleatoria para un total de 15 mapas distintos con un número determinado de obstáculos.

En la figura 6.1 podemos observar algunos de los resultados obtenidos con el algoritmo de búsqueda D*. Los resultados obtenidos usando el algoritmo A* son prácticamente iguales. La primera figura se corresponde con los resultados obtenidos para mapas con un sólo obstáculo, el siguiente son los resultados obtenidos para mapas con dos obstáculos y así sucesivamente. En cada gráfico podemos observar el número de nodos explorados frente al tiempo empleado en encontrar un camino desde la configuración de origen hasta la configuración de destino, elegidas de forma aleatoria por la aplicación de pruebas.

El rendimiento de los algoritmos es el mismo: $O(nlogn)$ como ya se indica en trabajos como [Tro96]. Esto es debido a que el algoritmo D* es una mejora para entornos dinámicos del algoritmo A*. La aplicación desarrollada solamente

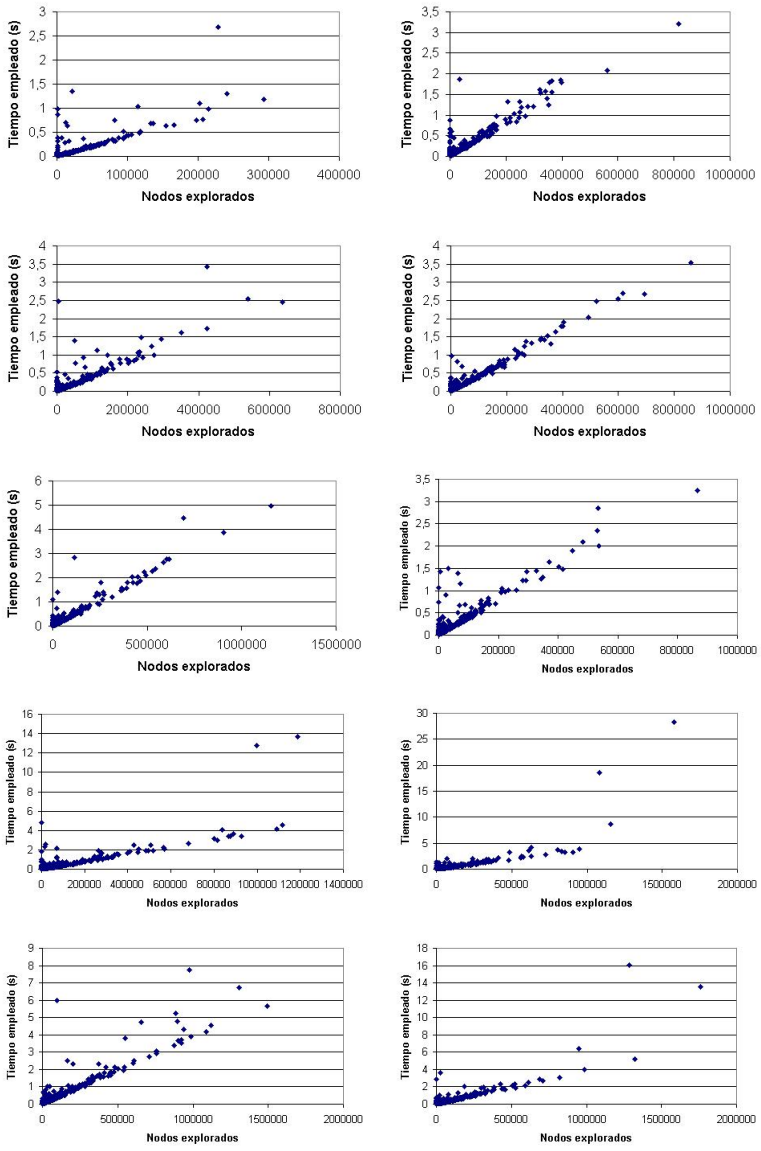


Figura 6.1: Resultados obtenidos con los algoritmos de búsqueda

genera espacios de trabajo estáticos, por lo que el comportamiento del D* es idéntico al del A*. Los resultados también son los mismos si utilizamos distintas estructuras de datos.

