UNIVERSIDAD DE SALAMANCA

Departamento de Informática y Automática

Facultad de Ciencias

Cálculo del C-espacio de un robot móvil: convolución jerárquica

Francisco Javier Blanco Rodríguez

Abril 2003

Dr. **Vidal Moreno Rodilla**, Prof. Titular de Ingeniería de Sistemas y Automática de la Universidad de Salamanca y Dr.ª **Belén Curto Diego**, Prof.ª 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 del C-espacio de un robot móvil: convolución jerárquica** y presentada por Don **Francisco Javier Blanco Rodríguez** para optar al grado de Doctor 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, 22 de Abril de 2003

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

Dr.ª **Belén Curto Diego**
Prof.ª Titular de Universidad
Área de Ingeniería de Sistemas y Automática
Universidad de Salamanca

A mi abuelo

Agradecimientos

En esta página quiero expresar mi agradecimiento a todas las personas que me han ayudado de alguna forma en la realización de este trabajo de investigación.

En primer lugar, deseo manifestar mi más sincera gratitud a Vidal y Belén, mis directores de tesis, por sus acertados consejos, ideas y comentarios. En definitiva, por la inestimable y generosa atención que me han prestado durante todos estos años, tanto en el terreno académico como personal.

También deseo dar las gracias a todas las personas del Departamento de Informática y Automática por generar ese ambiente tan agradable. En especial a Roberto, con quien he compartido las alegrías de ver cumplidos poco a poco los objetivos propuestos desde que empezamos juntos los cursos de doctorado, que siga siendo siempre así. Y como no, a todos aquellos con los que en algún momento he compartido despacho, por su paciencia.

A mis amigos, que entre todos han conseguido darme esas horas de desconexión necesarias, aún a riesgo de hacer de mi un hipocondriaco con supuestos conocimientos médicos.

Tampoco quiero olvidar a toda la gente de El Grove, en especial a Maruse-la, aunque no me enseñaran a andar físicamente, me han transmitido valiosas enseñanzas para caminar por el mundo desde la sencillez.

En último lugar, pero no por ello el menos importante, quiero dar las gracias a toda mi familia puesto que todos ellos han puesto su granito de arena. A mi madre y a "la tia Pili" por ser tan pesadas y estar siempre detrás de mi. A mi hermano Miguel Ángel por sus criticas constructivas. Y en especial a mi abuelo, quien me enseño las cosas más importantes de la vida.

Puede llegar el día en que la inteligencia humana sea definida como aquello no factible por las máquinas

---

Herman Kahn

Contenidos

Índice de figuras

1

Introducción

Si hacemos un breve repaso de los antecedentes de la Robótica se puede observar, que desde la antigüedad aparece un gran deseo de crear máquinas con un comportamiento similar al de los seres vivos, ya sean animales o personas. En la mayor parte de estos casos se correspondían con estatuas animadas con fines religiosos.

Es en el siglo XVIII cuando aparece el gran auge de los autómatas. En dicho siglo, se desarrollan gran cantidad de dispositivos que son capaces de escribir, dibujar o tocar un instrumento musical. Un claro ejemplo es el de Jacques de Vaucanson (1709-1782) con sus obras "El flautista", "El tamborilero" y "El pato" (figura 1.1(a)).

Un paso más adelante en este sentido se produce desde la ciencia ficción con los primeros usos de la palabra robot (derivada del vocablo checo *robota*, cuyo significado es "trabajo obligatorio") proveniente de la obra de teatro "R.U.R:

(a)

(b)

Figura 1.1: (a) El pato de Vaucanson hecho de cobre, que bebe, come, grazna, chapotea en el agua y digiere su comida como un pato real. (b) Robots de "La Guerra de las Galaxias"

Rossum's Universal Robots" (1920) del dramaturgo Karel Capek. En el mismo contexto aparece el concepto Robótica en las novelas de ciencia-ficción de Isaac Asimov (1942) (donde establece "Tres Leyes de la Robótica"), en las que se presentan los robots como unos ingenios tales que obedezcan órdenes de las personas, pero con un alto grado de autonomía. En esta línea se encuentra la famosa saga de "La Guerra de las Galaxias", en la que el planteamiento de robots totalmente autónomos es llevado a un gran extremo. Estos robots serán capaces de determinar qué es lo que deben hacer y cómo, así como los pasos necesarios para la consecución de tal fin. Se tratará, por tanto, de máquinas dotadas con un cierto grado de inteligencia, e incluso de sentimientos y conciencia.

En el mundo real, los primeros robots industriales, tal y como los conocemos actualmente, fueron los Unimate (1958), desarrollados por George Devol y Joe Engelberger a finales de la década de los cincuenta y principios de los sesenta. Desde entonces el funcionamiento de los robots industriales, en los sectores en los que se han adaptado de forma óptima, como la industria del automóvil o la de fabricación, está basado en la realización de una serie de tareas repetitivas (soldadura, paletización y montaje, entre otras) previamente programadas por un operador.

Así, en la actualidad la tendencia de la investigación en Robótica está enfocada hacia diferentes objetivos, como pueden ser: la construcción de nuevas estructuras robóticas con mejor capacidad de movimientos, la realización de mejores sistemas de control para realizar los movimientos con mayor precisión, o la consecución de robots con "inteligencia", de manera que puedan disponer de un grado suficiente de autonomía. Esta última tendencia está orientada a la consecución de sistemas que sean capaces de realizar una serie de trabajos de forma totalmente autónoma para que simplemente sea necesario ordenar una tarea a realizar y de manera "inteligente" el robot decida cuáles son las actividades que tiene que realizar para alcanzar la consecución de ésta.

Los robots autónomos son especialmente interesantes para diversas aplicaciones como la agricultura, la construcción, la ayuda a discapacitados, la fabricación, la exploración espacial y submarina, etc. Si se profundiza un poco más detenidamente en el problema de los robots autónomos, se puede observar que es excesivamente complejo, en contraste con el hecho de que un ser humano lo resuelve de forma automática, sin tener que prestar atención a los pasos que debe seguir para su consecución. Así, cuando un robot autónomo realiza algún trabajo ejecutando diferentes secuencias de movimientos en un espacio de trabajo ocupado por objetos, un encadenamiento de tareas evoluciona recabando datos, procesándolos y produciendo resultados que sirven de nuevos datos de entrada. Además, en la resolución de dicho problema están implicadas numerosas técnicas procedentes de diferentes disciplinas como la Inteligencia Artificial, la Ciencia de la Computación, el Control Automático, la Tecnología de los Computadores y la Ingeniería Mecánica, entre otras.

1.1 Estructura de tareas en Robótica

Para llevar a cabo la emulación de las actividades ([LGF86]) de un ser humano se necesita desarrollar tecnologías como el razonamiento automatizado, la percepción y el control. Un diagrama que va a mostrar las relaciones entre las diferentes tareas a realizar es el de la figura 1.2.

En primer lugar, es necesario un **procesador de lenguaje**, de manera que las órdenes puedan ser interpretadas y se determine cuál es la tarea a realizar.

Un segundo paso consiste en dividir la tarea en una secuencia de operaciones más sencillas que lleven a la consecución de la operación compleja. De este trabajo se encarga el **planificador de tareas**.

Un ejemplo lo constituiría un robot móvil encargado de repartir y recoger la correspondencia en un edificio de oficinas. La entrada al planificador de tareas sería la especificación del objetivo en lenguaje natural: "repartir este conjunto de cartas". A su salida la tarea se habría dividido en las siguientes operaciones:

1. Clasificar las cartas,
2. Determinar los puntos de destino de cada una de ellas,
3. Generar el itinerario a seguir para llegar a todos los puntos y recorrer el menor espacio posible,
4. Recorrer el itinerario, evitando colisiones tanto con los obstáculos fijos como los móviles, y dejando en el lugar especificado cada carta.

La operación 4 se subdividirá a su vez en pasos más sencillos para cada lugar donde deba ser depositada una carta (ir a una posición determinada y depositar correspondencia que corresponda).

Sin embargo, estas operaciones no se van a poder realizar si no se tiene un conocimiento del entorno donde se realiza la operación. Por ejemplo, en la operación de recorrer el itinerario, para realizar el movimiento es necesario conocer cómo están situados los demás elementos del entorno para que no se produzcan choques. Es decir, también va a ser necesario disponer de algún sistema de **percepción** que indique cómo está dispuesto el escenario en el que se va a realizar el movimiento. Esta percepción se puede realizar mediante gran variedad de sensores ([KCN89]) como pueden ser los de visión, láser o sonar. A partir de la información recogida por ellos, se puede llegar a disponer de una descripción del entorno, en concreto de los puntos del espacio de trabajo del robot ocupados por objetos.

Una vez que ya se dispone de la información sobre el medio será necesario encontrar la secuencia de las diferentes posiciones (camino) que deben alcanzarse sin que se produzcan choques con los diferentes elementos del entorno, y por supuesto, de manera que se alcance el lugar objetivo. Esta funcionalidad es realizada por el **planificador de movimientos**.

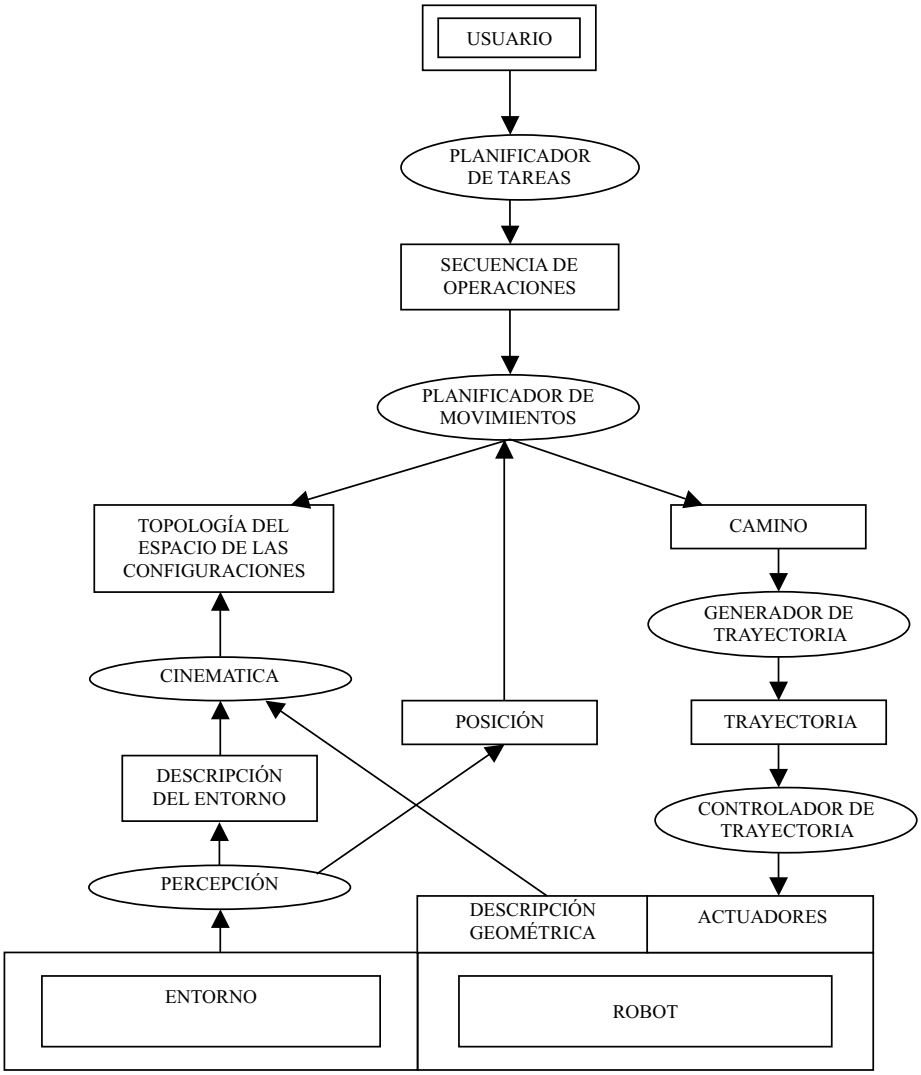


Figura 1.2: Esquema de las actividades necesarias para conseguir un robot autónomo

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

A la hora de realizar la planificación es fundamental saber dónde se encuentra el robot con respecto al entorno en cada momento. En el caso de los robots articulados no presenta ningún problema, puesto que cada articulación contiene un sensor para determinar su posición con gran precisión. Sin embargo, para los robots móviles el proceso es más complicado puesto que no se puede conocer la posición directamente midiendo con algún sensor incorporado al robot. Para conocer su posición se emplean técnicas de posicionamiento, como la odometría que consiste en determinar la posición conociendo la posición inicial y los movimientos realizados. Esta técnica no es suficiente debido a los deslizamientos no medibles en las ruedas del robot, y se complementa con otras técnicas para identificar el entorno en cada instante teniendo en cuenta alguna información previa.

Si nos centramos en el proceso de planificar los movimientos, es evidente que si se trabaja directamente con la descripción del entorno obtenida, será excesivamente complejo examinar en cada momento si se produce algún tipo de colisión de cualquiera de los elementos que forman el robot con cualquiera de los objetos que se encuentran en el entorno. En algunas referencias bibliográficas ([Her86b], [Kha86], [FT87] se trabaja con este planteamiento, y [Mor96]), pero, al intentar conseguir la simplificación de los algoritmos y, asimismo, de la velocidad en los cálculos, tienden a realizar grandes aproximaciones en las características geométricas de los elementos del robot y los objetos del entorno de forma poco realista.

Una solución muy extendida consiste en realizar un paso intermedio. Para ello, se hace uso de la **cinemática** del robot, de tal modo que se determina un conjunto de parámetros (configuración) que definen la posición y orientación de todos los puntos del robot con respecto un sistema de referencia fijo. Por

supuesto, el conjunto de parámetros está ligado con la geometría del robot ([Cra89]).

De esta forma, se realiza una proyección de la representación del entorno sobre un espacio característico para cada robot, de manera que la representación del robot en este espacio sea un solo punto. Así, cuando sea necesario comprobar las colisiones con objetos del entorno será mucho más sencillo, pues se deben examinar las colisiones únicamente con el punto que representa el robot. Este espacio característico para cada robot fue introducido por Lozano-Pérez [Loz83] en 1983 denominándolo "espacio de las configuraciones" o C-espacio. Sin embargo, presenta como principal inconveniente la necesidad de representar los obstáculos en el C-espacio. Este trabajo se va a centrar en este ámbito: en la proyección de la representación del espacio de trabajo en el C-espacio. El resultado obtenido por el planificador de movimientos es un camino geométrico que especifica la secuencia de configuraciones por las que el robot debe pasar para realizar la tarea especificada y alcanzar la configuración final.

La labor fundamental del controlador en tiempo real es lograr que el robot realice esta secuencia de movimientos, es decir, siga el camino geométrico 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 ([SV89]). 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 o durante 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 descripción de 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 las diferentes perturbaciones y a la 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.

1.2 Espacio de las configuraciones

Como se comentó anteriormente, se denomina espacio de las configuraciones al conjunto de todas las posibles configuraciones en que puede posicionarse un robot. Este concepto, que es esencialmente una herramienta de representación, fue introducido desde la Mecánica en la Robótica por Lozano-Pérez, y ocupará un papel fundamental en la descripción de este trabajo de investigación. Cada punto del C-espacio es una tupla de una cierta dimensión, donde se especifican los valores para los parámetros que se corresponden con los grados de libertad del robot. La posición y orientación del robot en su espacio de trabajo quedan completamente determinadas si se especifican los valores para los grados de libertad del robot.

Por ejemplo, supóngase un robot móvil que puede moverse y girar libremente sobre un plano con unas dimensiones $[0, a] \times [0, b]$ como se muestra en la figura 1.3. Por tanto, para describir una configuración del robot, y así todos los puntos que lo componen, es necesario fijar la posición de uno de los puntos del robot y la orientación de este respecto a un sistema de referencia fijo. La configuración del robot vendrá dada por una tupla de dimensión tres (x,y,θ), y el espacio de las configuraciones estaría formado por todas los posibles configuraciones, donde x e y vienen determinados por las dimensiones del espacio y θ tomará valores en el intervalo $[0, 2\pi]$.

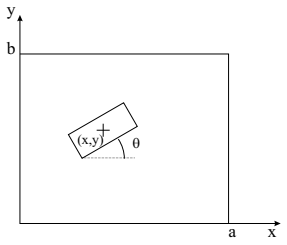


Figura 1.3: Robot móvil

Al lograr reducir la descripción del robot a un punto se produce una simpli-

ficación considerable del problema. Algunas de las tareas que se benefician de esta simplificación son la planificación y control de movimientos y la generación de trayectorias. Sin embargo, aparece el problema de obtener el conjunto de configuraciones prohibidas (C-obstaculos) o, lo que es lo mismo, las configuraciones que producen alguna colisión ya sea entre elementos del mismo robot o con obstáculos en el espacio de trabajo.

La gran diferencia de realizar la planificación de movimientos en el espacio de trabajo o en el de las configuraciones radica en que en el espacio de trabajo es necesario planificar los movimientos de todos y cada uno de los puntos del robot, mientras que en el C-espacio solamente es necesario considerar la planificación de un único punto. Es decir, al realizar la planificación en el C-espacio cada una de las configuraciones por las que ha de pasar el robot se corresponden con un punto del C-espacio, por lo que es una tarea mucho más sencilla.

Además, las restricciones cinemáticas se pueden trasladar directamente al C-espacio para realizar la planificación de movimientos, considerando únicamente transiciones con aquellas configuraciones que se pueden alcanzar desde una configuración dada. Por ejemplo, un robot móvil con giro diferencial puede rotar sobre su eje sin moverse de posición pero no puede desplazarse lateralmente. Así, el conjunto de configuraciones accesibles serán todas aquellas orientaciones en la posición actual, y aquellas que en que se produzca un avance en la dirección frontal del robot.

En cuanto a la generación de trayectorias, el hecho de conocer previamente las configuraciones que producen colisión o no, puede facilitar el desarrollo de los algoritmos. Así, por ejemplo, se puede generar el perfil de velocidad teniendo en cuenta la distancia desde las configuraciones que constituyen el movimiento a las configuraciones que producen colisión.

Por lo tanto, el tiempo que conlleva el cálculo del C-espacio tiene gran importancia puesto que es una información que será utilizada en las siguientes tareas. Está claro que el tiempo de cálculo va a estar asociado a la precisión con la que se realicen los cálculos. De tal manera que, representaciones aproximadas del robot y de los objetos requerirán menos tiempo de cálculo que si se utilizan representaciones más realistas.

La precisión no solamente va a afectar al tiempo de cálculo, sino que también

afectará a las necesidades de almacenamiento para los datos de origen (espacio de trabajo y robot), el resultado (C-espacio) y todos los cálculos intermedios que sean necesario realizar. Podemos comprobarlo con el ejemplo de un robot móvil que se mueve por un entorno de $100 \times 100 m^2$. Si para evitar colisiones se requiere trabajar con una precisión de al menos $10cm$ en las coordenadas espaciales, y de $0,5°$en la coordenada angular, al hacer una representación del C-espacio de forma matricial se obtendría una matriz tridimensional con un tamaño de $1.000 \times 1.000 \times 720 = 7,2 \times 10^8$ elementos. Si cada elemento se representa mediante 1 byte, se necesitarían aproximadamente 700 Mbytes.

Así, la elección de la estructura de datos para la representación de la información va a marcar la cantidad de memoria utilizada en este cálculo. Aunque actualmente puede parecer de poca importancia debido a la posibilidad de disponer de elevada memoria en los ordenadores, en algunos casos el requerimiento puede llegar a ser prohibitivo (supóngase que en el caso anterior se necesitara trabajar con una precisión de $1cm$).

Entre los diferentes modos de representación se encuentran:

- Geométricos. Utilizan figuras geométricas para representar el espacio de trabajo y el C-espacio, por ejemplo figuras poligonales o poliédricas. Nos va a permitir alcanzar una gran exactitud con poco coste de almacenamiento. Presenta el inconveniente de que para casos reales, sin realizar simplificaciones, las representaciones de los objetos van a ser extremadamente complejas, y es difícil establecer relaciones espaciales entre ellos. Además, los algoritmos para calcular el C-espacio usando una representación geométrica son dependientes del número de vértices, por lo que en situaciones reales el tiempo de cálculo puede aumentar considerablemente.

- Matriciales. Tanto el entorno de trabajo como el C-espacio se representan como matrices de dimensión N formadas por pixels, voxels[1], etc. según el caso. Con este tipo de representación los algoritmos sólo dependerán de la resolución dada, pero con el inconveniente de que cuanto mayor resolución se utilice, mayor será el requerimiento en memoria.

[1] Un voxel es la parte más pequeña distinguible con forma de cubo de una imagen tridimensional.

- Estructuras multiresolución. Con este tipo de estructuras se trata de reducir el coste excesivo de almacenamiento que ocasiona la representación matricial. Teniendo en cuenta que, normalmente, sólo es necesario tener una gran exactitud en zonas concretas, este tipo de estructuras varían su resolución dependiendo de la zona que se intenta representar para adaptarse a las variaciones existentes.

1.3 Estructuras de datos espaciales

Los datos espaciales consisten en puntos, líneas, regiones, rectángulos, o incluso datos de dimensiones mayores que son utilizados en gran cantidad de campos como pueden ser los Sistemas de Información Geográfica (GIS, donde su uso se ha extendido ampliamente), tratamiento de imágenes, representación tridimensional, monitorización medioambiental, planificación urbanística e, incluso astronomía. Por ejemplo, pueden representar ríos, carreteras, países, superficies de cultivo, estrellas o constelaciones, entre otros.

Aunque este tipo de datos pueden ser fácilmente almacenados en una base de datos mediante una parametrización, de manera que cada una de ellas ocupe un registro, solamente sería útil si se pretendiera realizar una simple recuperación de los datos. Sin embargo, cuando se pretenden obtener resultados más complejos relacionados con la distribución espacial entre ellos (como por ejemplo, proximidad, intersección,...), este modo de almacenamiento se vuelve inadecuado puesto que esta información no se encuentra almacenada en la base de datos. Por ejemplo, para representar un segmento en un plano sería necesario almacenar las posiciones de sus extremos, por lo que el segmento se representará por un punto (un registro en la base de datos) en un espacio de cuatro dimensiones. Hacer una consulta de cuál es el segmento más cercano a un punto determinado llevaría un gran coste asociado.

Por otro lado, almacenar toda la información espacial que relaciona los diferentes datos entre sí supondría un volumen de información considerable. Por tanto, es necesario el uso de algún tipo de ordenación de manera que esté basada en la posición espacial ocupada por el dato, de modo que, datos próximos espa-

cialmente también lo están en la base de datos. Estas técnicas de ordenación se conocen como "métodos de indexación espacial".

Gran parte de los métodos de indexación se fundamentan en estructuras de datos que están basadas en la ocupación espacial. Estas estructuras descomponen de forma iterativa el espacio en el que se representan los datos (i.e., espacio tridimensional para objetos tridimensionales)[2] en regiones más pequeñas llamadas "sectores de almacenamiento" (*buckets*). Por lo tanto las estructuras deben representar la jerarquía de las divisiones indicando como deben hacerse.

Estas estructuras se pueden clasificar en:

- Basadas en el mínimo rectángulo contenedor.
- Basadas en la descomposición en celdas disjuntas.

1.3.1 Mínimo rectángulo contenedor

En este tipo de estructuras se van encontrando mediante sucesivas divisiones los rectángulos que tienen menor tamaño y que además contienen a todos los datos. La jerarquía de las divisiones realizadas es almacenada en una estructura de árbol como puede ser un B-*tree* (*Balanced-Tree*) [BM70], que es un árbol de búsqueda balanceado. Así un B-*tree* de orden m cumple las siguientes características:

1. Todas las hojas[3] están en el mismo nivel.
2. Todos los nodos excepto el raíz están obligados a tener m nodos hijos como máximo y $m/2$ como mínimo.
3. El nodo raíz debe tener como máximo m hijos.

Un caso en el que se utiliza un B-*tree* para almacenar datos espaciales basado en el mínimo rectángulo contenedor es el R-*tree* [Gut84], en el cual las hojas del árbol tienen apuntadores a los datos que contienen. Cada nodo del árbol solamente puede contener un máximo de nodos hijos, y los datos deben estar completamente contenidos en una hoja. En la figura 1.4 se muestra un ejemplo de R-*tree* en el que los datos son rectángulos, y se almacenará en un B-*tree* de

[2] No en un espacio de dimensión mayor como ocurre con las bases de datos relacionales según se vio en el ejemplo anterior en el que se almacenan segmentos.

[3] Una hoja es un nodo terminal del árbol, es decir, que ya no presenta más dividiones.

orden 4. De este modo el nodo raíz tiene tres nodos hoja (A, B y C) que son los mínimos rectángulos contenedores. Estos nodos hoja contienen los datos espaciales, llegando a un máximo de 4 datos por hoja y un mínimo de 2.

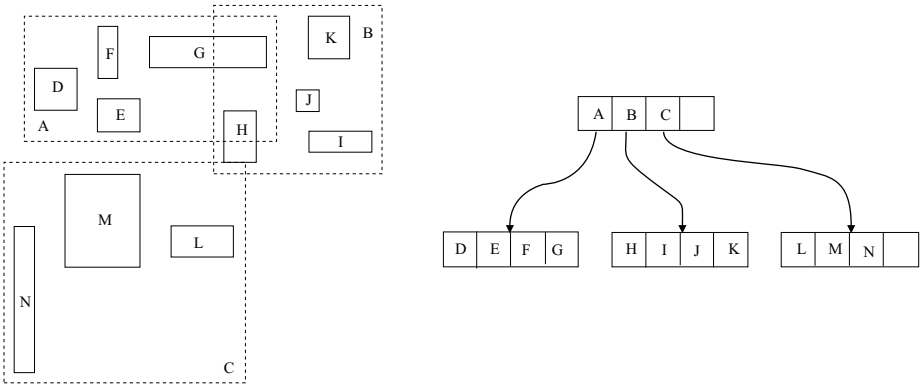


Figura 1.4: Rectángulos organizados en un R-*tree*.

1.3.2 Descomposición en celdas disjuntas

Todas las estructuras que se engloban dentro de esta denominación tienen en común que los objetos se descomponen en elementos más pequeños disjuntos, de modo que cada uno de éstos se asocie a una celda diferente. El precio que se paga es que para determinar el área ocupada por un objeto es necesario buscar todas las celdas que ocupa. Los estructuras de este tipo se diferencian en el grado de regularidad impuesto en la descomposición, así como en el modo en el que las celdas son agrupadas. Basándose en esta característica se distinguen las siguientes estructuras:

- Celdas Agrupadas en una estructura de tipo B-*tree*. Son muy similares a las estructuras del mínimo rectángulo contenedor, pero ahora los contenedores son disjuntos. Ejemplos de este tipo de estructuras son el R^+-*tree* [SRF87] y el *cell-tree*. En los primeros los contenedores son rectángulos y en los segundos son poliedros. En el caso del R^+-*tree*, la principal diferencia con el R-*tree*, del que proviene, es que ahora no existen superposiciones entre los nodos del árbol, por lo que algunos de los objetos se encuentran englobados por el conjunto de varias hojas y no sólo por una, como era en

el caso anterior. Tiene el inconveniente que la descomposición depende de los datos. Este inconveniente se intenta salvar con las estructuras que se comentarán a continuación, donde la descomposición es regular.

En la figura 1.5, tomando el mismo conjunto de datos que en el ejemplo del R-*tree*, se muestra cómo se organizan los datos en un R^+-*tree*. Se tienen 4 rectángulos contenedores disjuntos, que son las hojas del árbol. Al ser un B-*tree* de orden 4, cada una de las hojas contiene entre 2 y 4 objetos. La principal diferencia con el R-*tree* es que el objeto G está distribuido entre dos rectángulos contenedores: A y P.

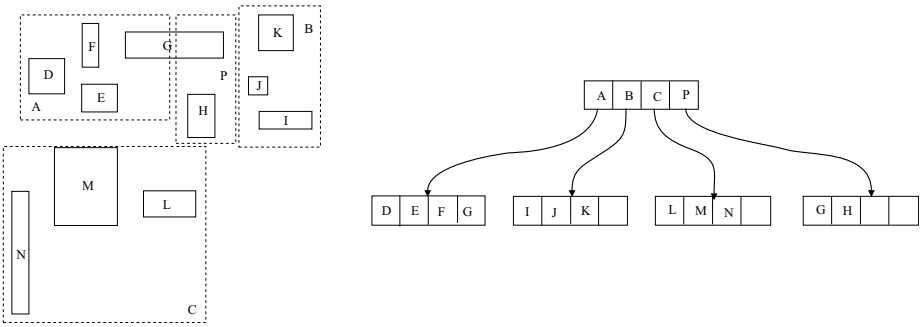


Figura 1.5: Rectángulos organizados en un R^+-*tree*

- Cuadrícula uniforme. Se distribuyen los datos de forma uniforme en bloques del mismo tamaño. Normalmente se utilizan estructuras de tipo matricial para su almacenamiento. Si se quiere un alto nivel de resolución necesitan de una gran cantidad de memoria necesaria para su almacenamiento. Un ejemplo aparece en la figura 1.6 (b).

- 2^k-*trees*. Adaptan la descomposición a la distribución de los datos, de tal manera que los bloques, en vez de tener el mismo tamaño, ocupan regiones de tamaño diferente. Se impone como restricción que la cuadrícula de cada lado sea una potencia de dos. Del mismo modo, las posiciones en las que pueden situarse los bloques también son restringidas. Este tipo de estructuras se conocen comúnmente como *quadtree* u *octree*, cuando hacen referencia a espacios bidimensionales o tridimensionales, respectivamente.

  El *quadtree* se basa en la división recursiva de un espacio bidimensional

en cuadrantes, de modo que si un cuadrante representa diferentes datos se subdividirá de nuevo hasta obtener que cada bloque represente únicamente un solo dato. Al dividirse cada bloque en cuatro partes iguales es común representarlo mediante un árbol de grado 4. En el caso de tener los datos dispersos de forma uniforme (el caso extremo es un tablero de ajedrez) el *quadtree* degenerará en una cuadricula uniforme.

En la figura 1.6 (c) se muestra un ejemplo de *quadtree* para representar una determinada región. En él se puede observar que hay bloques cuyos lados tienen dimensiones de 1, 2 y hasta 4 pixels. Además, cada uno de los bloques se encuentra completamente incluido en la región o es completamente externo. Las subdivisiones realizadas en el *quadtree* se pueden representar en forma de árbol según se muestra en la figura 1.6 (d), donde cada nodo hoja representa un bloque del *quadtree*. En este caso, como se han realizado tres divisiones, el árbol es de nivel 3.

Por ser este tipo de estructuras un punto central en este trabajo, se explicarán con mayor nivel de detalle en capítulos posteriores.

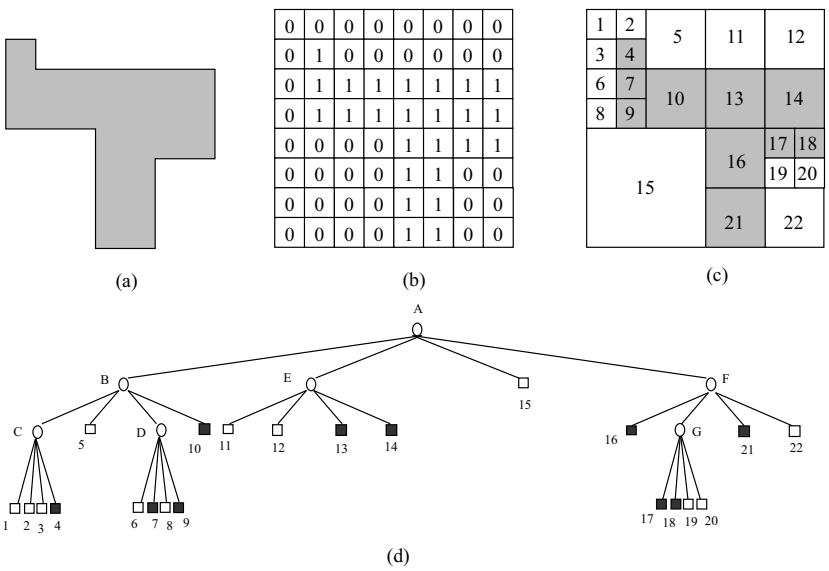


Figura 1.6: (a) Una región ejemplo, (b) representación en forma de cuadrícula uniforme, (c)(d) representación como *quadtree*

1.4 Antecedentes

En esta sección se va a hacer un repaso de los trabajos más destacados en los dos aspectos fundamentales en que se sustenta este trabajo de investigación. En primer lugar, se revisarán aquellos relacionados con el cálculo del espacio de las configuraciones, para pasar posteriormente a describir aquellos que se refieren al uso de estructuras de datos espaciales en los diferentes campos de la Robótica. De tal modo que, a continuación, se puedan poner de manifiesto los principales objetivos de este trabajo de tesis doctoral.

1.4.1 Métodos para el cálculo del C-espacio

En lo que respecta al cálculo del C-espacio existen dos tendencias claramente diferenciadas. La primera de ellas consiste en la aplicación de técnicas geométricas, mientras que la segunda consiste en la obtención del C-espacio de forma discreta.

En lo que respecta a las técnicas geométricas, en [Loz83] se propone un método para calcular la representación del límite de un C-obstáculo cuando los objetos son polígonos convexos. El algoritmo propuesto es bastante óptimo, pues calcula los vértices del C-obstáculo, que también es convexo, en un tiempo de cálculo $O(n_A n_B log(n_A n_B))$, siendo n_A y n_B el número de vértices del robot y del obstáculo, respectivamente. En [Sha87] se plantea un algoritmo con un tiempo de cálculo de $O(n_A^2 n_B^2 \log(n_A n_B))$ cuando el robot se traslada y no rota, y los obstáculos son polígonos no convexos. En [AB88] y en [Bro89] se describen métodos generales para polígonos convexos y no convexos, cuando el robot se desplaza y gira libremente, y se obtiene el límite del C-obstáculo en un tiempo $O(n_A^3 n_B^3)$ y $O(n_A^3 n_B^3 \log(n_A n_B))$, respectivamente. En cuanto a espacios de trabajo poliédricos, en [Loz83] se presenta una extensión de su algoritmo poligonal; en [GS86] se calcula la representación del límite del C-obstáculo en $O(n_A + n_B + n_A n_B)$; y el algoritmo propuesto en [AB88] construye el C-obstáculo en $O(n_A^3 n_B^3 \log(n_A n_B))$.

Todos los trabajos citados anteriormente solamente se refieren a robots móviles. También existen trabajos, aunque en menor cantidad, que tratan de encontrar el C-espacio para robots articulados en 2 ó 3 dimensiones. Entre

ellos, un grupo plantea el uso de las técnicas provenientes de los robots móviles, considerando a los articulados como un conjunto de móviles ([Gou84] [Fav86] [Loz87]). Otros, sin embargo, consideran al manipulador como un todo, de manera que intentan calcular las ecuaciones del límite de los obstáculos en el C-espacio. Consideran, por un lado, las ecuaciones algebraicas que definen los obstáculos y, por otro, las expresiones matemáticas que describen la cinemática del robot. A partir de las condiciones que se establecen cuando los elementos del robot intersectan con los obstáculos se calculan las ecuaciones explícitas del límite de los C-obstáculos ([GM89, DM90] [GM90] [Hwa90]).

En las técnicas que siguen la segunda tendencia, las que calculan el C-espacio de forma discreta, se utiliza una matriz binaria para representar los obstáculos en el C-espacio, donde una determinada celda, asociada a unos determinados valores discretos de los grados de libertad del robot, puede tomar dos valores: '1' si es una configuración prohibida y '0' si pertenece al espacio libre. Así, se consigue reducir de forma drástica el tiempo de cálculo que la etapa de planificación dedica a la detección de colisiones. A este respecto, algunos trabajos calculan la proyección de un conjunto de primitivas de objetos sencillos desde el espacio de trabajo al espacio de las configuraciones, y el resultado se almacena en matrices binarias. Basándose en las propiedades de la traslación, rotación y superposición, las transformaciones de estas primitivas se combinan para calcular matrices binarias que representan a objetos más complejos en el C-espacio ([NB91] [LP91]).

Kavraki [Kav95] propone un algoritmo para calcular una matriz binaria que representa al C-espacio de un móvil bidimensional como la convolución de una matriz binaria de los obstáculos en el espacio de trabajo y de otra del robot. Esta idea de ver la proyección al C-espacio como una convolución ya se proponía en [Bro89] para obtener la representación de los obstáculos en el espacio de las configuraciones mediante una convolución algebraica de un obstáculo y un robot móvil. Sin embargo, el principal aporte de Kavraki radica en la utilización de la Transformada Rápida de Fourier (FFT) para realizar la convolución de forma eficiente. Además, se obtiene como ventaja adicional la independencia del tiempo de cálculo con el número de obstáculos, con la forma de estos y con la del robot, ya que únicamente depende de la resolución con la que se discretiza.

En trabajos más recientes ([Cur98][The02]) se han establecido formalismos matemáticos para el uso eficiente de la FFT en el cálculo del C-espacio de forma discreta de manipuladores. Éstos se fundamentan en la elección adecuada de sistemas de coordenadas, de tal manera que la variación de los grados de libertad del robot articulado aparezcan como una convolución. Así, se llegan a obtener resultados aplicables a manipuladores redundantes, con bajos tiempos de computación. Aunque al ser discretos, cuando aumentan los grados de libertad los requerimientos de memoria aumentan considerablemente.

1.4.2 Uso de estructuras jerárquicas en Robótica

Aunque el uso de estructuras de datos espaciales se ha extendido a gran cantidad de áreas, este apartado se va a centrar en una revisión de las diversas aplicaciones que se han dado en las diferentes tareas involucradas en Robótica. En concreto, veremos las aplicaciones que han tenido los *quadtrees* y *octrees* que, al ser estructuras con grandes capacidades para la representación de regiones, son las que han tenido una mayor aceptación. Sin duda fue esta razón la que llevo a Nilsson [Nil69] a utilizar estructuras similares a los *quadtrees* para representar el espacio de trabajo del robot.

Una de las primeras tareas expuestas en el apartado 1.1, la percepción, está fuertemente relacionada con los campos de la visión artificial y el procesamiento de imágenes. Debido a los orígenes de estas estructuras de datos, existen gran cantidad de trabajos asociados a la visión artificial que hacen uso de este tipo de estructuras como, por ejemplo, [CSA88][LC94][Con84]. Por esta razón, en el campo de la percepción se han incorporado de una manera directa para realizar una descripción del espacio de trabajo del robot. Por ejemplo, ya en [HS85] se propone un procedimiento para reconstruir un espacio de trabajo y, en concreto, determinar la forma y tamaño de los objetos que se encuentran mediante imágenes tomadas por una cámara desde diferentes puntos de vista, utilizando una estructura de *octree* para almacenar la descripción del entorno.

En esta dirección existen trabajos mucho más recientes que hacen uso de sistemas LADAR[4] para la detección mucho más precisa de los objetos. En [BT94] se realiza una caracterización tridimensional por medio de *octrees* (aun-

[4] LAser Detection And Ranging

que el objetivo sea transformar esos *octrees* en una representación de superficies poligonales) de tanques de residuos peligrosos para la operación remota de un manipulador. En otros casos [HBM+02], la representación será en forma de *quadtree* (en dos dimensiones) al estar enfocada a la navegación de un vehículo autónomo.

Otra tarea en la que el uso de los *quadtrees* y *octrees* ha tenido una gran aceptación es la de planificación de movimientos, debido a la gran capacidad para compactar los datos del espacio de trabajo y, sobre todo, por la facilidad de acceder a esos datos. Los primeros trabajos en los que aparecen estructuras de datos similares al *quadtree* para realizar la planificación se deben a Brooks [BL83]. Pero es en [Fav84] donde se plantea la posibilidad de utilizar *octrees*. En trabajos como [Her86b][Her86a] se realiza una planificación partiendo de una representación del espacio de trabajo, ya sea en forma de *quadtree* u *octree*, aunque únicamente se centran en robots móviles. Un paso más adelante aparece en [FS89][KTKY95] donde se tiene en cuenta, además, un espacio de trabajo dinámico. También se han aplicado para la planificación de robots articulados [SH92], aunque solamente se utilizan como una representación del espacio de trabajo que simplifica el test de colisiones.