6.3. Coste mínimo de los algoritmos

Cada uno de los algoritmos implementados lleva asociado un coste computacional que no es el empleado en el cálculo del camino pero que aparece reflejado en los resultados representados en 6.1. Se han calculado cuales son dichos tiempos en la misma máquina que se han realizado las pruebas. Este tiempo se emplea en las siguientes operaciones:

- Reserva y liberación de memoria para el mapa de búsqueda. Para ocultar al usuario de las funciones de búsqueda qué información de cada nodo del mapa es necesaria para la implementación del algoritmo, el mapa de búsqueda se genera dentro de la función y se libera al terminar la búsqueda. En la reserva de memoria se emplean 10325 μs y en la liberación de la memoria reservada 182 μs.

- Comparaciones necesarias para el cálculo en paralelo de los dos caminos necesarios en cada movimiento del brazo. Se han incluido ciertas comprobaciones que nos permiten comprobar en cada iteración del algoritmo si se ha terminado de calcular el otro camino y si además la profundidad del árbol de búsqueda generado tiene mayor profundidad que el generado para el otro camino. El tiempo empleado en estas operaciones es despreciable.

- Creación de la lista de nodos explorados. En cada iteración se inserta en la lista de nodos explorados el nuevo nodo elegido para continuar la búsqueda. El tiempo empleado en insertar cada uno de esos nodos es de 25 μs aproximadamente.

- Preparación de la lista que contiene el camino calculado para su devolución como resultado de la llamada a la función de búsqueda. Tiempo empleado: 25 μs aproximadamente para cada uno de los nodos que forman el camino.

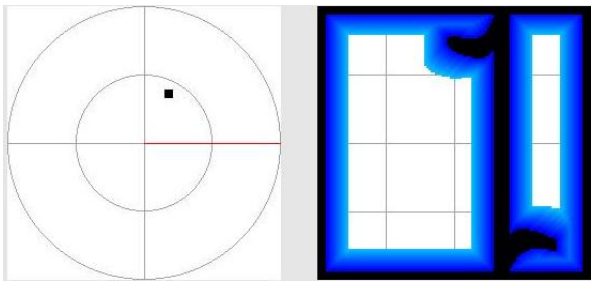


Figura 6.2: Ejemplo de mapa con configuraciones no accesibles

Por lo tanto, el tiempo empleado en operaciones auxiliares que no son estrictamente el cálculo del camino, se puede calcular como:

$$t_{exceso} = 10352 + 182 + (25 * N_{expl}) + (25 * N_{camino})\ \mu s$$

6.4. Resultados para configuraciones no accesibles

En los resultados de la figura 6.1 no se han incluido búsquedas de caminos hasta configuraciones que no se pueden alcanzar. En el espacio de las configuraciones de la figura 6.2 podemos observar que no existe ningún camino que pueda llevar a las configuraciones que quedan a la derecha del obstáculo desde la configuración inicial del robot (centro del mapa de configuraciones). Si intentamos que el brazo robótico alcance alguna de dichas configuraciones, cualquiera de los dos algoritmos de búsqueda determinará que es imposible alcanzar la configuración de destino (como ya vimos en los apartados 3.3 y 3.4) y la aplicación informará al usuario mediante un mensaje en la barra de estado.

En ambos casos el tiempo obtenido es de algo menos de 1 segundo.

6.5. Nodos explorados

Como podemos observar en la mayoría de las gráficas, el número de nodos explorados que se representa es bastante mayor que el número de nodos que forman el mapa. Esto es así porque se identifican como nodos distintos aquellos que pese a tener coordenadas (θ_1, θ_2) iguales tienen un valor de la función f distinto.

Si obtenemos el valor medio para un número alto de ejecuciones de:

$$\frac{nodosexploradosdistintaf}{nodosexplorados}$$

podremos disponer de una medida que nos indique las veces que se ha pasado de media por cada uno de los nodos en la búsqueda de un camino. De esta forma podremos evaluar el grado de información de la que disponemos en la representación del entorno. Los valores obtenidos son:

	Algoritmo A*	Algoritmo D*
Valor medio	6.8894	8'8148
Valor más alto	7.7661	12.4925
Valor más bajo	1.7142	1.7391

Es decir, cada uno de los nodos se visita una media de 7.85 veces. Lógicamente, algunos nodos sólo se explorarán una vez, y otros se explorarán muchas más veces. Los nodos más próximos a los obstáculos que el robot tenga que esquivar para alcanzar la configuración de destino serán los más visitados.