En la tarea de percepción se han utilizado estas estructuras pero no para representar el espacio de trabajo, sino para realizar una estimación de la posición que ocupa un robot móvil [BDFC98]. En este caso, un *octree* que varía continuamente en el tiempo representará la densidad de probabilidad de encontrar el robot en cada posición y orientación.

Otros trabajos integran la etapa de percepción con la de planificación [Zel92] [YG00]. En e[Zel92] se aplica a un robot móvil, de tal modo que según navega va detectando nuevos obstáculos que utiliza para la optimización de la longitud del camino al destino. En [YG00] su aplicación se dirige a la planificación de movimientos de un manipulador, de tal modo que se va explorando el espacio de trabajo, a medida que es necesario, para realizar los movimientos.

Hay que destacar que en ninguno de estos ejemplos se realiza una representación explícita de los obstáculos en el C-espacio, simplemente se realiza un test de colisiones en el espacio de trabajo.

En el mismo ámbito de la planificación se han propuesto diferentes estruc-

turas basadas en los *quadtrees* y *octrees* para mejorar diferentes aspectos. En [JG97] se propone una estructura denominada ODM (*Octree Distance Maps*) para simplificar la etapa de detección de colisiones. Otras estructuras propuestas [CSU97][YSSB98] son los "*Framed-Quadtrees*" para planificaciones en 2 dimensiones o los "*Framed-Octrees*" [CSU95] para el caso tridimensional.

Para la planificación en el C-espacio también se han utilizado representando el C-espacio en forma de *octrees* para realizar una planificación más rápida [KTKY95]. En trabajos más recientes [RI02] utiliza funciones harmónicas para planificar sobre una representación jerárquica del C-espacio, permitiendo así una planificación dinámica con los cambios en el espacio. Otra posibilidad es utilizar los *octrees* para representar diagramas de Voronoi en espacios de cualquier dimensión como se propone en [VO95].

Sin embargo, en lo referente al cálculo del C-espacio en forma de estructuras de datos espaciales (*quadtrees* y *octrees*) solamente se han encontrado los trabajos de Geem [Gee95] [Gee96] como referencias. En ellos se realiza el cálculo del C-espacio para manipuladores robóticos a partir de aproximaciones de sus elementos a superelipsoides. De modo que después se realiza una planificación mediante campos de potencial directamente en el C-espacio.

1.5 Objetivos de este trabajo

En los apartados anteriores ha quedado reflejada la gran importancia de realizar una representación de los obstáculos en el C-espacio. Como ha quedado expuesto, la carga computacional es elevada, y las técnicas que existen en la actualidad para realizar los cálculos exactos de forma discreta requieren una gran cantidad de memoria para alcanzar resoluciones realistas.

Dado que la representación en forma de *quadtree* u *octree* de cualquier espacio reduce las necesidades de almacenamiento, y la reducción sería más significativa si el espacio pudiera alcanzar dimensiones mayores. Por tanto, sería aconsejable el uso de esas estructuras para representar el C-espacio.

Un posible planteamiento constaría de dos etapas: en la primera se calcularía el C-espacio mediante uno de los métodos más óptimos propuestos (la convolución de funciones); y en la segunda se representaría el C-espacio calcu-

lado utilizando *quadtree* u *octrees*. Sin embargo, este planteamiento tiene como desventaja que aunque para almacenar el resultado se reducen las necesidades de memoria no se obtiene ningún beneficio en la etapa de evaluación, por lo que los requerimiento de memoria siguen limitando el proceso.

Así, el objetivo principal de este trabajo es proponer un método que permita el cálculo del C-espacio en forma de 2^k-*tree* directamente desde la representación del espacio de trabajo sin necesidad de pasar por estructuras de datos intermedias. De tal modo que, aprovechando de nuevo las características de estas estructuras de datos, se pueda ir refinando la precisión para evaluar los C-obstáculos en aquellos lugares que se requiera.

Puesto que la evaluación del C-espacio de forma discreta está basada en la convolución de funciones, el primer objetivo será la determinación de las herramientas matemáticas que permitan el desarrollo de un procedimiento general para realizar la convolución de funciones representadas mediante estructuras jerárquicas de datos.

La convolución mediante estructuras jerárquicas a la evaluación del C-espacio se aplicará a robots móviles, en los que la correspondencia entre espacio de trabajo y espacio de las configuraciones es mucho más clara. De este modo, se planteará un desarrollo incremental partiendo de un robot móvil sin posibilidad de giros y, una vez estén validados los datos, se ampliarán los resultados introduciendo la posibilidad de giro.

De igual modo, al ser un procedimiento propuesto en este trabajo de tesis doctoral se considera como objetivo imprescindible el estudio de sus características.Así, habrá que determinar su coste computacional así como las dependencias que puedan aparecer. También, será necesario comparar los resultados con los obtenidos utilizando técnicas anteriores.

1.6 Estructura de la memoria

Una vez establecidos los puntos de partida en los que se enmarca el presente trabajo a continuación se desglosan los capítulos en los que se ha organizado la memoria.

En el capítulo segundo se definen los conceptos referentes a la evaluación

el espacio de las configuraciones. También, se hace un análisis detallado de las técnicas que se han desarrollado para dicha evaluación. En particular, se prestará especial atención a aquellas técnicas que hacen que uso de representaciones discretas.

Siguiendo con la definición de conceptos, en el capítulo tercero se describen aquellos aspectos relacionados con las estructuras jerárquicas de datos. De entre ellos, se hará énfasis en los *quadtree* y *octree* por la especial relevancia que tienen en este trabajo.

En el capítulo cuarto se propone un formalismo matemático para realizar la convolución jerárquica, siempre con la mirada puesta en el objetivo último, que es la evaluación del espacio de las configuraciones.

Una vez expuestas las bases de la convolución jerárquica, en el capítulo quinto, se presenta el algoritmo propuesto para la evaluación del C-espacio de un robot móvil sin giros de forma jerárquica. La aplicación para este tipo de robot servirá para asentar las bases de la convolución jerárquica. Además, se exponen las particularidades que es necesario tener en cuenta para un comportamiento eficiente del algoritmo.

Una vez que se ha aplicado la convolución jerárquica en la evaluación de un C-espacio bidimensional, en el capítulo sexto se mostrará su aplicación para un robot móvil con la posibilidad de girar. En este capítulo se definirá una nueva estructura de datos necesaria, la cual hemos denominado árbol multigrado-2^k.

En el siguiente capítulo se realiza una descripción de las técnicas y herramientas más relevantes en el desarrollo de este trabajo.

Para la aplicación de los algoritmos propuestos, en el capítulo octavo se propone un procedimiento para la evaluación del C-espacio donde se determina cómo utilizar los algoritmos para obtener resultados óptimos. También, en este capítulo se analizan las principales características de los algoritmos, y se compararán sus bondades e inconvenientes con los ya existentes.

Para terminar en el capítulo noveno se exponen la principales conclusiones a las que ha llevado el desarrollo de este trabajo, así como las futuras líneas de investigación.

2

Espacio de las Configuraciones

Como se ha comentado previamente, la idea fundamental del espacio de las configuraciones consiste en la representación el robot como un punto en un espacio apropiado. Pero para poder trabajar en este espacio es necesario proyectar los obstáculos en este espacio. Esta proyección simplifica de forma drástica las tareas de planificar y controlar los movimientos de un robot. Por un lado, el problema de planificación se reduce a determinar la secuencia los movimientos de un punto y, además, se tienen de forma explícita las configuraciones que producen colisiones. Por otro lado, es bastante común identificar un punto del espacio de las configuraciones con la referencia de los controladores que se ocupan de gobernar los movimientos del robot. En este contexto la proyección de los obstáculos en este espacio se puede considerar como restricciones en el problema de control.

Esta proyección depende tanto de los obstáculos como del tipo de robot con-

siderado. En cuanto a los primeros, no se impondrá ninguna limitación debida a su forma geométrica, número de vértices, etc. Respecto al segundo, una clasificación muy genérica es aquella que diferencia entre robots que pueden moverse libremente en su espacio de trabajo, denominados móviles, y aquellos que tienen un punto fijo. Otra los clasifica en robots articulados o no-articulados, dependiendo de si su estructura mecánica está formada por uno o varios elementos que se mueven. Tomando como base estos criterios de clasificación existirían cuatro tipos de estructuras robóticas. Los robots más implantados en las aplicaciones más usuales son los móviles constituidos por un único elemento y los articulados con un punto fijo, aunque también existen aplicaciones muy específicas donde se utilizan robots móviles articulados.

En este apartado se va a presentar un conjunto de nociones necesarias para definir el concepto de espacio de las configuraciones de un robot y la representación de los obstáculos en este espacio. Aunque los conceptos de configuración y espacio de las configuraciones ya fueron presentados en la introducción, aquí se presentarán de una forma más rigurosa incluyendo ejemplos que ayudarán a comprenderlos.

Inicialmente estas definiciones se enfocarán hacia objetos rígidos que se mueven libremente. Posteriormente, se van a extender dichas definiciones para robots articulados con una base fija. No obstante, no se profundizará demasiado en este segundo aspecto, ya que nuestro trabajo se centrará principalmente en los robots móviles.

2.1 Objeto rígido móvil

En este apartado se explicarán los conceptos de C-espacio y C-obstáculo para un objeto rígido móvil donde se pueden entender con mayor claridad. Una vez sentadas las bases, en el apartado siguiente se abordarán los robots articulados.

2.1.1 Espacio de las configuraciones

Sea **A** un objeto rígido - el robot - que se mueve en un espacio de trabajo W (figura 2.1). Se representa W como un espacio Euclídeo R^n de dimensión n, donde $n = 2$ ó 3. En él se define un sistema de referencia fijo denominado

F_W. Se representa **A** en una posición y orientación de referencia como un subconjunto compacto de R^n. En **A** se establece un sistema de referencia, F_A, que se mueve con él, de tal forma que cada punto en el robot tiene unas coordenadas fijas respecto a F_A. Los orígenes de ambos sistemas F_W y F_A se denotan por O_W y O_A, respectivamente. O_A se denomina el punto de referencia de A.

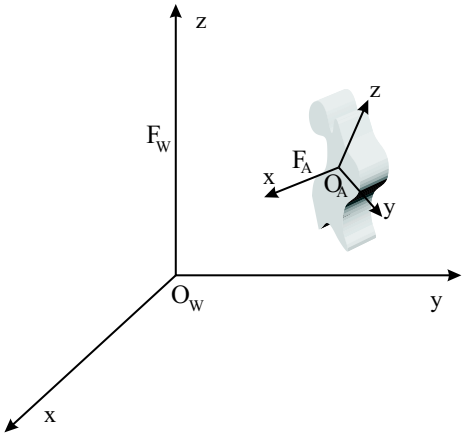


Figura 2.1: Robot móvil en $W = R^3$

Definición 2.1
Una configuración q de **A** es una especificación de la posición y la orientación de F_A con respecto a F_W.

La configuración de referencia de **A**, que se denota por 0, con respecto a la que se especifican las demás es una única configuración seleccionada de forma arbitraria.

Definición 2.2
El espacio de las configuraciones de **A** es el espacio C de todas las posibles configuraciones de **A**.

El espacio de las configuraciones es intrínsecamente independiente de la elección de los sistemas de referencia F_A y F_W. Solamente la representación de C depende de la elección de estos sistemas.

El subconjunto de W ocupado por $\mathbf{A}$ en una configuración q se denota por $\mathbf{A}_{(q)}$ o $\mathbf{A}(q)$. El robot en su configuración de referencia se denota por $\mathbf{A}_{(0)}$ o $\mathbf{A}(0)$. Cuando $\mathbf{A}$ está en la configuración q, un punto a de $\mathbf{A}_{(q)}$ se denota por $a(q)$ en W. Así, para dos configuraciones q y q' cualesquiera, $a(q)$ y $a(q')$ representan el mismo punto del robot $\mathbf{A}$, pero al estar éste en diferentes configuraciones los dos puntos no tienen por qué coincidir en W.

Estas definiciones son válidas de forma general para cualquier tipo de robot, aunque son directamente aplicables para un objeto rígido que se mueve libremente en su espacio de trabajo. Para particularizarlas a robots articulados con un punto fijo es necesario realizar algunas consideraciones adicionales, lo que se efectuará en secciones posteriores.

2.1.2 Obstáculos en el C-espacio

En general, W contiene obstáculos fijos que son regiones de R^n. $\mathbf{B}$ denota tanto al obstáculo "físico" como al subconjunto de R^n que lo representa. En lo sucesivo se considerará que los obstáculos son rígidos y que están fijos en W. Por tanto, cada punto de $\mathbf{B}$ tiene una posición fija con respecto a F_W.

Definición 2.3
El obstáculo $\mathbf{B}$ en W se proyecta en C sobre la región

$$\mathbf{CB} = \{q \in C / \mathbf{A}(q) \cap \mathbf{B} \neq \emptyset\}$$

CB se denomina obstáculo en el espacio de las configuraciones o "C-obstáculo".

Esta es la definición propuesta por Lozano-Pérez [Loz83] y es admitida de forma unánime en la bibliografía. En concreto, significa que si $q \in \mathbf{CB}$ entonces $\mathbf{A}(q)$ intersecta con $\mathbf{B}$. Por tanto, un C-obstáculo representa el conjunto de configuraciones de $\mathbf{A}$ que producen colisión con $\mathbf{B}$. Los C-obstáculos son las restricciones en el movimiento del robot debidas a la presencia de obstáculos en el espacio de trabajo.

Definición 2.4

El conjunto

$$\mathbf{C_{free}} = \{q \in C / \mathbf{A}(q) \cap \mathbf{B} = \emptyset\}$$

se denomina espacio libre. Una configuración en $\mathbf{C_{free}}$ se denomina una configuración libre de colisiones.

2.1.3 Ejemplos

A continuación se ilustrarán con ejemplos sencillos las nociones y definiciones establecidas previamente, es decir, el C-espacio y el C-obstáculo. En concreto, se explicarán tres ejemplos de robots móviles sobre el plano, con tres formas geométricas diferentes. El espacio de trabajo se considerará ocupado por un único obstáculo para así poder mostrar claramente la proyección de cada obstáculo en el espacio de trabajo al C-espacio.

- Sea **A** un robot con forma de disco que se mueve libremente en $W = R^2$. En este caso, no es posible diferenciar entre dos configuraciones que solamente se diferencian en una rotación alrededor del centro del disco. O expresado de otra forma, el espacio ocupado por el robot en W es el mismo en ambas configuraciones. Por lo tanto, tomando el centro del sistema de referencia asociado al robot (O_A) en el centro del disco, el espacio de las configuraciones se puede expresar por el espacio R^2. Por lo que una configuración vendrá dada por la posición de O_A respecto a un sistema de referencia F_W fijo en W.

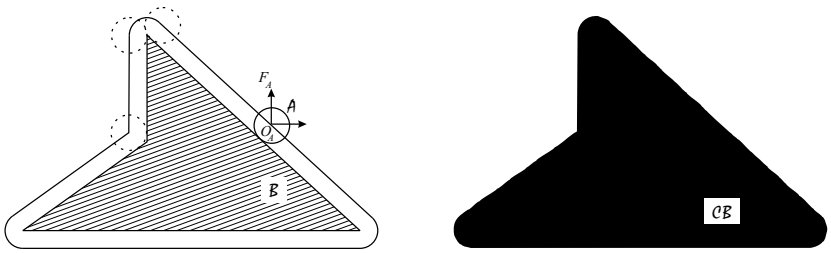


Figura 2.2: C-obstáculo para un disco móvil y un obstáculo poligonal

Sea ahora **B** un obstáculo poligonal como se muestra en la figura 2.2. Todas aquellas configuraciones pertenecientes al C-obstáculo **CB** debidas al polígono

B serán aquellas en las que se produzca intersección entre el robot **A** y **B**. De tal forma que el límite de **CB** es la curva seguida por el punto O_A cuando **A** da vueltas sobre el contorno de **B**. De tal forma que el C-obstáculo **CB** se puede obtener aumentando **CB** de forma isotrópica con el radio de **A**.

- Sea **A** un polígono convexo, en concreto un triángulo rectángulo (figura 2.3), que se mueve con una orientación fija (no puede rotar) en $W = R^2$. Al moverse con una orientación fija, cualquier sistema de referencia fijo a él siempre tendrá la misma orientación, por lo que solamente es necesario determinar la posición de su origen O_A respecto al sistema fijo en el espacio de trabajo. Por lo tanto, como O_A es un punto de R^2 el conjunto de todas las posiciones que puede tomar es $C = R^2$. Por otro lado, según se comentó, el espacio de las configuraciones es independiente de la elección de los sistemas de referencia, por lo que podemos situar el origen O_A en cualquier punto de **A**. En este caso se ha elegido uno de los vértices del triángulo. Aunque el C-espacio es independiente de la elección realizada, no sucede lo mismo con la representación de los obstáculos en él. Es decir, la elección del sistema de referencia F_A determina la proyección de los obstáculos en el C-espacio.

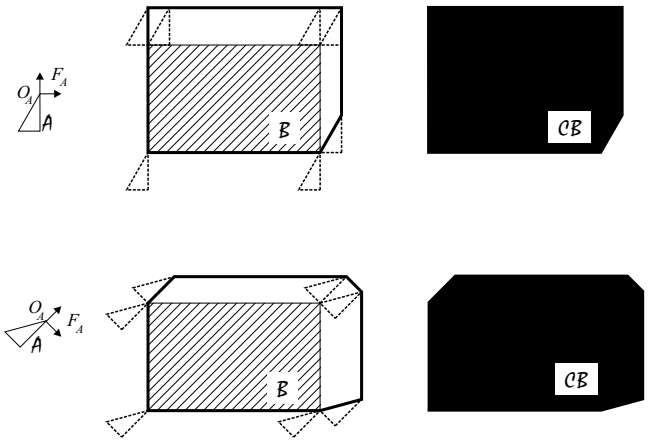


Figura 2.3: C-obstáculo para un triángulo móvil con orientación fija y un obstáculo poligonal

En el espacio de trabajo se encuentra un obstáculo **B** poligonal convexo, un rectángulo. De forma similar al ejemplo anterior, la frontera del C-obstáculo coincide con el recorrido que realiza el punto O_A cuando el robot **A** se mueve

en contacto con el contorno del obstáculo **B**, sin que los interiores de **A** y **B** se solapen. Así, en la figura 2.3 se muestran los C-obstáculos en R^2 obtenidos para dos orientaciones fijas de **A** distintas.

- Sea **A** un polígono convexo (figura 2.4) que se puede trasladar y rotar en $W = R^2$. Con la misma elección de F_A que la efectuada en el caso previo, una configuración estaría dada por la posición (x, y) y la orientación θ de F_A con respecto F_W. Por tanto, C sería un espacio de dimensión 3. La representación de C vendría dada parametrizando cada configuración q por $(x, y, \theta) \in R^2 \times [0, 2\pi]$.

Sea **B** un obstáculo poligonal convexo en el espacio de trabajo W. Para determinar el C-obstáculo se puede considerar el conjunto de planos paralelos, perpendiculares al eje θ. En cada uno de estos planos $\theta = constante$, puesto que la orientación permanece fija, la representación del C-obstáculo es similar a la del ejemplo anterior. Así, en la figura 2.4 se muestra el C-obstáculo **CB** como el conjunto de C-obstáculos para cada orientación del robot.

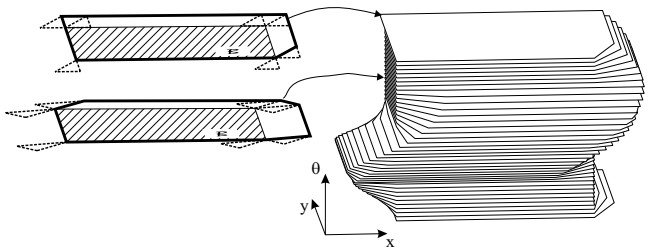


Figura 2.4: C-obstáculo para un triángulo móvil con orientación variable y un obstáculo poligonal

2.2 Robot articulado

Una vez sentadas las bases para un objeto rígido móvil vamos a explicar su generalización y aplicación sobre robots articulados.

2.2.1 Espacio de las configuraciones

Las definiciones presentadas previamente se pueden extender para robots formados por varios objetos rígidos, que se mueven, conectados por articulaciones mecánicas que imponen unas restricciones a su movimiento relativo. Un ejemplo

típico es un brazo manipulador, que consta de una secuencia de objetos rígidos conectados en una cadena mediante articulaciones.

Sea $\mathbf{A}$ un robot articulado formado por p objetos rígidos o *elementos* $\mathbf{A_1}, \cdots,$ $\mathbf{A_p}$ que se mueven. Dos objetos $\mathbf{A_i}$ y $\mathbf{A_j}$ cualesquiera podrían estar conectados por una articulación. Se supondrá, para simplificar, que la articulación es de revolución o prismática. Existen otros tipos de articulaciones, aunque éstas dos son las más típicas. Una articulación de revolución restringe el movimiento relativo de $\mathbf{A_i}$ y $\mathbf{A_j}$ a una rotación alrededor de un eje fijo con respecto a ambos elementos. Una articulación prismática es una conexión que limita el movimiento relativo a una traslación a lo largo de un eje fijo con respecto a ambos objetos.

Sea F_{A_i} el sistema de referencia unido a $\mathbf{A_i}$, $i \in [1, p]$. Por la Definición 2.1 en la página 25, la configuración de $\mathbf{A}$ es una especificación de la posición y la orientación de cada sistema de referencia F_{A_i}, $i \in [1, p]$, con respecto a F_W. Si los distintos objetos pudieran moverse de forma independiente en $W = R^n$, el espacio de las configuraciones de $\mathbf{A}$ sería

$$C' = \underbrace{(R^n \times SO(N)) \times \cdots \times (R^n \times SO(N))}_{p}$$

donde $SO(N)$ es el Grupo Ortogonal Especial de matrices $N \times N$ con columnas y filas ortonormales y determinante $+1$. Sin embargo, las distintas articulaciones imponen restricciones en las configuraciones posibles de C'. Estas restricciones seleccionan un subespacio C de C' de dimensión más pequeña, que es el espacio de las configuraciones de $\mathbf{A}$.

De este modo considerando un manipulador formado por p elementos tal que $\mathbf{A_j}$ se encuentra unido a $\mathbf{A_i}$, siendo $j > i$, se puede definir una configuración de $\mathbf{A_j}$ respecto a $\mathbf{A_i}$ especificando la posición y la orientación de F_{A_j} con respecto a F_{A_i}. Se supone, inicialmente, que no existen topes mecánicos en las articulaciones. Se denota con $C_j^{(i)}$ al C-espacio de $\mathbf{A_j}$ respecto a $\mathbf{A_i}$. Si dos objetos están conectados por una articulación de revolución se tiene que $C_j^{(i)} = S^1$, donde S^1 denota al círculo unidad en R^2. Si los dos elementos están unidos por una articulación prismática entonces se tiene que $C_j^{(i)} = R$. Por tanto, el espacio de las configuraciones de un robot articulado $\mathbf{A}$ formado por p elementos rígidos conectados por p_1 articulaciones de revolución y p_2 prismáticas ($p = p_1 + p_2$)

es:

$$C = \underbrace{S^1 \times \cdots \times S^1}_{p_1} \times \underbrace{R \times \cdots \times R}_{p_2}$$

Un problema práctico que se presenta es que la dimensión del espacio de las configuraciones aumenta con el número de articulaciones.

2.2.2 Obstáculos en el C-espacio

Existen dos tipos de C-obstáculos para un robot articulado:

1. Los correspondientes a la colisión de un elemento $\mathbf{A_i}$ con los obstáculos en su espacio de trabajo.

2. Los originados por la colisión entre dos elementos, $\mathbf{A_i}$ y $\mathbf{A_j}$, del robot.

En [Lat91] se proponen las siguientes definiciones para estas dos clases de C-obstáculos:

Definición 2.5
El C-obstáculo debido a la interacción de un elemento $\mathbf{A_i}$ con un obstáculo $\mathbf{B_j}$ se denota por $\mathbf{CB_{ij}}$ y se define por

$$\mathbf{CB_{ij}} = \{q = (q_1, \cdots, q_i, \cdots, q_p) \in C / \mathbf{A_i}(q_1, \cdots, q_i) \cap \mathbf{B_j} \neq 0\}$$

Definición 2.6
El C-obstáculo debido a la interacción del elemento $\mathbf{A_i}$ con el elemento $\mathbf{A_j}$, siendo $i < j$, se denota por $\mathbf{CA_{ij}}$ y se define por

$$\begin{aligned}\mathbf{CA_{ij}} = \quad & \{q = (q_1, \cdots, q_i, \cdots, q_j, \cdots, q_p) \in C \ tal \ que \\ & \mathbf{A_i}(q_1, \cdots, q_i) \cap \mathbf{A_j}(q_1, \cdots, q_j) \neq 0\}\end{aligned}$$

2.2.3 Ejemplo

- Sea el manipulador planar con dos articulaciones de revolución de la figura 2.5 (a) cuyos elementos son segmentos de línea. Se eligen como sistemas de referencia F_W en W, F_{A_1} unido a $\mathbf{A_1}$ y F_{A_2} unido a $\mathbf{A_2}$.

Su espacio de las configuraciones sería $S^1 \times S^1$, por tanto un toro en el espacio euclídeo de 3 dimensiones. Una carta en esta variedad se puede definir asociando un ángulo en $(0, 2\pi)$ con cada una de las dos articulaciones. La figura 2.5(b) muestra estos dos ángulos θ_1 y θ_2. El toro está conectado de forma múltiple y se pueden necesitar varias cartas para formar un atlas. Sin embargo, en muchos casos, es adecuado considerar una única carta y permitir que los ángulos θ_1 y θ_2 varíen en $[0, 2\pi)$, con aritmética módulo 2π. Esta simplificación corresponde con representar el toro por un cuadrado $[0, 2\pi] \times [0, 2\pi]$ (figura 2.5(c)).

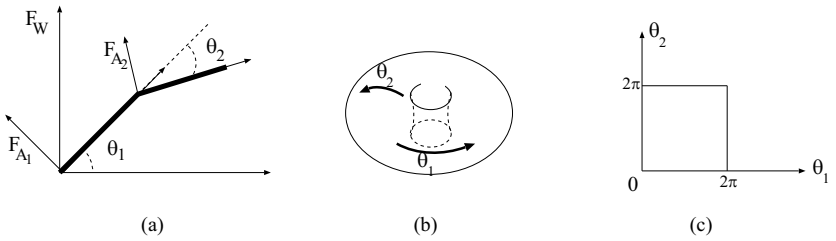


Figura 2.5: Manipulador planar de revolución y su C-espacio

Sea ahora **B** un conjunto de obstáculos puntuales en el espacio del robot $W = R^2$. Los C-obstáculos que se obtienen debidos a los obstáculos puntuales (figura 2.6(a)) situados en el espacio de trabajo aparecen en la figura 2.6(b). Un obstáculo puntual en W se transforma en una curva en C, que representa todas las posibles colisiones del obstáculo con la longitud completa del robot.

2.3 Métodos para el cálculo de los C-obstáculos

Una vez que ya se han definido los conceptos de espacio de las configuraciones y de C-obstáculo asociados tanto a robots móviles como articulados se está en disposición de analizar los métodos que han sido utilizados por diferentes autores para generación de los C-obstáculos. Así, en esta sección se hará una revisión detallada de los procedimientos utilizados para la evaluación del espacio de las configuraciones para robots móviles.

Mediante esta revisión se pretende proporcionar una visión amplia de los diferentes métodos existentes para la evaluación de los C-obstáculos, con sus particularidades e inconvenientes, de tal modo que se puede encuadrar correcta-

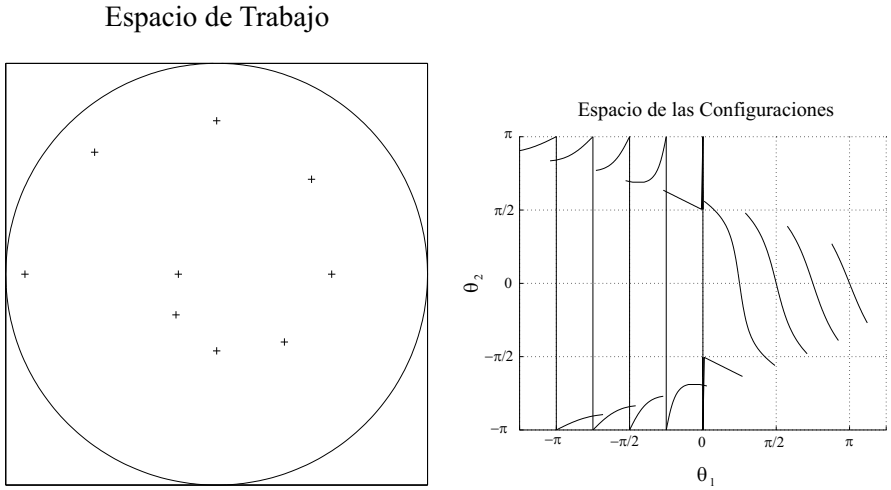


Figura 2.6: (a) Obstáculos en el espacio de trabajo. (b) Obstáculos en el espacio de las configuraciones

mente la propuesta que se realiza en esta tesis doctoral y, así, poner de manifiesto las principales aportaciones introducidas. Según este planteamiento, el estudio no se centrará en los robots articulados, pues como se indicó en los objetivos, este trabajo está enfocado solamente en robots móviles.

Según se describió en la introducción, los métodos existentes para la evaluación de los C-obstáculos se pueden clasificar en dos grupos: los que pertenecen al primero utilizan una representación de los obstáculos y el robot basada en modelos algebraicos; en el segundo se plantea la evaluación de forma discreta de los C-obstáculos, partiendo de representaciones discretas tanto del espacio de trabajo como del robot.

2.3.1 Métodos algebraicos

El uso de modelos algebraicos para representar tanto el robot como los obstáculos tienen como principal ventaja que los objetos se pueden describir por un numero pequeño de parámetros y, además, incluyen un caso particular muy interesante, como es la representación mediante polígonos o poliedros.

En [Lat91] se presenta una recopilación de los principales trabajos que abordan este problema con la construcción de dos representaciones diferentes de los

C-obstáculos, denominadas, respectivamente, *representación de un C-obstáculo con C-restricciones* y *representación límite de un C-obstáculo*. En el análisis se muestra cómo en ambas la complejidad computacional depende de la geometría de los objetos, en concreto, del número de vértices. Además, los algoritmos propuestos para las dos representaciones inicialmente tienen como limitación que tanto el robot como los obstáculos sean convexos. Posteriormente, se considera que los objetos no convexos se pueden descomponer en una unión de convexos, lo que supone una carga computacional adicional.

Para proporcionar una visión general de ellos se van a presentar las dos perspectivas para abordar el problema en el caso sencillo que considera únicamente un espacio de trabajo con objetos modelados mediante regiones poligonales, en donde el robot también tiene forma poligonal y puede moverse libremente en el plano. Ambos puntos de vista parten de una serie de conceptos comunes que se comentarán a continuación, para pasar posteriormente a analizar las particularidades de cada uno de ellos.

2.3.1.1 Conceptos comunes

Sean **A** (robot) y **B** (obstáculo) dos objetos poligonales. Se han de construir las ecuaciones que definen las superficies que limitan al correspondiente C-obstáculo **CB**. Estas superficies provienen de dos tipos de contactos ([Loz83] [Don84]) entre **A** y **B**:

1. contacto tipo A, cuando un lado de **A** contiene un vértice de **B**
2. contacto tipo B, cuando un vértice de **A** está contenido en un lado de **B**

Si la zona de contacto $\mathbf{A}(q) \cap \mathbf{B}$ contiene un vértice de **A** y uno de **B**, entonces el tipo de contacto es A y B.

La suposición de que los interiores de **A** y **B** no intersecten implica que el contacto tipo A solamente es posible para un conjunto de configuraciones de **A**. Este conjunto está determinado por dos expresiones, con cuya conjunción se define la denominada condición de aplicabilidad del contacto tipo A entre el lado j de **A** y el vértice i de **B**, denotada por $\mathrm{APPL}_{i,j}^{\mathbf{A}}(q)$. Esta condición establece la posibilidad de que se produzca un contacto tipo A, para lo que el lado j de A se debe encontrar en un posición adecuada respecto al vértice i de

B. Si **A** se desplaza de forma que se mantenga el contacto tipo A, entonces la configuración del robot se mueve a lo largo de una superficie del C-espacio, denominada C-superficie de tipo A, cuya ecuación es $f_{i,j}^{\mathbf{A}}(q) = 0$. Esta superficie divide C en dos semiespacios. El C-obstáculo **CB** se encuentra completamente dentro del semiespacio determinado por $f_{i,j}^{\mathbf{A}}(q) \leq 0$.

De la misma forma, un contacto tipo B sólo se produce para un subintervalo de orientaciones de **A** determinado por dos condiciones. La conjunción de estas dos condiciones define la condición de aplicabilidad del contacto tipo B entre el vértice i de **A** y el lado j de **B**, denotada por $\text{APPL}_{i,j}^{\mathbf{B}}(q)$. Cuando **A** se mueve manteniendo el contacto tipo B, la configuración del robot se mueve en una superficie, denominada C-superficie de tipo B, cuya ecuación es $f_{i,j}^{\mathbf{B}}(q) = 0$. El C-obstáculo **CB** se encuentra completamente dentro del semiespacio determinado por $f_{i,j}^{\mathbf{B}}(q) \leq 0$.

2.3.1.2 Representación del C-obstáculo con C-restricciones

Esta representación fue propuesta en [Loz83] [Don84], y utiliza las condiciones de aplicabilidad para definir C-restricciones[1] de tipo A o de tipo B dependiendo del tipo de la condición de aplicabilidad.

Definición 2.7

La expresión

$$\text{APPL}_{i,j}^{\mathbf{A}}(q) \Rightarrow [f_{i,j}^{\mathbf{A}}(q) \leq 0]$$

se denomina una C-restricción de tipo A y se denota por $\text{CONST}_{i,j}^{\mathbf{A}}(q)$.

Definición 2.8

La expresión

$$\text{APPL}_{i,j}^{\mathbf{B}}(q) \Rightarrow [f_{i,j}^{\mathbf{B}}(q) \leq 0]$$

se denomina una C-restricción de tipo B y se denota por $\text{CONST}_{i,j}^{\mathbf{B}}(q)$.

En esta representación se describe un C-obstáculo **CB** mediante un predicado $\text{CB}(q)$. $\text{CB}(q)$ se construye como la conjunción de todas las C-restricciones

[1] C-constraints

de tipo A y B generadas por los lados y vértices de **A** y **B**. En [Loz83] y [Don84] se demuestra el siguiente teorema:

Teorema 2.1

Sean **A** *y* **B** *dos polígonos convexos. El C-obstáculo*

$$\mathbf{CB} = \{q \in C / \mathbf{A}(q) \cap \mathbf{B} \neq 0\}$$

es tal que

$$q \in CB \Longleftrightarrow CB(q)$$

con

$$CB(q) \equiv \left(\bigwedge_{i,j} (CONST)_{i,j}^{\mathbf{A}}(q) \right) \bigwedge \left(\bigwedge_{i,j} (CONST)_{i,j}^{\mathbf{B}}(q) \right)$$

Utilizando el predicado CB se puede calcular si una configuración q se encuentra o no en **CB**, evaluando individualmente cada C-restricción hasta que una de ellas evalúe a falso (entonces $q \notin CB$) o todas evalúen a verdadero (entonces $q \in CB$). El número de C-restricciones en CB(q) es $2n_A n_B$.

Si **A** y **B** son polígonos no convexos, se pueden representar como uniones finitas de polígonos convexos. Con esto, el predicado CB(q) se puede poner como la disyunción de los predicados asociados a cada una de las combinaciones posibles de los distintos polígonos convexos en los que se descomponen **A** y **B**. Esto supone la evaluación de un número mayor de predicados, además de introducir una carga computacional adicional asociada al algoritmo de descomposición de polígonos no convexos en una unión de convexos.

Cuando esta formulación general se particulariza para una determinada parametrización del C-espacio, que es diferente para cada tipo de robot, las expresiones para la evaluación del predicado CB(q) son de elevada complejidad aun en casos sencillos. En [BL83] se propone una expresión parametrizada de las C-restricciones para el caso de un robot que se mueve y gira libremente en un plano.

2.3.1.3 Representación del límite del C-obstáculo

En el apartado anterior se representaba un C-obstáculo mediante un predicado CB. Este predicado es conceptualmente simple, pero no proporciona una representación explícita del límite de un C-obstáculo **CB** como una lista de

caras, lados y vértices, con sus ecuaciones y su relación topológica de adyacencia. Solamente permite obtener una serie de condiciones para conocer si una configuración determinada forma o no parte de los C-obstáculos.

A continuación, se presenta un método para construir una representación del límite de **CB** cuando el robot solamente se traslada. Después, se describirá un método más general que se aplica cuando el robot se traslada y rota.

- Caso de traslación

Inicialmente se supone que **A** y **B** son convexos y que **A** se traslada con una orientación fija θ_0. Se puede demostrar fácilmente que los vértices de $\mathbf{CB}_{\theta_0}$ se obtienen de la forma siguiente:

1. Si se cumple la condición de aplicabilidad de un contacto tipo A $(APPL)^{\mathbf{A}}_{i,j}(\theta_0)$, entonces los puntos

$$b_j - a_i(0,0,\theta_0) \text{ y } b_j - a_{i+1}(0,0,\theta_0) \tag{2.1}$$

   son vértices de $\mathbf{CB}_{\theta_0}$, donde b_j y a_i son, respectivamente, los vértices **B** y **A**.

2. Si se cumple la condición de aplicabilidad de un contacto tipo B $(APPL)^{\mathbf{B}}_{i,j}(\theta_0)$, entonces los puntos

$$b_j - a_i(0,0,\theta_0) \text{ y } b_{j+1} - a_i(0,0,\theta_0) \tag{2.2}$$

   son vértices de $\mathbf{CB}_{\theta_0}$.

Las diferencias entre los métodos que calculan el límite del C-obstáculo aparecen en el momento de determinar cuándo se cumplen las condiciones de aplicabilidad. En [Loz83] se propone un algoritmo en el que para establecer cuándo se cumplen ambas condiciones de aplicabilidad se utilizan vectores normales a los lados de **A** y **B**. Teniendo en cuenta las relaciones entre ellos se pueden determinar criterios para la aplicabilidad. El algoritmo así planteado determina que se puede encontrar el límite completo de $\mathbf{CB}_{\theta_0}$ en un tiempo $O(n_A + n_B)$, que es suficientemente óptimo.

Otra posibilidad la plantea también Lozano Pérez ([Loz83]) con un algoritmo menos eficiente que el anterior. El planteamiento consiste en evaluar la diferencia de todos los vértices de **B** con todos los de **A** sin tener en cuenta la condición de

aplicabilidad. Posteriormente se calcula el polígono más pequeño que encierra todos los vértices evaluados anteriormente mediante la envolvente convexa. De esta forma quedaría que

$$\mathbf{CB}_{\theta_0} = conv(vert(\mathbf{B}) \ominus vert(\mathbf{A}(0,0,\theta_0)))$$

donde $vert(P)$ denota el conjunto de vértices de un polígono P, y $conv(S)$ denota la envolvente convexa de un conjunto de puntos S. El tiempo de ejecución de este algoritmo es $O(n_A n_B log(n_A n_B)$.