Si tenemos en cuenta todos los nodos visitados con distinto valor de f en las pruebas y los nodos realmente visitados, el 86 % de los nodos que explora cualquiera de los dos algoritmos son nodos explorados anteriormente pero con distinto valor de f, es decir, nodos a los que se ha llegado explorando distintas ramas del árbol de búsqueda.

Como podemos observar en la figura 6.3, si en la búsqueda del camino el robot debe superar un obstáculo de gran tamaño, el algoritmo debe explorar

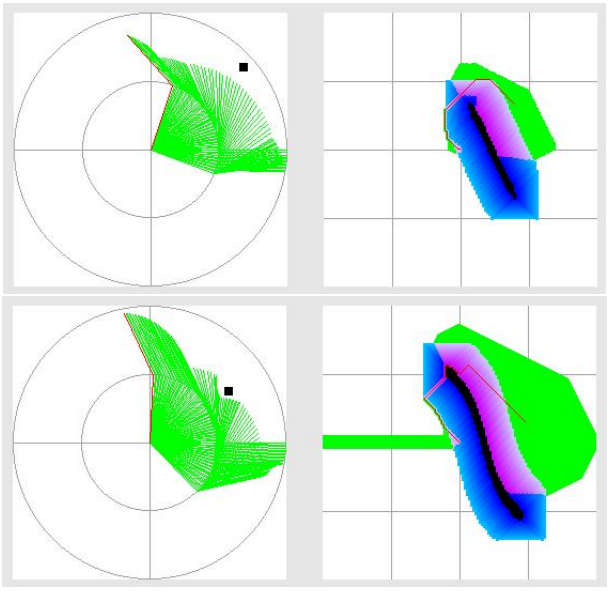


Figura 6.3: Campo de potenciales y obstáculos de distinto tamaño

en amplitud una gran parte del árbol de búsqueda hasta que se logra superar el obstáculo, lo que supone explorar el mismo nodo llegando a él desde muchas ramas distintas. Sin embargo, si el obstáculo no es muy grande, el campo de repulsión que rodea al obstáculo es capaz de redirigir la búsqueda de tal forma que se necesitan explorar muchos menos nodos para alcanzar la configuración de destino.

6.6. Campo de repulsión de los obstáculos

Una solución a lo planteado en el punto anterior podría ser el aumentar el tamaño del campo de repulsión que rodea a los obstáculos intentando de esta forma aumentar la información contenida en el mapa de búsqueda, pero como podemos ver en la figura 6.4, el número de nodos no sólo no disminuye, sino que aumenta debido a que el campo de repulsión penaliza nodos que antes no se veían afectados por el campo de repulsión, y el algoritmo de búsqueda no los selecciona como candidatos al camino óptimo. Dicho de otra forma, aumentando el tamaño del campo de repulsión que rodea a los obstáculos sólo conseguimos

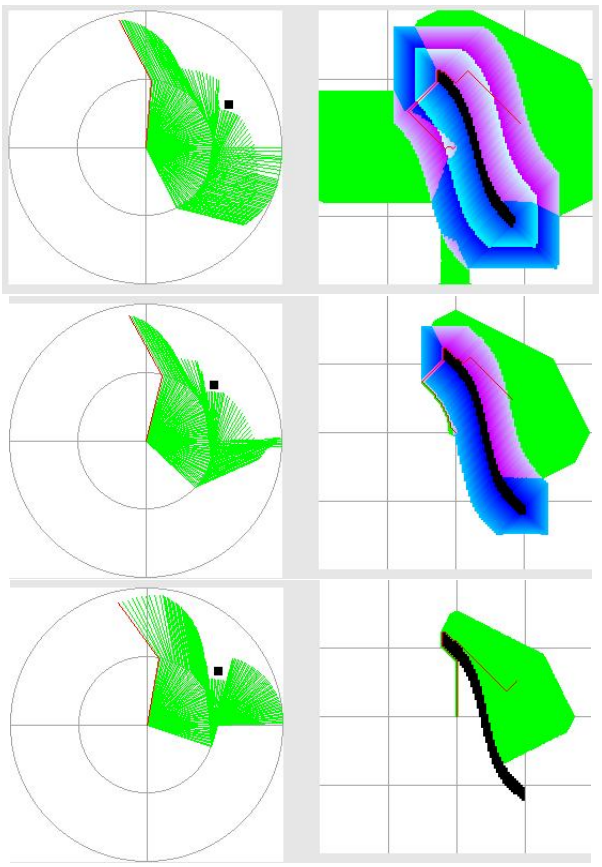


Figura 6.4: Aumento y disminución del tamaño del campo de repulsión

aumentar el tamaño del obstáculo para el algoritmo de búsqueda.

También en la figura 6.4 podemos observar los resultados planteando el caso contrario, es decir, si se elimina el campo de potenciales. En este caso el número de nodos explorados muy similar que en el caso de utilizar el campo de potenciales planteado inicialmente y mucho menor en el caso de doblar el tamaño del campo de repulsión que rodea a los obstáculos. Ya que no existe mejora en la velocidad del cáculo del camino si añadimos el campo de repulsión de los obstáculos, usarlo dependerá de otras utilidades del campo de repulsión, como, por ejemplo, evitar que el robot se acerque demasiado a los obstáculos.

Con esto queda de manifiesto la importancia de utilizar un tamaño de campo de repulsión ante los obstáculos no excesivamente grande, ya que obligamos a los algoritmos de búsqueda a dar un rodeo mayor para evitar el obstáculo.

Esto pone de manifiesto el hecho de que la representación del mapa de búsqueda mediante un campo de potenciales dispone de poca información, como ya se adelantaba en 4.5, pero se construye de manera muy rápida, y es suficiente para tener un rendimiento bueno a la hora de calcular los caminos entre configuraciones.

6.7. Cálculo de la distancia

La aplicación desarrollada dispone de dos maneras distintas de calcular la distancia al nodo destino: la distancia euclidea y la distancia manhattan. Se han realizado pruebas con distintos mapas y distinto número de obstáculos para determinar si la manera de calcular la distancia al nodo destino influye de alguna manera en el tiempo empleado en la búsqueda y los nodos explorados para encontrar el camino hasta el nodo destino. Los resultados obtenidos han sido los siguientes:

	Longitud camino	Nodos explorados	Tiempo (s)
Euclidea	100'774	61894'857	0'659
Manhattan	94'833	38947'389	0'421

Como podemos observar en la tabla anterior, aunque la media de los tiempos empleados en las búsquedas es muy similar, el número de nodos explorados es bastante mayor en el caso de usar la distancia euclidea.

Como podemos ver en la figura 6.5, el hecho de que el número de nodos explorados sea mayor se refleja en la forma que generan los nodos explorados representados en verde en la aplicación. Incluso, como podemos ver en el segundo ejemplo, el camino encontrado varia en determindos casos. En las imagenes impares se ha calculado la distancia euclidea al destino, mientras que en las pares se ha calculado la distancia manhattan.