Si **A** y **B** son polígonos no convexos, se pueden descomponer en polígonos convexos y calcular el C-obstáculo para cada par de polígonos. De la unión de todos estos C-obstáculos se obtiene $\mathbf{CB}_{\theta_0}$, que es un polígono. En [Sha87] se propone un algoritmo que divide **A** y **B** en triángulos y trapecios obteniendo un tiempo $O(n_A^2 n_B^2 \log n_A n_B)$.

- Caso General

Cuando se supone que **A** puede trasladarse y girar se puede extender el método visto anteriormente en el que se obtenía un tiempo de $O(n_A + n_B)$. La extensión consiste en considerar secciones de la coordenada de giro (θ) en cuyos extremos se cumple la condición que algún lado de **A** está alineado con algún lado de **B**. En toda esa sección se cumplen las mismas condiciones de aplicabilidad y por las ecuaciones 2.1 y 2.2 los lados del C-obstáculo en esa sección vienen dados por una función de θ. De esta forma se puede obtener una representación explícita del **CB** en un tiempo $O(n_A n_B (n_A + n_B))$.

Si **A** o **B** son no convexos entonces se puede pensar en descomponerlos en partes convexas, calcular los límites de los C-obstáculos correspondientes y obtener su intersección. Sin embargo, la intersección de los límites que están formados por caras de tipo A o B no es fácil. En lugar de hacer esto, se podría calcular el límite de la sección trasversal del C-obstáculo para una orientación fija de **A**, y analizar cómo varía este límite con los cambios de orientación de **A**. Pero, a diferencia con el caso convexo, la sección trasversal puede experimentar cambios topológicos significativos en orientaciones distintas a aquéllas donde un lado de **A** es paralelo a un lado de **B**. En [AB88] se presenta un método donde directamente se construye el límite de **CB** a partir de **A** y **B**. En este cálculo se emplea un tiempo $O(n_A^3 n_B^3 \log n_A n_B)$. Esta representación describe el límite

del **CB** como una colección de trozos de C-superficies, denominadas caras, y sus relaciones de adyacencia.

Un método más general que no diferencia entre polígonos convexos y no convexos fue propuesto por [Bro89]. Este algoritmo acepta como entrada dos polígonos, donde uno permanece fijo y el otro puede desplazarse y girar, y calcula una representación del correspondiente C-obstáculo, incluyendo información sobre los contactos. No se construye el C-obstáculo tridimensional de forma completa, sino que se calculan múltiples proyecciones sobre planos. Con ello el algoritmo genera una estructura de datos que contiene toda la información métrica y topológica necesaria para describir el conjunto de configuraciones coherentes con cada condición de contacto que puede presentarse. Con este algoritmo se puede obtener un tiempo de $O(n_A^3 n_B^3)$.

2.3.1.4 Consideraciones adicionales

Del mismo modo que los diferentes métodos partían de una base común, una vez que se han examinado se puede obtener una serie de consideraciones para todos ellos.

En todos los algoritmos presentados en esta sección, una característica importante a destacar es que su tiempo de computación depende, tanto si se trata de polígonos convexos o no convexos, del número de vértices del robot y de los obstáculos.

Una condición necesaria que se encuentra presente en todos ellos es que es preciso distinguir unos objetos de otros en espacios de trabajo que cuentan con varios objetos. También, los vértices y lados de los obstáculos y del robot deben de estar perfectamente identificados. Esto supone que existe, bien un conocimiento del espacio de trabajo modelado mediante ecuaciones algebraicas, o bien un sistema de visión con suficiente precisión como para tener una representación exacta del entorno.

De cualquier modo, en general, se suele tender a realizar aproximaciones tanto del robot como de los obstáculos para poder aplicarlos.

2.3.2 Métodos discretos

Otros métodos, en vez de buscar una función que modele los C-obstáculos por medio de métodos algebraicos, tienden a buscar una representación aproximada por medio de la discretización tanto del espacio de trabajo como del robot.

En [NB91] se plantea la creación de un conjunto de formas elementales de las cuales se pueda calcular fácilmente su proyección sobre el C-espacio. Estas formas elementales, generalmente en forma de bitmap, se tratan de identificar en los obstáculos, de manera que éstos se puedan considerar como una combinación de dichas formas elementales. De este modo, la representación del C-obstáculo se puede obtener como la combinación de las C-formas correspondientes. Un procedimiento similar aparece en [LP91] para obtener el bitmap que representa el C-espacio de un robot con 6 grados de libertad.

Desde el punto de vista del planteamiento discreto, también se propone [DHS89] el uso de una malla de $N \times N$ procesadores para obtener una representación del C-espacio en forma de bitmap para robots "rectilineamente" convexos, obteniéndose tiempos de $O(n)$.

Un enfoque sin duda novedoso es el presentado en [Kav95] donde se plantea el uso del concepto de convolución (y para su evaluación el algoritmo de la FFT[2]) entre el bitmap que representa un robot y el del espacio de trabajo, aunque únicamente aplicable a robots móviles. Posteriormente, [Cur98] realizó una formalización matemática del problema, que permitió su aplicación a manipuladores. Siguiendo en esta línea [The02], a través del concepto de deconstrucción aplicó estas técnicas a manipuladores redundantes haciendo uso del cálculo paralelo.

Puesto que la convolución discreta entre el espacio de trabajo es un punto de vista en el que se fundamenta este trabajo, a continuación se va realizar una revisión detallada.

2.3.2.1 Evaluación del C-espacio como una convolución discreta

Este planteamiento trabaja con un espacio de trabajo discreto, de tal modo que un espacio de trabajo limitado $W = [a, b] \times [c, d] \subset R^2$ se representa por una

[2]Fast Fourier Transform (Transformada Rápida de Fourier)

matriz de dimensiones $N \times N$, con N lo suficientemente grande para que la precisión sea apropiada. De este modo cada elemento de la matriz representará la región del espacio

$$W(i,j) = [a + i\frac{b-a}{N}, a + (i+1)\frac{b-a}{N}] \times [c + j\frac{d-c}{N}, c + (j+1)\frac{d-c}{N}]$$

donde $i, j \in 0, \ldots, N-1$. Si algún obstáculo ocupa parte de la celda $W(i,j)$ entonces ésta tomará valor 1, y en caso contrario valor 0.

Del mismo modo se puede crear una matriz $N \times N$ que representa al robot en una configuración determinada (x, y, θ), obteniéndose la matriz $A_{(x,y,\theta)}$, donde los elementos de la matriz que forman parte del robot están marcados con 1.

Según se analizó en el apartado 2.1.3 el C-espacio para una plataforma móvil que puede girar libremente es un espacio R^3. En concreto una configuración viene dada por la tripleta (x_r, y_r, θ_r), donde (x_r, y_r) son las coordenadas de un punto fijo del robot, y θ_r es la orientación del robot. Entonces, en este caso, el C-espacio está limitado al intervalo $[a,b] \times [c,d] \times [0, 2\pi]$. Por lo que se puede almacenar en una matriz tridimensional $N \times N \times N$ donde

$$C(i,j,k) = [a+i\frac{b-a}{N}, a+(i+1)\frac{b-a}{N}] \times [c+j\frac{d-c}{N}, c+(j+1)\frac{d-c}{N}] \times [k\frac{2\pi}{N}, (k+1)\frac{2\pi}{N}]$$

Para evaluar si un determinado elemento de la matriz C, $C(x,y,\theta)$, que representa el C-espacio, pertenece a un C-obstáculo es necesario considerar la matriz que representa al robot en dicha configuración $A_{(x,y,\theta)}$. De este modo, si se comprueba que alguno de los pixels ocupados por el robot coincide con algún pixel ocupado en el espacio de trabajo entonces el pixel $C(x,y,\theta)$ pertenece a un C-obstáculo puesto que el robot intersecta con un obstáculo.

Para evaluar si existe dicho solapamiento entre el robot y los obstáculos se puede realizar mediante la suma del producto pixel a pixel de las dos matrices, la que representa al robot $A_{(x,y,\theta)}$ y la que representa al espacio de trabajo.

$$C(x,y,\theta) = \sum_{i,j=0}^{N-1} W(i,j) A_{(x,y,\theta)}(i,j)$$

Puesto que ambas toman únicamente valores 1 ó 0, si el resultado es 0 no existe solapamiento y, por lo tanto, $C(x,y,\theta)$ es una configuración libre. En cualquier otro caso se trata de un pixel perteneciente a un C-obstáculo.

Por otro lado, al tratarse el robot de un objeto rígido, cuando éste se traslada sin girar, todos sus puntos se trasladan solidariamente con él. Transportándolo

a la matriz que representa al robot $A_{(x,y,\theta)}$, la traslación del robot implica la traslación de cada uno de los pixels que lo representan en la matriz. De esta forma se tiene que

$$A_{(x,y,\theta)}(i,j) = A_{(x-x,y-y,\theta)}(i-x,j-y) = A_{(0,0,\theta)}(i-x,j-x)$$

o lo que es lo mismo, únicamente son necesarias las matrices del robot en la coordenada de referencia con las orientaciones a considerar. Así, el sumatorio anterior se puede poner de la forma

$$C(x,y,\theta) = \sum_{i,j} W(i,j) A_{(0,0,\theta)}(i-x,j-y)$$

Si se considera la matriz simétrica (A'_θ) a la matriz que representa el robot de modo que $A'_\theta(i,j) = A_{(0,0,\theta)}(-i,-j)$ la expresión anterior se puede ver como una convolución de matrices

$$C(x,y,\theta) = \sum_{i,j} W(i,j) A'_\theta(x-i,y-j) = (W \otimes A'_\theta)(x,y)$$

donde $\otimes$ representa el producto de convolución de matrices.

Puesto que para evaluar cada configuración es necesario realizar un sumatorio en dos índices que tienen N valores cada uno, el tiempo necesario para evaluar una configuración será $O(N^2)$. Si se considera una orientación fija es necesario evaluar $N \times N$ posiciones del robot, por lo que para cada orientación se requiere un tiempo $O(N^4)$. Si ahora se incluyen los N posibles giros, el tiempo que lleva la evaluación del C-espacio es $O(N^5)$.

Si se tiene en cuenta el Teorema de Convolución, la operación puede realizarse como la multiplicación de dos matrices elemento a elemento.

Teorema 2.2 (Teorema de Convolución)
Si dos funciones f y g definidas en R son integrables entonces

$$\mathscr{F}(f \otimes g(x)) = \mathscr{F}(f(x)) \times \mathscr{F}(g(x))$$

donde $\mathscr{F}$ representa la Transformada de Fourier, y $\times$ el producto normal de funciones.

Por tratarse en este caso de funciones discretas en dos dimensiones la expresión del Teorema de Convolución se puede expresar en términos de la Transformada de Fourier Discreta (DFT)

$$DFT(f \otimes g(x,y)) = DFT(f(x,y)) \times DFT(g(x,y))$$

donde ahora $\times$ denota el producto elemento a elemento de las matrices.

De este modo la expresión para evaluar una configuración del C-espacio queda de la forma

$$C(x, y, \theta) = DFT^{-1}\left(DFT(W) \times DFT(A'_\theta)\right)(x, y)$$

Es importante destacar, que realizando la convolución de esta forma se obtiene la evaluación del C-espacio en todas las configuraciones con orientación constante con una única operación, sin necesidad de tener que realizarla para cada una de ellas.

Si para realizar la DFT se utiliza el algoritmo de la FFT, para evaluar una transformada en dos dimensiones se requiere un tiempo $O(N^2 logN)$. El tiempo de la FFT es el que impone el tiempo necesario para la evaluación del C-espacio, puesto que se realizan 3 transformadas de Fourier y una única multiplicación de matrices elemento a elemento (requiere un tiempo $O(N^2)$). De este modo, para evaluar el C-espacio del robot con orientación constante se requiere un tiempo $O(N^2 logN)$. Cuando además se consideran N posibles orientaciones, se debe repetir la operación para la matriz que representa el robot en cada orientación, por lo que se requiere un tiempo $O(N^3 logN)$.

El mismo planteamiento se puede realizar cuando se consideran robots moviéndose por el espacio tridimensional. En este caso se utilizaría la Transformada de Fourier en tres dimensiones para realizar la convolución en cada orientación del robot. De este modo, si se considera que el robot mantiene una orientación constante se requeriría un tiempo de cálculo $O(N^3 logN)$. Cuando se consideren variaciones en la orientación respecto algún eje el tiempo se multiplicará por N puesto que se debe realizar una convolución por cada orientación.

Una diferencia fundamental que aparece con respecto de los métodos algebraicos, es que ahora el tiempo de cálculo únicamente depende de la resolución empleada para la representación, y no depende en absoluto de la forma de los obstáculos o del robot.

Posteriormente, Curto ([Cur98]) estableció un formalismo matemático que permite la aplicación de esta idea a manipuladores. Este formalismo se basa fundamentalmente en la elección de sistemas de coordenadas correspondientes al espacio de trabajo, de tal modo que se puedan establecer relaciones de

convolución entre las coordenadas espaciales y los grados de libertad del manipulador. Sin embargo, cuando se aplica a manipuladores redundantes presenta problemas respecto al tiempo de cálculo y al consumo de memoria, pues a lo sumo se establece la convolución de tres grados de libertad (espacio de trabajo tridimensional).

Recientemente, Therón ([The02]) utiliza los planteamientos anteriores para su aplicación paralela sobre manipuladores. En este caso, aprovecha las características cinemáticas del manipulador para realizar una "deconstrucción" del espacio de trabajo, de modo que es posible la aplicación de la convolución a todos los grados de libertad. De esa forma se reduce el tiempo de cálculo aunque se requieren grandes capacidades de almacenamiento.

3

Estructuras de Datos Espaciales, *Quadtrees y Octrees*

Existen numerosas estructuras de datos jerárquicas usadas para la representación de datos espaciales (*k-d trees*, *quadtrees*, *BSP trees*) [Sam90]. Sin duda, una de las más utilizadas cuando se representan datos en dos dimensiones es el **quadtree**, aunque existen distintas adaptaciones de éste dependiendo del tipo de dato que se represente. Su desarrollo surge por la motivación de reducir las necesidades de almacenamiento mediante la agrupación de datos con valores idénticos o similares. Los algoritmos que utilizan este tipo de estructuras suelen conseguir, además de la reducción del espacio de almacenamiento, mejoras en los tiempos de ejecución debidas a dicha agrupación de los datos. La principal característica que diferencia a los *quadtrees* del resto de estructuras jerárquicas radica en la agrupación regular de los datos. Cuando los datos aparecen en espa-

cios tridimensionales la extensión del *quadtree* para incluir la tercera dimensión se denomina *octree*.

En este capítulo solamente se hará una revisión del uso de los *quadtrees* para la representación de regiones, en cuyo caso se denomina *region quadtree*. Esto se debe a que en este trabajo únicamente es necesaria la representación de regiones, ya sea para representar obstáculos en el espacio de trabajo, el robot o los C-obstáculos. Por lo tanto, a partir de este momento, con el término *quadtree* se sobreentenderá que se hace referencia a una *region quadtree*.

3.1 Definiciones básicas

Comenzaremos realizando unas definiciones iniciales referentes a datos bidimensionales. En primer lugar, consideraremos una representación discreta de un espacio bidimensional (*imagen*) mediante una matriz de elementos de información que describen puntos en la imagen (*pixel*). Si únicamente se consideran imágenes binarias los pixels solamente pueden tomar dos posibles valores, blanco (0) o negro (1). En todas aquellas aplicaciones que requieran pixels externos a la imagen (p.e. la convolución), por lo que no están definidos, se entiende que estos son blancos. Denominaremos como "borde de la imagen" a la frontera exterior de la matriz que representa a la imagen.

Una vez sentadas estas bases, se pueden establecer las siguientes definiciones sobre vecindad y conectividad de los pixels de una imagen.

Definición 3.1
Dos pixels se dice que son **4-vecinos** si lo son en la dirección horizontal o vertical. Si además se considera la posibilidad de adyacencia por las esquinas, se dice que son **8-vecinos**.

A partir de la condición de vecindad se puede definir también cuándo un conjunto de pixels es conexo.

Definición 3.2
Un conjunto de pixels S se dice que es **4-conexo** (**8-conexo**) si para dos pixles cualesquiera $p, q \in S$ existe una secuencia de pixels $\{p = p_0, p_1, \ldots, p_n = q\} \in S$, tal que p_{i+1} es 4-vecino (8-vecino) de p_i, $0 \leq i < n$.

3.2 Visión general de los *quadtrees* y *octrees*

Un amplio conjunto de estructuras jerárquicas de datos están basadas en el principio de descomposición recursiva del espacio que deben representar. Con estas estructuras se intenta reducir la necesidad de recursos para su almacenamiento debido a monotonías en la imagen. Se pueden caracterizar por los siguientes criterios:

1. El tipo de dato que representan
2. El proceso para realizar la descomposición
3. La resolución es variable o no

En lo que respecta al tipo de datos, se pueden utilizar para representar puntos, áreas, curvas, superficies y volúmenes. Se puede realizar la descomposición mediante fragmentos iguales en cada nivel, o bien que el tamaño y forma de éstos dependan de las características de la zona que representan. En cuanto a la resolución (el número de veces que el proceso de descomposición es aplicado) puede ser un valor predeterminado o puede depender del conjunto de datos que se está representando.

El *quadtree* es un tipo de este conjunto de estructuras. Como se comentó anteriormente solamente se hará referencia al *quadtree* para la representación de regiones bidimensionales, *region quadtree*. El *quadtree* está basado en la subdivisión sucesiva de una imagen en 4 cuadrantes de las mismas dimensiones hasta encontrar regiones homogéneas. Si la matriz que representa la zona ocupada por uno de los cuadrantes no está constituida en su totalidad por 1s ó 0s, entonces se subdivide a su vez en subcuadrantes, hasta que se obtiene que cada uno de los bloques está formado completamente por 1s ó 0s.

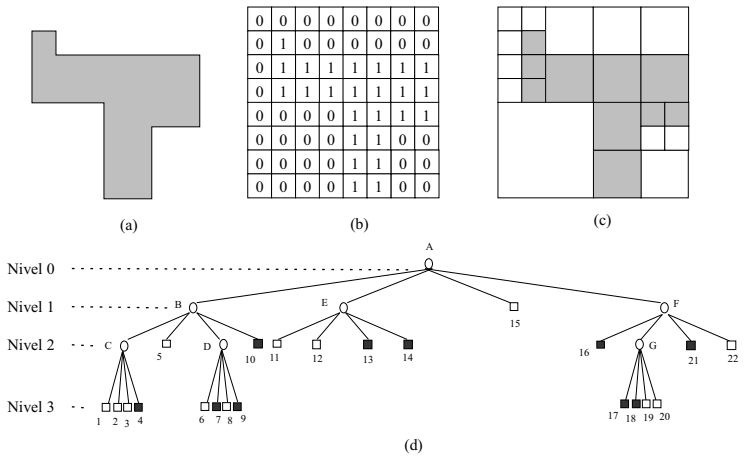


Figura 3.1: (a) Ejemplo de una región, (b) representación mediante una matriz binaria, (c) subdivisión de la imagen en cuadrantes, (d) el correspondiente *quadtree*, donde se representan como ○ los nodos con descendencia y con □ los nodos hoja

En la figura 3.1 se muestra un ejemplo del uso del*quadtree*e para la representación de regiones. Considerando la región de la figura 3.1(a), se puede hacer la representación mediante una matriz binaria $2^3 \times 2^3$, como se refleja en la figura 3.1(b), donde los 1s representan la región y los 0s la zona exterior. Si se realiza una división en cuadrantes, el resultado es el mostrado en la figura 3.1(c). Por último, este proceso se puede representar en forma de árbol, figura 3.1(d), de grado 4 (cada nodo del árbol tiene cuatro hijos).