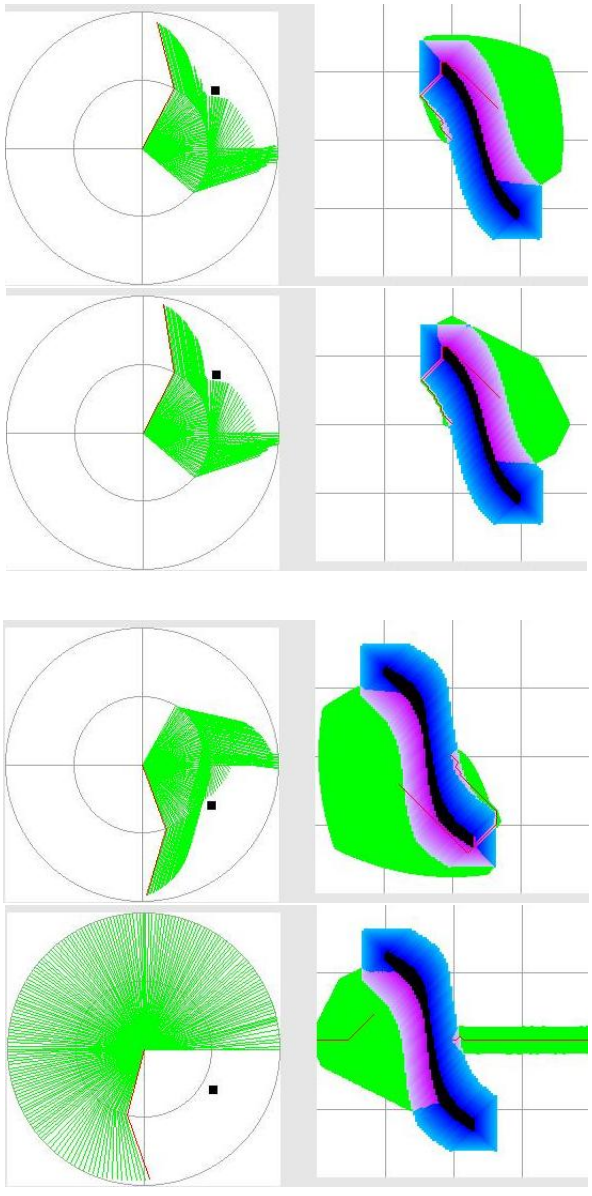


Figura 6.5: Distintos cálculos de la distancia al nodo destino.

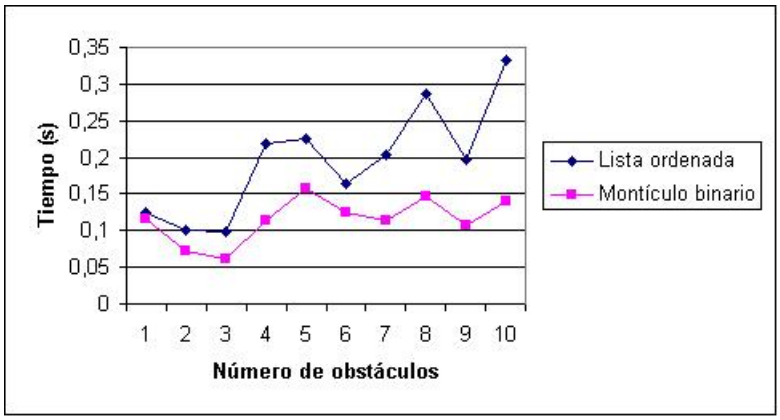


Figura 6.6: Resultados con distintas estructuras de datos para almacenar la lista de abiertos.

6.8. Lista de abiertos

Al igual que con el cálculo de la distancia a la configuración de destino, se han implementado dos posibles estructuras de datos para almacenar la lista de abiertos de los algoritmos de búsqueda.

Se han realizado pruebas con distintos mapas y configuraciones de inicio y fin para poder comparar el rendimiento de los algoritmos de búsqueda en el caso de elegir una estructura de datos u otra. Podemos observar los distintos resultados en la figura 6.6, en la que se representan los tiempos en segundos obtenidos de media en la búsqueda de distintos caminos en mapas con el número de obstáculos que indica el eje de las X.

Los resultados obtenidos nos demuestran que la estructura montículo binario es más eficiente que la lista ordenada. La diferencia entre los tiempos de media en las búsquedas utilizando las distintas estructuras aumenta a medida que se aumenta el número de obstáculos en los mapas de búsqueda, lo que indica que la mejora en la eficiencia de los algoritmos utilizando el montículo binario crece a medida que la dificultad en la búsqueda crece, o lo que es lo mismo, que la mejora utilizando el montículo binario es mayor a medida que la lista de abiertos contiene un mayor número de elementos.

7

Conclusiones y trabajos futuros

El objetivo fundamental de este trabajo ha sido la representación del entorno de un robot móvil mediante la técnica de los campos de potenciales y la implementación de algoritmos de búsqueda de caminos en dicho entorno y la medición de su rendimiento.

Se ha planteado una técnica de planificación en la que se ha dado el mayor peso de la búsqueda del camino a los algoritmos de búsqueda, lo que nos ha llevado a realizar una representación del entorno del robot mediante un mapa sin mucha información, pero muy rápido de construir. Esto nos puede permitir en futuros trabajos utilizar la misma técnica de planificación para entorno dinámicos, en los que debemos recalcular el espacio de trabajo cada muy poco tiempo.

Se han implementado los dos algoritmos de exploración de grafos más utilizados actualmente: el A* y su modificación para entornos dinámicos, el D*, con pequeñas modificaciones que nos han permitido adaptarlos sin problemas al caso particular de la representación del mapa de búsqueda mediante campos de potenciales.