En la representación en forma de árbol, el nodo raíz representa el espacio completa. Cada uno de los hijos de un nodo representa un cuadrante del área representada por ese nodo, donde los hijos se han ordenado de la siguiente forma NO, NE, SO y SE[1]. Las hojas del árbol se corresponden con aquellos nodos que no necesitan una subdivisión posterior, es decir, la región que representan en la matriz está completamente evaluada por 1s ó 0s. Así, una hoja se considera ocupada si los pixels de la matriz toman valor 1, y vacía si el valor es 0. Por lo tanto, una hoja forma parte de la región o de la zona libre dependiendo de si la

[1] Al dividirse cada hijo en cuadrantes se pueden nombrar mediante la posición del cuadrante que ocupen. Así, el nodo NO es el situado en la posición noroeste, el NE el de la posición noreste, y asi sucesivamente

hoja está ocupada o vacía. En la figura de ejemplo, se muestran los nodos hoja nombrados mediante números, y los que no son hojas con letras.

Así, en una imagen representada por una matriz $2^n \times 2^n$, el nodo raíz del árbol se dice que está en el nivel 0, aumentando el nivel según se profundiza en el árbol. De esta forma los nodos a mayor nivel de profundidad aparecen en el nivel n, y son aquellos que se representan como un único pixel en la matriz.

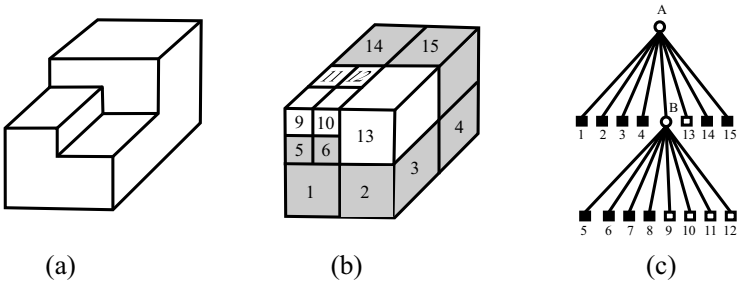


Figura 3.2: (a) Ejemplo de un objeto 3D, (b) subdivisión del objeto en octantes, (c) el correspondiente árbol

Estos *quadtrees* para representar regiones son fácilmente generalizables para representar volúmenes en un espacio tridimensional. La estructura de datos que se obtiene se denomina *region octree*, aunque normalmente se suele referir a ellos simplemente como *octree*. Al igual que con los *quadtrees*, se puede comenzar con una representación del entorno mediante una matriz binaria tridimensional de tamaño $2^n \times 2^n \times 2^n$, donde cada uno de los elementos de la matriz se denomina *voxel*. Para la construcción del *octree* se realiza un subdivisión sucesiva de la matriz en forma de octantes. La subdivisión se realiza hasta que se obtenga un nodo que represente una región completamente libre u ocupada.

Este procedimiento de subdivisión en octantes se representa mediante un árbol de grado 8, en el que el nodo raíz representa todo el espacio y las hojas se corresponden con partes completamente ocupadas por el objeto, o libres. En la figura 3.2(a) se muestra un objeto tridimensional, cuya descomposición se ve representada en la figura 3.2(b). El árbol resultante aparece en la figura 3.2(c).

Los *quadtrees* también pueden ser vistos como un elemento del conjunto de representaciones que consisten en una colección de bloques máximos (conexos), cada uno de ellos representando una determinada región y cuya unión forma

la imagen completa. Un ejemplo sencillo de este tipo de representaciones es la codificación[2] por longitud de secuencia (codificación *runlength*), en la cual los bloques se restringen a rectángulos del tipo $1 \times m$ pixels.

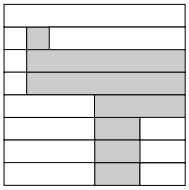


Figura 3.3: Codificación *runlength* del ejemplo de la figura 3.1

En estas representaciones como bloques máximos también están incluidas representaciones en las que los bloques pueden superponerse, como es la transformación axial media (*Medial Axis Transformation*, MAT). En este caso es necesario especificar la posición y el tamaño de cada uno de los bloques.

Cuando se ven los *quadtrees* como un caso particular de la representación en bloques máximos, los bloques en los que se descompone la imagen son disjuntos. Además, puesto que se fundamentan en la subdivisión regular de la imagen, los bloques son de un tamaño predefinido (potencias de 2) y las posiciones donde se encuentran situados también son fijas. Otra particularidad es que los bloques del *quadtree* no tienen por qué corresponderse con regiones máximas homogéneas, es decir, existen uniones de bloques que forman regiones homogéneas.

Si se fusionan todos los bloques del *quadtree* adyacentes (o uniones de bloques), mientras la región obtenida permanezca homogénea, se pueden obtener las regiones conexas máximas homogéneas que forman la imagen y que no se superponen. La partición así obtenida dejará de ser representada por un *quadtree* pero se puede entender como una segmentación de la imagen. Por lo tanto, el *quadtree* puede ser utilizado como un paso previo para la segmentación.

[2]Recibe el nombre de codificación puesto que se utiliza como codificación en la compresión de imágenes

3.3 Estructuras de datos para almacenamiento de los *quadtrees*

Por tratarse el *quadtree* de una estructura en la que los elementos en vez de ser pixels se corresponden con bloques de éstos, surgen diferentes posibilidades de representación: en forma de árbol, como un conjunto de nodos hoja, o como una cadena que representa las ramificaciones del árbol mediante un orden predefinido (recorrido del árbol - *tree traversals*).

3.3.1 Árboles

El uso de una estructura en forma de árbol es la manera más común para llevar a cabo la representación interna del *quadtree*. Tanto es así que, de forma natural, en los ejemplos anteriores se ha utilizado implicitamente esta estructura. Los *quadtrees* así representados se denominan "*quadtrees* explícitos", ya que se especifica todo el proceso de subdivisión de la imagen. A continuación se va a describir la forma de almacenar esta estructura cuando se lleva a cabo su implementación en un ordenador.

Cada uno de los nodos se representa mediante una estructura de datos que contiene:

- Un puntero hacia su nodo padre
- Cuatro punteros hacia sus hijos
- Un campo indicando de qué tipo de nodo se trata

El puntero hacia su nodo padre no es necesario, aunque se suele incluir para facilitar la posibilidad de moverse fácilmente por los nodos que forman el árbol. Así, para obtener el padre de un nodo p se puede hacer mediante una función $PADRE(p)$.

Los cuatro punteros a los hijos se corresponden con los cuatro cuadrantes en los que se puede dividir el nodo. De tal manera que, si el nodo se trata de una hoja esos cuatro punteros estarán vacíos. De forma análoga que con el puntero al padre, se puede realizar una función que determine cada uno de los cuadrantes, i, de un nodo p, $HIJO(p, i)$. Haciendo uso de estas funciones se

puede determinar en qué cuadrante se encuentra un nodo relativo a su padre. Un nodo p estará situado en el cuadrante i respecto a su padre si y sólo si se cumple $HIJO(PADRE(p), i) = p$. De tal modo que se puede obtener la región del espacio que ocupa un nodo determinando las posiciones relativas de forma recursiva hasta llegar al nodo raíz.

El último elemento de la estructura determina el tipo de nodo de que se trata, es decir, describe el contenido de la región que representa el nodo. Así, podrá tener los valores ocupado o libre, en el caso de que se trate de una hoja, o mezcla en el caso de que no sea hoja.

En el caso de que la representación sea para un *octree*, la estructura de datos es análoga con la única diferencia que se cambian los cuatro punteros por ocho para representar los correspondientes octantes.

La desventaja de utilizar una representación en forma de árbol es la excesiva cantidad de memoria necesaria para su almacenamiento. Por ejemplo, dado un *quadtree* con N nodos hoja, son necesarios $(N-1)/3$ nodos que no son hoja. Además, cada uno de estos nodos adicionales necesita espacio para almacenar los punteros a sus hijos. Una posible variación para reducir estos problemas es el uso de árboles binarios (de grado 2), alternando la orientación de la división en la imagen en los niveles del árbol, es decir, se alternan divisiones verticales y horizontales. En este caso, los nodos solamente necesitan espacio para punteros a dos hijos. Además, normalmente se obtiene menor número de hojas. Este hecho se hace más patente cuando aumenta la dimensión del espacio a representar (p.e. *octree*).

3.3.2 Conjunto de nodos hoja

La idea básica para representar el *quadtree* como un conjunto de nodos hoja consiste en codificar cada uno de ellos mediante un número, denominado código localizador, que determina la secuencia de ramas del árbol que se deben seguir desde la raíz para alcanzar dicho nodo. El código localizador está formado por una secuencia de dígitos, cada uno de los cuales indica la rama que ha de seguirse en cada uno de los niveles del árbol.

La forma más habitual de organizar el conjunto de nodos es mediante una lista ordenada de acuerdo con los valores que tomen los códigos. En otros

casos, los códigos localizadores también se pueden utilizar en conjunción con una estructura de árbol para agilizar las operaciones de búsqueda, y determinar el área que representa cada nodo del árbol.

Para explicar el proceso para la generación del código localizador se va a hacer uso de la siguiente notación. En primer lugar se denotará por R al nodo raíz del árbol, por lo tanto en el nivel 0. El nodo del cual se desea el código localizador P se encontrará en un nivel m del árbol. Por lo tanto, se puede determinar una secuencia de nodos $< P_0, P_1, \ldots, P_m >$ por la cual se debe pasar para alcanzar el nodo P desde R. Por lo tanto en dicha secuencia $P_0 = R$, $P_m = P$ y $P_{i-1} = PADRE(P_i)$.

Se numerarán las ramas que salen de cada nodo, NO, NE, SO y SE, con los dígitos 0, 1, 2 y 3 respectivamente. Si se considera el dígito C_i como la rama del árbol que se toma para llegar a P_i desde P_{i-1}, se puede obtener la secuencia de m dígitos $< C_1, C_2, \ldots, C_m >$ a partir de la secuencia de nodos. Por medio de esa secuencia de dígitos se obtendrá un código localizador C en forma de número entero mediante la siguiente expresión

$$C = \sum_{i=1}^{m} C_i 4^{m-i} \quad 0 \leq C_i \leq 3 \quad 0 \leq C < 4^m \tag{3.1}$$

Para ilustrar esta codificación consideremos el *quadtree* del ejemplo de la figura 3.1(d). Para alcanzar el nodo 4 la secuencia de nodos por la que se debe pasar es $< A, B, C, 4 >$, por lo que la secuencia de ramificaciones es $< 0, 0, 3 >$. Así, el código localizador que le corresponde es $C = 0 \times 4^2 + 0 \times 4^1 + 3 \times 4^0 = 3$. Y si se considera el nodo 16, la secuencia de ramificaciones es $< 3, 0 >$ y su código localizador es $C = 3 \times 4^1 + 0 \times 4^0 = 12$

El proceso de decodificación consiste en realizar divisiones del código localizador por 4 sucesivas obteniendo el resto, de manera que se vuelvan a obtener los C_i. Por ejemplo, para el código localizador 3 obtenido anteriormente en la primera división por 4 el resto es 3, lo cual indica la última rama del árbol a tomar. Como el resultado de la división es 0, los restos de las dos siguientes divisiones son 0. De esta forma se obtiene que las ramificaciones a seguir son $< 0, 0, 3 >$, que coinciden con las ramificaciones que llevan al nodo 4.

Sin embargo, el código obtenido de este modo requiere que cada nodo, además de su código localizador, lleve asociado el nivel en el cual se encuentra. Es

necesario, porque si no en el proceso de decodificación es imposible determinar en qué momento se debe parar, puesto que es posible seguir realizando múltiples divisiones de 0 que determinan una rama determinada. Esto se puede ver con el ejemplo anterior si en el código localizador 3 únicamente se hacen dos divisiones por 4, de tal modo que se tiene la secuencia de ramificaciones $< 0, 3 >$ que lleva al nodo 10. O la secuencia $< 3 >$ si únicamente se realiza una división, lo cual lleva al nodo F.

Este problema se puede evitar si se consideran los dígitos 1, 2, 3 y 4 para determinar la rama a seguir (C_i). Además, es necesario modificar la fórmula de la codificación, cambiando el factor 4 por 5, de modo que

$$C = \sum_{i=1}^{m} C_i 5^{m-i} \quad 1 \leq C_i \leq 4 \quad 5^{m-1} \leq C < 5^m \tag{3.2}$$

Mediante esta nueva codificación no es necesario incluir el nivel en el que se encuentra el nodo, puesto que ya se encuentra indirectamente en el código. Se terminará de decodificar en el momento que se obtenga el valor 0.

Considerando esta nueva codificación, el código localizador del nodo 4 sería $C = 1 \times 5^2 + 1 \times 5^1 + 4 \times 5^0 = 34$. Al realizar la decodificación, el resto de dividir 34 entre 5 es 4, lo que indica la primera rama. El resultado de la división, 6, se debe dividir de nuevo, cuyo resto es 1. Al dividir el nuevo resultado 1, da un resto 1 y el resultado de la división 0, por lo que se termina la decodificación.

Los códigos basados en ente tipo de operaciones se denominan códigos localizadores VL (*Variable Length*). El uso del código localizador para representar los nodos presenta ventajas en cuanto a la necesidad de memoria para su almacenamiento si se compara con la estructura de árbol. Se puede optimizar aún más si se representan únicamente aquellos nodos caracterizados como ocupados y no los que son libres. Una representación del *quadtree* utilizando estos códigos se denomina *quadtree* lineal VL.

Además de este tipo de códigos existen otros que presentan ciertas características adicionales, como por ejemplo los códigos localizadores FL (*Fixed Length*) [Gar82][AJ83], basados en la codificación de cada nodo utilizando un número fijo de dígitos. De esta forma, cada nodo, independientemente de la profundidad del árbol a la que se encuentre, tiene el mismo número de dígitos en su codificación. Esto se consigue utilizando un valor determinado de los dígitos

para indicar que la rama a seguir es indiferente, una vez que ya se ha alcanzado el nivel del nodo.

En el trabajo de Gargantini ([Gar82]) se utilizan los dígitos 0, 1, 2 y 3 para especificar las ramas del árbol a seguir, y el dígito 4 cuando la rama es indiferente. De este modo la unión de todos los dígitos produce un número en base 5. En el ejemplo de la figura 3.1(d) el nodo 4 quedaría codificado por el número 003, mientras para el nodo 14 el código sería 134. En el código propuesto en [AJ83] se intercambia el significado de los dígitos 0 y 4.

Entre las diferencias que aparecen entre los códigos VL y FL surge, con una gran relevancia, la forma en la que se realiza la ordenación. Si se ordenan los nodos en orden creciente de su código el resultado difiere con el tipo de código utilizado, obteniendo así una ordenación primero en anchura para el código VL o primero en profundidad para el FL.

3.3.3 Recorrido del árbol

Este tipo de representaciones está basado en realizar un recorrido por los nodos del árbol en profundidad. El resultado es una cadena de símbolos, cada uno de los cuales representa un nodo. Estos símbolos ('O', 'V' o 'P') indicarán si el nodo está ocupado, vacío o parcialmente ocupado. Por ejemplo, el *quadtree* de la figura 3.1 quedaría representado por la cadena

P(P(P(VVVO)VP(VOVO)O)P(VVOO)VP(OP(OOVV)OV)).

En ella se han agrupado mediante paréntesis los nodos que pertenecen al mismo padre para una mejor comprensión, aunque no son necesarios al venir especificados una vez que se ha elegido el orden a seguir en el árbol. Esta representación se denomina expresión DF (*Deep First*) y es debida a Kawaguchi y Endo [KE80]. Con ella se puede obtener una gran compresión de los datos, pero con el inconveniente de que cualquier operación con el *quadtree* (p.e. el acceso a un nodo concreto o la búsqueda de vecinos) se convierte en un proceso engorroso que lleva asociado un mayor tiempo para su consecución.

Entre las representaciones similares se encuentra la codificación de bloques autoadaptativas o codificación GW (*Gray White*) [DCJ76]. Ésta únicamente utiliza dos símbolos '0' y '1' para determinar si se trata de un nodo fuera de

la región ('0') o si se trata de cualquier otro caso ('1'). De este modo el '0' se asocia con el 'V' y el '1' con 'O' y 'P'. Los pixels pertenecientes a la región no se agrupan, por lo que para realizar la decodificación es necesario conocer el número de niveles del árbol.

3.4 Requerimientos de memoria

La principal motivación para el desarrollo de los *quadtrees* surge de la necesidad de un ahorro en la cantidad de memoria necesaria para el almacenamiento de la información. Dicho ahorro, como se ha visto en las secciones anteriores, se obtiene mediante la agrupación de elementos para formar bloques con características homogéneas. Un beneficio secundario, debido a estas agrupaciones, es que en muchos casos se produce una reducción del tiempo necesario para llevar a cabo diferentes tareas, como por ejemplo la segmentación.

Se tiene que la cantidad de memoria necesaria para una representación en forma de *quadtree* es dependiente de la región que se presente (de lo dispersa que se encuentre), no así cuando se utiliza una matriz para la representación, donde siempre es constante. La situación en la que el *quadtree* pierde toda su eficiencia se presenta cuando se trata de representar un tablero de ajedrez. En este caso es necesario dividir todos los nodos hasta el mayor nivel de resolución, en el que cada nodo hoja representa un pixel, y es imposible realizar ninguna agrupación. Un ejemplo se muestra en la figura 3.4.

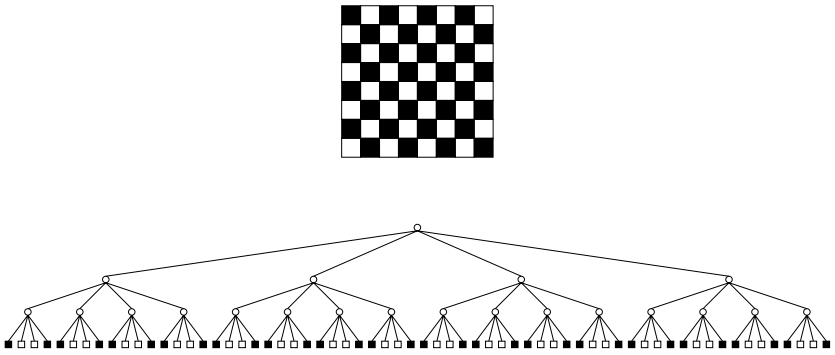


Figura 3.4: Tablero de ajedrez y el *quadtree* asociado

El número de nodos necesarios para representar una región mediante un *quadtree* viene acotado por el teorema de complejidad del *quadtree* [Hun78].

Teorema 3.1 (Teorema de Complejidad del *quadtree*)
El número de nodos en la representación de una región poligonal simple (sin huecos y segmentos que se cruzan) en forma de quadtree es $O(p+q)$ *para una imagen* $2^q \times 2^q$ *y un perímetro* p *de la región medido en pixels.*

En la mayoría de los casos, q *es despreciable frente a* p*, por lo que el número de nodos es proporcional al perímetro.*

El resultado se puede extrapolar para datos tridimensionales reemplazando el perímetro por el área de la superficie de la región, y en general para n-dimensiones donde es sustituido por las (d-1) hiper-superficies.

Aunque el teorema está enunciado para polígonos simples, se ha comprobado que sigue siendo valido para cualquier imagen [RSSW82]. Es necesario destacar que este teorema no solamente tiene repercusiones en cuanto a los requerimientos de memoria, sino también en la complejidad de los algoritmos que trabajan con *quadtrees*, puesto que suelen realizar recorridos por todos los nodos del *quadtree*.

Como se analizó en la sección anterior, el principal inconveniente de utilizar una estructura de árbol para representar el *quadtree* es el excesivo espacio de memoria necesario para los nodos que no son hoja y además mantener la estructura arbórea (punteros). Frente a ésta, es posible utilizar otras estructuras mucho más compactas que no requieren punteros, como son las representaciones mediante la descripción del conjunto de nodos hoja utilizando códigos localizadores o mediante la expresión DF. Pero presentan el inconveniente que, a la hora de realizar operaciones con los *quadtrees* así representados, los algoritmos se vuelven más complejos y menos intuitivos.

3.5 Antecedentes del uso de los *quadtrees*

Es difícil determinar el origen de la idea de la descomposición recursiva, en la cual se fundamentan los *quadtrees*. Inicialmente, la descomposición recursiva se asoció con la agrupación de conjuntos de datos en matrices dispersas, pero fue Morton [Mor66] quien lo introdujo como un método de ordenación en bases de datos espaciales.