Los resultados demuestran que, en entornos estáticos, no existen diferencias de rendimiento entre los dos algoritmos desarrollados. Otros de los datos obtenidos en las pruebas son los siguientes:

- Los algoritmos de búsqueda tienen un rendimiento más alto si en el calculo de la distancia a la configuración de destino se emplea la distancia manhattan en lugar de la distancia euclidea.
- Lo mismo ocurre si se utiliza un montículo binario para almacenar la lista de abiertos en lugar de una lista ordenada.
- El tamaño del campo de repulsión que aleja al robot de los obstáculos no debe ser excesivamente grande, ya que el rendimiento de los algoritmos disminuye a medida que crece dicho campo de repulsión.

Los resultados indican que, pese a que la información aportada por el campo de potenciales no es mucha, los algoritmos de búsqueda implementados nos permiten obtener el camino óptimo entre dos configuraciones de manera rápida.

Estos algoritmos se han usado como base para la implementación de una herramienta visual que nos permite estudiar los algoritmos desarrollados en distintas situaciones, ya que nos permite variar distintos parámetros de los algoritmos de búsqueda. La herramienta implementa el caso particular de un robot planar articulado.

También se ha desarrollado una herramienta para la generación de resultados de pruebas, capaz de generar distintos mapas y calcular distintos caminos de forma aleatoria.

Además, los algoritmos de búsqueda y la representación del entorno del robot desarrollados son fácilmente modificables para entornos con más de dos dimen-

siones, lo que nos puede permitir utilizarlos con robot articulados de más de dos grados de libertad. También pueden ser usados para otro tipo de robots.

Bibliografía

[AAM93] John J. Fox Anthony A. Maciejewski. Path planning and the topology of configuration space. *IEEE Transactions on Robotics and Automation*, 9(4), 1993.

[AB04] Jonathan Schaeffer Adi Botea, Martin Muller. Near optimal hierarchical path-finding. *Journal of Game Development*, 2004.

[BC97] V. Moreno B. Curto. Mathematical formalism to perform a fast evaluation of the configuration space. *Proc. of the IEEE Intern. Symp. on Computational Intelligence in Rob. and Autom.*, 1997.

[BDRT03] S. Borovkova, H. Dehling, J. Renkema, and H. Tulleken. A potential-field approach to financial time series modelling. *Comput. Econ.*, 22(2-3):139–161, 2003.

[BLR03] Anindya Basu, Alvin Lin, and S. Ramanathan. Routing using potentials: a dynamic traffic-aware routing algorithm. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 37–48. ACM Press, 2003.

[Che96] Danny Z. Chen. Developing algorithms and software for geometric path planning problems. *ACM Comput. Surv.*, 28(4es):18, 1996.

[EITE04] Edson Prestes E Silva Jr., Marco A. P. Idiart, Marcelo Trevisan, and Paulo M. Engel. Autonomous learning architecture for environmental mapping. *J. Intell. Robotics Syst.*, 39(3):243–263, 2004.

[Eln02] A. Elnagar. A heuristic approach for local path planning in 3d environments. *Robotica*, 20(3):281–290, 2002.

[GC02] S. S. Ge and Y. J. Cui. Dynamic motion planning for mobile robots using potential field method. *Auton. Robots*, 13(3):207–222, 2002.

[GL02] Policarpo Fabio Gilliard Lopes. Real-time bsp-based path planning with a*. *Brazilian Symposium on Computer Graphics and Image Processing*, 2002.

[Gou84] L. Gouzènes. Strategies for solving collision-free trajectories problems for mobile and manipulator robots. *International Journal of Robotics Research*, 3, 1984.

[HA88] Yong Koo Hwang and Narendra Ahuja. *Robot path planning using potential field representation*. PhD thesis, 1988.

[HKL01] R. V. Helgason, J. L. Kennington, and K. R. Lewis. Cruise missile mission planning: A heuristic algorithm for automatic path generation. *Journal of Heuristics*, 7(5):473–494, 2001.

[JB91] J. C. Latombe J. Barraquand. Robot motion planning: a distributed representation approach. *International Journal of Robotics Research*, 10, 1991.

[Kav95] L. E. Kavraki. Computation of configuration-space obstacles using de fast fourier transform. *IEEE Transactions on Robotics and Automation*, 5, 1995.

[Kha86] O. Khatib. Real-time obstacle avoidance for manipulators and movile robots. *International Journal of Robotics Research*, 5, 1986.

[KLW89] Peter Kacandes, Achim Langen, and Hans-Jergen Warnecke. A combined generalized potential fields/dynamic path planning approach to collision avoidance for a mobile autonomous robot operating in a constrained environment. In *Intelligent Autonomous Systems 2, An International Conference*, pages 145–154. IOS Press, 1989.

[Kod87] D.E. Koditscheck. Exact robot navigation by means of potential functions: Some topological considerations. *Proceedings of the IEEE International Conference on Robotics and Automation*, NC, 1987.

[LB98] Z. X. Li and T. D. Bui. Robot path planning using fluid model. *J. Intell. Robotics Syst.*, 21(1):29–50, 1998.

[Lee95] Bum Hee Lee. An analytic approach to moving obstacle avoidance using an artificial potential field. In *Proceedings of the International Conference on Intelligent Robots and Systems-Volume 2*, page 2482. IEEE Computer Society, 1995.

[LG86] R. Seidel L. Guibas. Computing convolution by reciprocal search. *Proc. of the ACM Symp. Computational Geometry*, 1986.

[LL00] C. Louste and A. Liegeois. Near optimal robust path planning for mobile robots: the viscous fluid method with friction. *J. Intell. Robotics Syst.*, 27(1-2):99–112, 2000.

[LP83] T. Lozano-Pérez. Spatial planning: a configuration space approach. *IEEE Transactions on computers*, C-32, Febrero 1983.

[MOL+03] P. Melchior, B. Orsoni, O. Lavialle, A. Poty, and A. Oustaloup. Consideration of obstacle danger level in path planning using a* and fast-marching optimisation: comparative study. *Signal Process.*, 83(11):2387–2396, 2003.

[NA97] Andreas C. Nearchou and Nikos A. Aspragathos. A genetic path planning algorithm for redundant articulated robots. *Robotica*, 15(2):213–224, 1997.

[OOML03] A. Oustaloup, B. Orsoni, P. Melchior, and H. Linares. Path planning by fractional differentiation. *Robotica*, 21(1):59–69, 2003.

[PK98] Kimmo Pulakka and Veli Kujanpaa. Rough level path planning method for a robot using sofm neural network. *Robotica*, 16(4):415–423, 1998.

[Rod97] Francisco Javier Blanco Rodríguez. *Un nuevo enfoque en la planificación de caminos libres de colisiones para estructuras robóticas.* PhD thesis, Universidad de Salamanca, 1997.

[Ste94] A. Stentz. Optimal and efficient path planning for partially-known environments. *Carneige Mellon University*, PA 15213, 1994.

[Ste95] A. Stentz. The focussed d* algorithm for real-time replanning. *Carneige Mellon University*, PA 15213, 1995.

[Tro96] Karen Irene Trovato. *A* Planning in Discrete Configuration Spaces of Autonomous Systems.* PhD thesis, Universidad de Amsterdam, 1996.

[Udu77] S. Udupa. *Collision Detection and Avoidance in Computer Controlled Manipulators.* PhD thesis, California Institute of Technology, 1977.

[WHL+04] Yongji Wang, Liu Han, Mingshu Li, Qing Wang, Jinhui Zhou, and Matthew Cartmel. A real-time path planning approach without the computation of cspace obstacles. *Robotica*, 22(2):173–187, 2004.

[WJ01] M. Williams and D. I. Jones. A rapid method for planning paths in three dimensions for a small aerial robot. *Robotica*, 19(2):125–135, 2001.

[WS00] Benedict Wong and Minas Spetsakis. Scene reconstruction and robot navigation using dynamic fields. *Auton. Robots*, 8(1):71–86, 2000.

[WSN91] M. S. Branicky W. S.Ñewman. Real-time configuration space transforms for obstacle avoidance. *The International Journal of Robotics Research*, 10, 1991.

[YK95] M. Yachida Y. Kitamura, T. Tanaka. 3-d path planning in a dynamic environment using an octree and an artificial potential field. In *Proceedings of the International Conference on Intelligent Robots and Systems-Volume 2*, page 2474. IEEE Computer Society, 1995.

[YM00] Simon X. Yang and Max Meng. Real-time collision-free path planning of robot manipulators using neural network approaches. *Auton. Robots*, 9(1):27–39, 2000.

[ZV97] Yanjun Zhang and Kimon P. Valavanis. A 3-d potential panel method for robot motion planning. *Robotica*, 15(4):421–434, 1997.