Desde entonces el uso de este tipo de estructuras de datos se extendió a gran cantidad de áreas que utilizaban datos espaciales: gráficos por computador, procesamiento de imágenes y modelado de figuras tridimensionales, entre otras.

En el campo de la representación de escenas por ordenador destacan, en los orígenes, los trabajos de Warnock, donde describe el proceso de descomposición recursiva de una imagen con el objetivo de simplificar los algoritmos para eliminar de la representación líneas y caras ocultas. Esta idea también fue utilizada para el procesamiento de imágenes y reconocimiento de patrones. Hunter utilizó las aportaciones anteriores para crear algoritmos de traslación de *quadtrees*, lo que le permitió aplicarlos a la animación por ordenador.

También fue Hunter quien propuso los *octrees* como la extensión natural de los *quadtrees*, pero fueron Reddy y Rubin quienes propusieron el uso de los *octrees* para representar sólidos. Posteriormente, los algoritmos de traslación de *quadtrees* fueron adaptados a los *octrees* por Jackins y Tanimoto. A partir de entonces aparecieron gran cantidad de algoritmos de modelado de sólidos tridimensionales.

El mismo concepto de subdivisión en el que se fundamentan los *quadtrees* fue utilizado en el procesamiento de imágenes. De este modo aparecen las imágenes multiresolución, también representadas como pirámides [Uhr72][Tan76].

En cuanto a la robótica, desde prácticamente el origen de los *quadtrees* su utilidad para representar regiones se ha puesto de manifiesto. Así, Nilsson usa estructuras multiresolución para representar el espacio de trabajo de un robot. Estas estructuras multiresolución son utilizada por Eastman [Eas70] para simplificar el proceso de planificación. Pero es más tarde cuando se comienza a utilizar el *quadtree* como tal en el proceso de planificación [KE80]. Los *octrees* también fueron utilizados para la planificación en espacios tridimensionales [Fav84][ST85].

4

Convolución Jerárquica

En este capítulo se va a proponer un método, denominado *convolución jerárquica*, que permite realizar la convolución de dos funciones discretizadas de forma no uniforme. Así, en lugar de trabajar con una discretización uniforme, las funciones estarán discretizadas de forma no uniforme para adaptarse a los cambios locales de las funciones. De esta forma, es posible alcanzar mayor precisión en aquellas zonas donde las funciones presentan mayores cambios. Además, la función resultado de la convolución jerárquica también estará discretizada de forma no uniforme.

El objetivo de proponer este método de convolución jerárquica es que pueda ser utilizado para la evaluación del C-espacio, ya que en [Bro89] se presenta la evaluación del C-espacio como la convolución de dos funciones. Además, [Kav95] propone la evaluación del C-espacio de forma discreta haciendo uso de la Transformada Rápida de Fourier para realizar la convolución.

Al poder evaluarse el C-espacio como una convolución de funciones en forma discreta, la evaluación se puede realizar a cualquier discretización según se desee conocer el resultado de forma más o menos precisa.

El hecho de poderse aplicar independientemente de la discretización empleada concuerda con el planteamiento de realizarla de forma jerárquica. Parece apropiado utilizar una representación de las dos funciones que convolucionan y la función resultado en forma de *quadtree*, puesto que si observamos el *quadtree* desde el punto de vista piramidal, tenemos una representación con diferentes discretizaciones.

4.1 Convolución discreta: principio de localidad

Vamos a partir de la convolución discreta de dos funciones A y B en dos dimensiones representadas como matrices, donde la función resultado de la convolución es la función C:

$$C(x,y) = \sum_{i,j} B(i,j)A(x-i,y-j) \tag{4.1}$$

Se corresponde con una suma infinita, por lo que al ser las matrices de tamaño finito, se considera una condición nula en el cálculo de C para aquellos índices donde las matrices no están definidas.

Como el objetivo de nuestro trabajo es evaluar el C-espacio mediante la convolución, únicamente se van a considerar funciones binarias, por lo que los elementos de las matrices tomarán valor 0 ó 1, y por lo tanto positivos. Así, la convolución tomará valor igual a 0 en el caso de que todos los términos del sumatorio sean iguales a 0, y mayor que 0 en caso contrario. Por lo tanto, a partir de ahora, aunque no se haga mención expresa, se sobreentenderá que todas las matrices que se consideren son binarias.

Además, como el producto de convolución discreto está orientado concretamente a la evaluación del C-espacio, se irán imponiendo restricciones de manera que los resultados puedan ser aplicados a este fin.

4.1.1 Principio de localidad

El objetivo que se pretende con la convolución jerárquica consiste en **calcular intervalos de la función resultado de la convolución con diferentes discretizaciones** según la necesidad de exactitud que sea necesaria en cada intervalo. De esta forma, en primer lugar es necesario establecer qué intervalos de cada una de las funciones es necesario conocer para evaluar una determinada región de la convolución.

En este sentido, y tomando como base la definición de la convolución discreta 4.1 es posible proponer el siguiente Lema:

Lema 4.1

Sea el producto de convolución discreto de las matrices A y B, donde la matriz A solamente toma valores distintos de 0 *en el rango de índices* $i \in \{\alpha_A, \ldots, \beta_A\}$ $j \in \{\chi_A, \ldots, \delta_A\}$. *Para calcular el valor que toman un conjunto finito,* $\mathbf{C}$, *de elementos de la matriz* C, *solamente es necesario conocer los valores de la matriz* B *para un conjunto finito* $\mathbf{B}$ *de elementos.*

Demostración:

Sea $\mathbf{C} = \{C(x_1, y_1) \ldots C(x_k, y_k) \ldots C(x_n, y_n)\}$ el conjunto de elementos de la matriz C que se desean conocer.

Dado que A solamente toma valores distintos de 0 en el rango de índices $i \in \{\alpha_A, \ldots, \beta_A\}$, $j \in \{\chi_A, \ldots, \delta_A\}$ entonces, para cada uno de los elementos de matriz C a evaluar, el sumatorio puede reducirse a un conjunto discreto de índices puesto que para el resto el producto es 0:

$$C(x,y) = \sum_{i,j} B(i,j)A(x-i, y-j) \qquad (4.2)$$
$$i \in \{x-\beta_A, \ldots, x-\alpha_A\}\ j \in \{y-\delta_A, \ldots, y-\chi_A\}$$

Entonces, para calcular cada uno de los elementos $C(x,y)$ del conjunto $\mathbf{C}$ únicamente es necesario el conjunto $\mathbf{B_{C(x,y)}}$ de elementos de la matriz B, donde

$$\mathbf{B_{C(x,y)}} = \{B(i,j)\ /i \in \{x-\beta_A, \ldots, x-\alpha_A\}\ j \in \{y-\delta_A, \ldots, y-\chi_A\}\} \quad (4.3)$$

que es un conjunto finito.

Por lo tanto, para evaluar todos los elementos del conjunto $\mathbf{C}$ es necesario

conocer los valores que tienen los elementos del conjunto **B** dado por

$$\mathbf{B} = \bigcup_{C(x,y)\in\mathbf{C}} \mathbf{B_{C(x,y)}} \tag{4.4}$$

Además, **B** es un conjunto finito por ser una unión finita de conjuntos finitos. Por lo que el Lema queda demostrado. ♦

El Lema 4.1 indica que, si una de las funciones que convolucionan está limitada a un intervalo, y solamente se desea evaluar el valor de la convolución en una región determinada, no es necesario conocer la segunda función en su totalidad sino solamente una parte de ella. Viene a ser como un principio de localidad, puesto que están relacionadas la región que se desea evaluar y la que se necesario conocer. Es decir, solamente se necesita una información local.

Sin embargo, el conjunto de elementos de la matriz B determinado por el Lema 4.1 puede restringirse aún más si el conjunto de **C** que se desea evaluar y la matriz A cumplen ciertas características adicionales. Para ello vamos a enunciar el siguiente Lema:

Lema 4.2

Sea A una matriz con las siguientes características:

- *el pixel* $A(0,0)$ *toma valor 1,*
- *para cualquier par de pixels con valor 1 existe una sucesión* $\{A(x_1,y_1),\ldots,A(x_n,y_n)\}$ *de pixels 4-vecinos con valor distinto de 0 que los conecta. Es decir, el conjunto de pixels con valor 1 es 4-conexo.*

Sea también $\boldsymbol{C}$ *el conjunto de pixels de la matriz C que se desea evaluar. De tal modo, que el conjunto* $\boldsymbol{C}$ *está delimitado por el conjunto* $\{C(x_1,y_1),\ldots,C(x_m,y_m)\}$*, que forma el contorno externo, y cuyo valor de la convolución es 0.*

Entonces, el conjunto de pixels $\boldsymbol{B}$ *que es necesario conocer de la matriz B para realizar la convolución coincide con* $\boldsymbol{C}$*.*

Demostración:

- $\mathbf{C} \subseteq \mathbf{B}$

  Si $A(0,0) \neq 0$ entonces, por la expresión de la convolución

$$C(x,y) = \sum_{i,j} B(i,j)A(x-i,y-j)$$

se puede deducir que para determinar el valor de $C(x, y)$ es necesario conocer el de $B(x, y)$ pues $A(x-x, y-y) = A(0, 0) = 1$. Por lo tanto si se desea evaluar el valor de un conjunto de elementos $\mathbf{C}$ es necesario conocer, al menos, el valor que tienen en la matriz B, es decir, $\mathbf{C} \subseteq \mathbf{B}$.

- $\mathbf{B} \subseteq \mathbf{C}$

  Veamos ahora que no es necesario conocer ningún elemento de B que no se encuentre en $\mathbf{C}$, es decir $\mathbf{B} \subseteq \mathbf{C}$.

  Sea el elemento $B(k, l)$ que no pertenece al conjunto determinado por $\mathbf{C}$. Si $B(k, l) = 0$, no produce ningún término en el sumatorio, y por lo tanto no afectará al valor que toman los elementos $C(x, y) \in \mathbf{C}$.

  Veamos ahora que pasa si $B(k, l) = 1$. La región que queremos evaluar está delimitada por el contorno $\{C(x_1, y_1), \ldots, C(x_m, y_m)\}$ con valor igual a 0. Entonces, para cada $C(x_i, y_i)$ todos los términos del sumatorio son igual a 0, en concreto el término

  $$B(k, l)A(x_i - k, y_i - l) = 0,$$

  por lo que $A(x_i - k, y_i - l) = 0$.

  Entonces, los elementos del conjunto $\{A(x_1-k, y_1-l), \ldots, A(x_m-k, x_m-l)\}$ de la matriz A también toman valor 0. Como simplemente es una traslación de los elementos del conjunto $\{C(x_1, y_1), \ldots, C(x_m, y_m)\}$ en una cantidad $(-k, -l)$, el conjunto $\mathbf{C}$ trasladado $(-k, -l)$ en A, está delimitado por el contorno $\{A(x_1 - k, y_1 - l), \ldots, A(x_m - k, x_m - l)\}$.

  Si demostramos que para cualquier $C(x, y) \in \mathbf{C}$ el término para evaluarlo $B(k, l)A(x - k, y - l) = 0$, entonces el valor de $B(k, l)$ no era necesario, puesto que independientemente de su valor ese término vale 0.

  Vamos a demostrar esta parte por reducción al absurdo. Supongamos que $B(k, l)A(x - k, y - l) \neq 0$. Como $C(x, y) \in \mathbf{C}$, entonces $A(x - k, y - l)$ se encuentra en la región delimitada por $\{A(x_1 - k, y_1 - l), \ldots, A(x_m - k, x_m - l)\}$ puesto que se trataba de una traslación de $(-k, -l)$.

  Para que el producto sea distinto de 0, $A(x - k, y - l) \neq 0$. Por lo tanto, dadas las características de A en el enunciado, existe una sucesión de pixels 4-vecinos $\{A(0, 0) \ldots A(x - k, y - l)\}$, o lo que es lo mismo $\{A(k - k, l -$

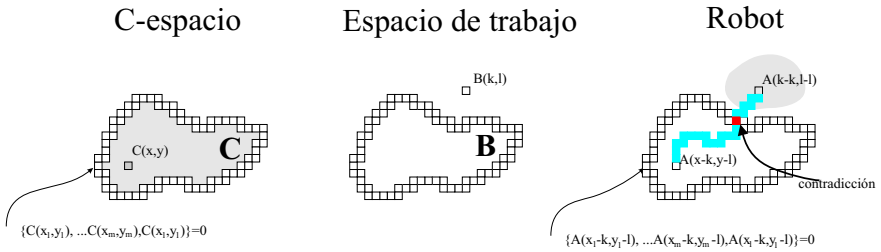


Figura 4.1: Ilustración de los diferentes elementos utilizados en la demostración

$l) \ldots A(x-k, y-l)\}$, con valor distinto de 0. Para cada uno de estos pixels $A(x'-k, y'-l)$, el producto $B(k,l)A(x'-k, y'-l) \neq 0$. Y por lo tanto también $C(x', x') \neq 0$

Entonces existe una sucesión de 4-vecinos $\{C(k,l) \ldots C(x,y)\}$ con valor distinto de 0. Lo cual es imposible puesto que $C(x,y)$ pertenecía a la región delimitada por un contorno de ceros y $C(k,l)$ era externo a ella. Por lo tanto queda demostrado que el pixel $B(k,l)$ no era necesario, puesto que el término correspondiente del sumatorio para la evaluación de cualquier pixel de $\mathbf{C}$ es 0 independientemente del valor que tenga $B(k,l)$.

De este modo, como todos los pixels de $\mathbf{B}$ que son necesarios para evaluar $\mathbf{C}$ se encuentran en $\mathbf{C}$ se tiene que $\mathbf{B} \subseteq \mathbf{C}$.

- $\mathbf{B} = \mathbf{C}$

  Puesto que $\mathbf{B} \subseteq \mathbf{C}$ y $\mathbf{C} \subseteq \mathbf{B}$ se puede concluir que $\mathbf{B} = \mathbf{C}$.

♦

Como resultado de los Lemas anteriores se puede establecer el siguiente corolario.

Corolario 4.1

Es posible calcular la función C discreta, resultado de la convolución de las funciones A y B, como la unión de varios subconjuntos finitos $\mathbf{C_i}$ *(disjuntos o no) (* $C = \cup \mathbf{C_i}$ *) calculados de forma independiente, de modo que para calcular el subconjunto* $\mathbf{C_i}$ *únicamente es necesario un subconjunto finito de B.*

4.2 Efectos de la resolución: conjuntos invariantes

Como el objetivo de la convolución jerárquica es realizar la convolución con diferentes discretizaciones dependiendo de las necesidades en cada zona donde se evalúan las funciones, es necesario determinar el efecto de las diferentes resoluciones con las que se discreticen las funciones.

Entenderemos por resolución la fineza de las divisiones utilizadas en el proceso de discretización de la función. De tal modo que, a mayor resolución más finas son las divisiones y se obtendrá mayor exactitud en la representación de la función. Una forma de medir la resolución será el número de elementos utilizados para la representación de un intervalo determinado de la función. En el caso de las funciones bidimensionales que se han venido utilizando, si se representan por una matriz $N \times N$, entonces diremos que la resolución es N.

Por otro lado, como para realizar la convolución se utilizarán *quadtrees*, que están basados en la subdivisión en cuadrantes, todas las resoluciones serán potencias de 2. Por lo tanto, denominaremos nivel de resolución a la potencia de 2 que proporciona esa resolución. De este modo si $N = 2^n$, el nivel de resolución utilizado es n.

Ni los Lemas 4.1 y 4.2 ni el Corolario 4.1 hacen referencia en ningún momento a la resolución utilizada, por lo que cada uno de los diferentes subconjuntos $\mathbf{C_i}$ pueden ser calculados con diferentes resoluciones. Pero, siempre se ha de tener en cuenta que la representación obtenida en cada subconjunto será valida únicamente a esa resolución.

Según el objetivo que se había presentado, calcular la convolución a diferentes resoluciones **según la necesidad en cada zona**, el corolario 4.1 podría ser de interés. Pero, para que la última parte del objetivo se pueda cumplir, la resolución utilizada en cada región debe ser tal que los resultados sean válidos a cualquier resolución. O lo que es igual, los resultados deberán ser iguales que si se hubieran calculado al mayor nivel de resolución conocido.

Para tener en cuenta esto es necesario hacer las siguientes definiciones.

Definición 4.1 (Conjunto invariante hasta n)
Sea **C** subconjunto de la función C discreta calculado con un nivel de resolución i. Diremos que **C** es invariante hasta n si y solo si para cada elemento $C(x, y) \in \mathbf{C}$ el valor que toma la función en su interior no se ve modificado cuando se realiza el cálculo con una resolución j tal que $n \geq j \geq i$.

Esta definición indica que si un subconjunto a resolución i es invariante hasta n, no es necesario evaluarlo utilizando un nivel de resolución n, puesto que el resultado será el mismo que el evaluado al nivel de resolución i.

Definición 4.2 (Subconjunto invariante)
Sea **C** subconjunto de la función C discreta calculado con un nivel de resolución i. Diremos que **C** es **invariante** si y solo si es invariante hasta n para todo $n \geq i$.

Con esta nueva definición se va todavía más lejos, puesto que si un subconjunto es invariante, el resultado será valido para cualquier resolución mayor.

Utilizando estas definiciones el Corolario 4.1 puede convertirse en el siguiente

Corolario 4.2
Si se realiza una partición del espacio donde se encuentra definida la función C en subconjuntos finitos invariantes, o invariantes hasta n

$$\mathbf{C} = \bigcup \mathbf{C_i} \quad : \mathbf{C_i} \textit{ es invariante o invariante hasta } n$$

entonces se conoce el resultado de la función hasta la resolución n. Además, para el cálculo de cada uno de los $\mathbf{C_i}$ *solamente es necesario conocer un conjunto finito* $\mathbf{B_i}$ *de la función B.*

La única dificultad radica en que en principio no se conoce la condición de invarianza, ni cuáles son los subconjuntos invariantes del resultado de la convolución. Para poder caracterizarlos es preciso conocer qué es lo que sucede con el producto de convolución al pasar de un nivel de resolución a otro.

4.3 Relaciones entre distintos niveles de resolución

En esta sección se pretende analizar diferentes relaciones que aparecen cuando se evalúa la función resultado de la convolución con diferentes niveles de resolución. Se irán enunciando lemas con resultados parciales, con el objetivo de ir avanzando en el análisis de lo que sucede al cambiar la resolución, y de este modo, encontrar una condición de suficiencia de los subconjuntos invariantes.

Como se va a trabajar con subconjuntos a diferentes niveles de resolución, denotaremos $\mathbf{C^n}$ al subconjunto $\mathbf{C}$ calculado al nivel de resolución n. De igual modo, $C^n(x,y)$ será un elemento de $\mathbf{C^n}$.

El primer efecto inmediato al aumentar el nivel de resolución en la discretización de una función X cualquiera es que, para dos niveles de resolución consecutivos n y $n+1$, el elemento $X^n(i,j)$ se convierte en los siguientes cuatro elementos: $X^{n+1}(2i,2j)$, $X^{n+1}(2i+1,2j)$, $X^{n+1}(2i,2j+1)$ y $X^{n+1}(2i+1,2j+1)$. Además, para una función binaria X se tiene que la discretización se realiza de tal modo que el elemento de la matriz que la representa tomará valor 0 únicamente si lo hace la función en toda la región que representa. De este modo se tendrá:

$$X^n(i,j)=0 \Leftrightarrow \begin{cases} X^{n+1}(2i,2j) & =0 \\ X^{n+1}(2i+1,2j) & =0 \\ X^{n+1}(2i,2j+1) & =0 \\ X^{n+1}(2i+1,2j+1) & =0 \end{cases} \tag{4.5}$$

$$X^n(i,j)=1 \Leftrightarrow \text{alguno de} \begin{cases} X^{n+1}(2i,2j) & =1 \\ X^{n+1}(2i+1,2j) & =1 \\ X^{n+1}(2i,2j+1) & =1 \\ X^{n+1}(2i+1,2j+1) & =1 \end{cases} \tag{4.6}$$

Por lo tanto, independientemente del valor que tome, siempre se cumple que

$$X^n(i,j) \geq \begin{matrix} X^{n+1}(2i,2j) \\ X^{n+1}(2i+1,2j) \\ X^{n+1}(2i,2j+1) \\ X^{n+1}(2i+1,2j+1) \end{matrix} \tag{4.7}$$

Por otro lado, al evaluar la convolución, la expresión general

$$C^{n+1}(x,y) = \sum_{i,j} B^{n+1}(i,j) A^{n+1}(x-i, y-j)$$

se puede extender en cuatro sumatorios correspondientes a los índices pares e impares en ambos ejes. De forma que se obtiene

$$\begin{aligned} C^{n+1}(x,y) = \quad & \sum_{i,j} B^{n+1}(2i,2j) A^{n+1}(x-2i, y-2j) + \\ & \sum_{i,j} B^{n+1}(2i+1,2j) A^{n+1}(x-2i-1, y-2j) + \\ & \sum_{i,j} B^{n+1}(2i,2j+1) A^{n+1}(x-2i, y-2j-1) + \\ & \sum_{i,j} B^{n+1}(2i+1,2j+1) A^{n+1}(x-2i-1, y-2j-1) \end{aligned} \tag{4.8}$$

Teniendo en cuenta esta nueva expresión se puede expresar el siguiente Lema:

Lema 4.3

Cada uno de los cuatro elementos que forman $C^n(x,y)$ a la resolución $n+1$ ($C^{n+1}(2x,2y)$, $C^{n+1}(2x+1,2y)$, $C^{n+1}(2x,2y+1)$ y $C^{n+1}(2x+1,2y+1)$) se puede acotar en función de los valores de los vecinos de $C^n(x,y)$ en $\mathbf{C^n}$.

Además las cotas son las siguientes:

$$\begin{aligned} C^{n+1}(2x,2y) &\leq C^n(x,y) + C^n(x-1,y) + \\ &\quad + C^n(x,y-1) + C^n(x-1,y-1) && (4.9) \\ C^{n+1}(2x+1,2y) &\leq 2C^n(x,y) + 2C^n(x,y-1) && (4.10) \\ C^{n+1}(2x,2y+1) &\leq 2C^n(x,y) + 2C^n(x-1,y) && (4.11) \\ C^{n+1}(2x+1,2y+1) &\leq 4C^n(x,y) && (4.12) \end{aligned}$$

Demostración:

Veamos que es lo que ocurre con el elemento $C^{n+1}(2x,2y)$. Considerando cada uno de los cuatro sumatorios que aparecen en la expresión 4.8, y acotando cada uno de ellos por el valor que le corresponde en el nivel de resolución n por medio de la relación 4.7 se tiene que

$$
\begin{aligned}
C^{n+1}(2x,2y) &= \sum_{i,j} B^{n+1}(2i,2j)A^{n+1}(2x-2i,2y-2j)+ \\
&\quad \sum_{i,j} B^{n+1}(2i+1,2j)A^{n+1}(2x-2i-1,2y-2j)+ \\
&\quad \sum_{i,j} B^{n+1}(2i,2j+1)A^{n+1}(2x-2i,2y-2j-1)+ \\
&\quad \sum_{i,j} B^{n+1}(2i+1,2j+1)A^{n+1}(2x-2i-1,2y-2j-1) \\
&\leq \sum_{i,j} B^{n}(i,j)A^{n}(x-i,y-j)+ \\
&\quad \sum_{i,j} B^{n}(i,j)A^{n}((x-1)-i,y-j)+ \\
&\quad \sum_{i,j} B^{n}(i,j)A^{n}(x-i,(y-1)-j)+ \\
&\quad \sum_{i,j} B^{n}(i,j)A^{n}((x-1)-i,(y-1)-j) \\
&\leq C^{n}(x,y)+C^{n}(x-1,y)+C^{n}(x,y-1)+C^{n}(x-1,y-1)
\end{aligned}
\tag{4.13}
$$

Para los otros tres elementos la demostración es análoga. ♦

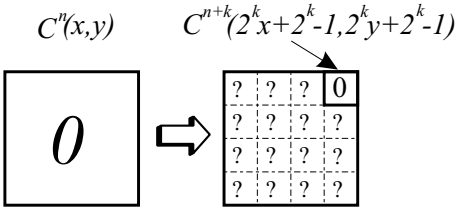
A partir del resultado de este Lema se pueden observar ciertas características que se van a mantener al aumentar el nivel de resolución. Las vamos a ver mediante el enunciado de los siguientes Lemas.

Lema 4.4

Sea el elemento $C^n(x,y)$ evaluado mediante la convolución discreta a resolución n, y cuyo valor es igual a 0. Entonces se tiene que a resolución $n+k$

$$C^{n+k}(2^k x + 2^k - 1, 2^k y + 2^k - 1) = 0 \ \forall k \geq 0$$

Explicación gráfica



Demostración:

- Veámoslo para $k = 1$.

  Tenemos que

$$C^{n+1}(2x+2-1,2y+2-1) = C^{n+1}(2x+1,2y+1),$$

por la ecuación 4.12 se tiene que

$$C^{n+1}(2x+2-1, 2y+2-1) \leq 4C^n(x,y)$$

Por lo tanto si $C^n(x,y) = 0$ entonces $C^{n+1}(2x+2-1, 2y+2-1) = 0$

- Ahora veamos que si se cumple para k también lo hace para $k+1$. De forma análoga a la anterior podemos acotarlo con el valor del elemento correspondiente en la resolución anterior por lo que

$$\begin{aligned} &C^{n+k+1}(2^{k+1}x + 2^{k+1} - 1, 2^{k+1}y + 2^{k+1} - 1) = \\ &C^{n+k+1}(2(2^k x + 2^k - 1) + 1, 2(2^k y + 2^k - 1) + 1) \leq \\ &4C^{n+k}(2^k x + 2^k - 1, 2^k y + 2^k - 1) = 0 \end{aligned}$$

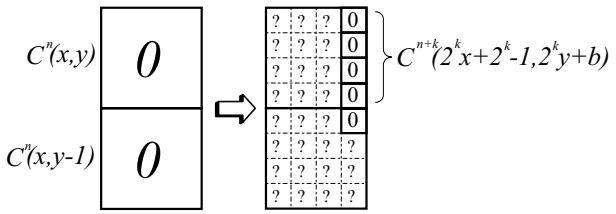
- Entonces, por inducción se cumple para cualquier $k \geq 0$. ♦

Lema 4.5

Sea el elemento $C^n(x,y) = 0$ evaluado mediante la convolución discreta a resolución n. Si el elemento $C^n(x, y-1) = 0$, entonces, en el nivel de resolución $n+k$ se cumple

$$C^{n+k}(2^k x + 2^k - 1, 2^k y + b) = 0 \quad / \; 0 \leq b < 2^k \qquad \forall k \geq 0$$

Explicación gráfica



Demostración:

- Veamos qué es lo que sucede para $k = 1$.

  Los elementos que hay que evaluar son $C^{n+1}(2x+1, 2y)$ y $C^{n+1}(2x+1, 2y+1)$. Por lo que utilizando las ecuaciones 4.10 y 4.12 se obtiene:

$$\begin{aligned} &C^{n+1}(2x+1, 2y) \leq 2C^n(x,y) + 2C^n(x, y-1) = 0 \\ &C^{n+1}(2x+1, 2y+1) \leq 4C^n(x,y) = 0 \end{aligned}$$

- Ahora veamos que si se cumple para k también lo hace para $k+1$.

  Si se cumple para k entonces se tiene que

  $$C^{n+k}(2^k x + 2^k - 1, 2^k y + b) = 0 \quad / \quad 0 \leq b < 2^k$$

  Además, por el Lema anterior, al ser $C^n(x, y-1) = 0$ entonces se tiene que

  $$C^{n+k}(2^k x + 2^k - 1, 2^k(y-1) + 2^k - 1) = C^{n+k}(2^k x + 2^k - 1, 2^k y - 1) = 0$$

  Por lo tanto para cada

  $$C^{n+k}(2^k x + 2^k - 1, 2^k y + b) = 0 \quad / \quad 0 \leq b < 2^k$$

  se tiene que

  $$C^{n+k}(2^k x + 2^k - 1, 2^k y + b - 1) = 0 \quad / \quad 0 \leq b < 2^k$$

  Entonces, aplicando el primer paso de la demostración a cada uno de estos pares, se tiene que

  $$C^{n+k+1}(2(2^k x + 2^k - 1) + 1, 2(2^k y + b)) =$$
  $$C^{n+k+1}(2^{k+1} x + 2^{k+1} - 1, 2^{k+1} y + 2b) = 0$$

  $$C^{n+k+1}(2(2^k x + 2^k - 1) + 1, 2(2^k y + b) + 1) =$$
  $$C^{n+k+1}(2^{k+1} x + 2^{k+1} - 1, 2^{k+1} y + 2b + 1) = 0$$

  y por lo tanto

  $$C^{n+k+1}(2^{k+1} x + 2^{k+1} - 1, 2^{k+1} y + b) = 0 \ / \ 0 \leq b < 2^{k+1}$$

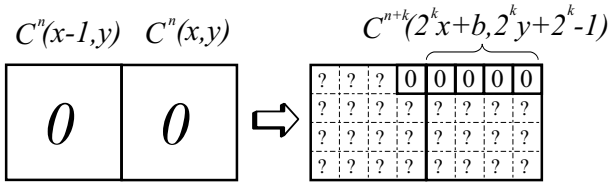
- Entonces por inducción es cierto para cualquier $k \geq 0$. ♦

Lema 4.6

Sea el elemento $C^n(x, y) = 0$ evaluado mediante la convolución discreta a resolución n. Si el elemento $C^n(x-1, y) = 0$, entonces en el nivel de resolución $n+k$

$$C^{n+k}(2^k x + b, 2^k y + 2^k - 1) = 0 \ / \ 0 \leq b < 2^k \ \forall k \geq 0$$

Explicación gráfica



Demostración:

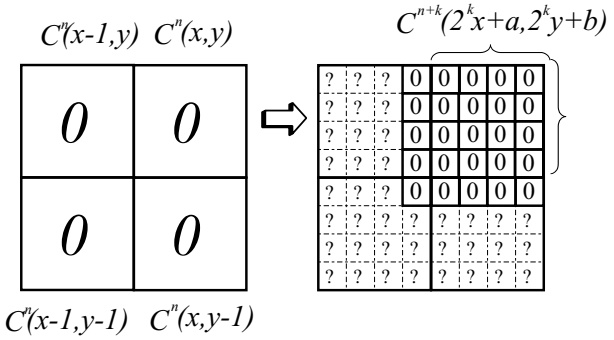
La demostración es análoga a la anterior. ♦

Una vez vistos estos lemas podemos encontrar la condición de suficiencia para que un elemento calculado a un nivel de resolución n sea invariante.

Teorema 4.1

Sea el elemento $C^n(x,y) = 0$ evaluado mediante la convolución discreta a resolución n. Si además $C^n(x-1,y) = C^n(x,y-1) = C(x-1,y-1) = 0$ entonces $C^n(x,y)$ es invariante.

Explicación gráfica



Demostración:

- Veamos que es invariante hasta $n+1$.

  Para que sea invariante hasta $n+1$ todos los elementos que lo componen a esta resolución deben ser iguales a 0. Comprobemoslo haciendo uso de las ecuaciones 4.9, 4.10, 4.11 y 4.12.

$$\begin{aligned}
C^{n+1}(2x,2y) &\leq C^n(x,y) + C^n(x-1,y) + \\
&\quad C^n(x,y-1) + C^n(x-1,y-1) = 0 \\
C^{n+1}(2x+1,2y) &\leq 2C^n(x,y) + 2C^n(x,y-1) = 0 \\
C^{n+1}(2x,2y+1) &\leq 2C^n(x,y) + 2C(x-1,y) = 0 \\
C^{n+1}(2x+1,2y+1) &\leq 4C^n(x,y) = 0
\end{aligned}$$

Por lo cual queda demostrado que es invariante hasta $n+1$.

- Veamos ahora que si es invariante hasta $n+k$ también lo es hasta $n+k+1$.

  Si es invariante hasta $n+k$ entonces

$$C^{n+k}(2^k x + a, 2^k y + b) = 0 \ / \ 0 \leq a, b < 2^k$$

  Puesto que $C^n(x-1, y) = C^n(x-1, y-1) = 0$, por el lema 4.5

$$C^{n+k}(2^k(x-1) + 2^k - 1, 2^k y + b) = 0 \ \ / \ 0 \leq b < 2^k \ \forall k \geq 0$$

$$C^{n+k}(2^k x - 1, 2^k y + b) = 0 \ \ / \ 0 \leq b < 2^k \ \forall k \geq 0$$

  De igual modo, ya que $C^n(x, y-1) = C^n(x-1, y-1) = 0$, ahora por el Lema 4.6

$$C^{n+k}(2^k x + a, 2^k(y-1) + 2^k - 1) = 0 \ \ / \ 0 \leq a < 2^k \ \forall k \geq 0$$

$$C^{n+k}(2^k x + a, 2^k y - 1) = 0 \ \ / \ 0 \leq a < 2^k \ \forall k \geq 0$$

  Y, por último, por ser $C^n(x-1, y-1) = 0$ y aplicando el Lema 4.4

$$C^{n+k}(2^k(x-1) + 2^k - 1, 2^k(y-1) + 2^k - 1) = 0 \ \forall k \geq 0$$

$$C^{n+k}(2^k x - 1, 2^k y - 1) = 0 \ \forall k \geq 0$$

  Por lo tanto para cada

$$C^{n+k}(2^k x + a, 2^k y + b) = 0 \ / \ 0 \leq a, b < 2^k$$

  se tiene

$$C^{n+k}(2^k x + a - 1, 2^k y + b) = C^{n+k}(2^k x + a, 2^k y + b - 1) =$$

$$C^{n+k}(2^k x + a - 1, 2^k y + b - 1) = 0$$

  Aplicando el mismo procedimiento visto en el primer paso de la demostración a cada uno de los términos

$$C^{n+k}(2^k x + a, 2^k y + b) = 0 \ / \ 0 \leq a, b < 2^k$$

se tiene que cada uno los elementos en que se divide en el nivel de resolución $n+k+1$

$$C^{n+k+1}(2^{k+1}x+2a, 2^{k+1}y+2b) = C^{n+k+1}(2^{k+1}x+2a, 2^{k+1}y+2b+1) =$$

$$C^{n+k+1}(2^{k+1}x+2a+1, 2^{k+1}y+2b) = C^{n+k+1}(2^{k+1}x+2a+1, 2^{k+1}y+2b+1) = 0$$

Por lo tanto se deduce que $C^n(x,y)$ es invariante hasta $n+k+1$.

- Por inducción, $C^n(x,y)$ es invariante hasta $n+k$ $\forall k \geq 0$. Y por lo tanto es invariante. ♦

Corolario 4.3

Sea $\mathbf{C^n}$ *un subconjunto de la función C evaluada mediante la convolución discreta a resolución n. Si todo elemento* $C^n(x,y) \in \mathbf{C^n}$ *toma valor 0, y* $\forall C^n(x,y) \in \mathbf{C^n}$ *se tiene que* $C^n(x-1,y) = C^n(x,y-1) = C^n(x-1,y-1) = 0$*, entonces, el subconjunto* $\mathbf{C^n}$ *es invariante.*

Demostración:

A partir del Lema 4.6 se puede demostrar que cada uno de los elementos que forman $\mathbf{C^n}$ es invariante, es decir, que no cambian de valor al aumentar la resolución. Por lo tanto también lo será el conjunto formado por todos ellos. ♦

4.4 Convolución jerárquica

Una vez que se han enunciado los teoremas previos se está en condiciones de proponer un método para la evaluación de la convolución de una forma jerárquica.

El objetivo del método es evaluar la convolución de dos funciones de forma discreta en cada región con diferentes resoluciones, dependiendo del nivel de resolución necesario para que la función resultado se represente de forma correcta. Puesto que se trata de una convolución discreta, se obtendrá el resultado válido para una resolución máxima determinada. Para ello, es necesario conocer las dos funciones que convolucionan a dicho nivel de resolución máximo.

4.4.1 Planteamiento de la convolución jerárquica

Este planteamiento se fundamenta en la base de que es posible obtener representaciones discretas de las funciones a cualquier nivel de resolución.

- Inicialmente, se realizará el cálculo de la convolución a un nivel de resolución bajo, n_1, en la forma matricial clásica. A partir del resultado obtenido, la representación de la función C a resolución n_1, es posible determinar cierto subconjunto $\mathbf{C_1^{n_1}} \in C$ (puede ser incluso el conjunto vacío) invariante (será aquel conjunto de pixels que cumplan el Corolario 4.3 y por lo tanto con valor 0).

  Por definición de conjunto invariante, la región que representa $\mathbf{C_1^{n_1}}$ es válida aunque se aumente la resolución para evaluarlo, por lo que no será necesario calcularlo a mayor resolución. Por lo tanto, solamente se necesitará calcular a mayor resolución el subconjunto complemento $\overline{\mathbf{C}}_\mathbf{1}^{\mathbf{n_1}}$, tal que

  $$C = \mathbf{C_1^{n_1}} \cup \overline{\mathbf{C}}_\mathbf{1}^{\mathbf{n_1}}$$

- En el siguiente paso se calculará el subconjunto $\overline{\mathbf{C}}_\mathbf{1}^{\mathbf{n_1}}$ a resolución mayor, $\overline{\mathbf{C}}_\mathbf{1}^{\mathbf{n_2}}$, de manera que se pueda alcanzar mayor detalle en las variaciones de la función en la región que representa. Para lo cual, por el Lema 4.2, solamente es necesario el mismo conjunto de pixels, $\overline{\mathbf{B}}_\mathbf{1}^{\mathbf{n_2}}$, puesto que alrededor los pixels toman valor 0, ya que forman parte del conjunto invariante $\mathbf{C_1^{n_1}}$. Así, $\overline{\mathbf{C}}_\mathbf{1}^{\mathbf{n_2}}$ se calculará mediante la convolución de $\overline{\mathbf{B}}_\mathbf{1}^{\mathbf{n_2}}$ con la función $\mathbf{A^{n_2}}$, cuyos valores no nulos estaban en una región limitada.

  Una vez llegado a este punto, el valor de la función C calculado como $C = \mathbf{C_1^{n_1}} \cup \overline{\mathbf{C}}_\mathbf{1}^{\mathbf{n_2}}$ es válido hasta la resolución n_2, puesto que $\mathbf{C_1^{n_1}}$ es invariante, y por lo tanto, el resultado válido para cualquier resolución mayor y $\overline{\mathbf{C}}_\mathbf{1}^{\mathbf{n_2}}$ está calculado a esta resolución. Al igual que se hizo anteriormente se puede encontrar un subconjunto $\mathbf{C_2^{n_2}}$ de $\overline{\mathbf{C}}_\mathbf{1}^{\mathbf{n_2}}$ ($\mathbf{C_2^{n_2}} \subset \overline{\mathbf{C}}_\mathbf{1}^{\mathbf{n_2}}$) invariante, y por lo tanto con valores 0.

  Si consideramos el complemento de $\mathbf{C_2^{n_2}}$ relativo a $\overline{\mathbf{C}}_\mathbf{1}^{\mathbf{n_2}}$ ($\overline{\mathbf{C}}_\mathbf{1}^{\mathbf{n_2}} \backslash \mathbf{C_2^{n_2}}$) que denotaremos como $\overline{\mathbf{C}}_\mathbf{2}^{\mathbf{n_2}}$, se tiene que

  $$C = \mathbf{C_1^{n1}} \cup \mathbf{C_2^{n2}} \cup \overline{\mathbf{C}}_\mathbf{2}^{\mathbf{n_2}}$$

De nuevo, $\overline{\mathbf{C}}_{\mathbf{2}}^{\mathbf{n_2}}$ es necesario calcularlo con un mayor nivel de resolución ($\overline{\mathbf{C}}_{\mathbf{2}}^{\mathbf{n_3}}$), puesto que es el único subconjunto no invariante. Además, al estar rodeado por 0s para su evaluación solamente es necesaria la misma región del espacio de trabajo tomada con semejante resolución, $\overline{\mathbf{B}}_{\mathbf{2}}^{\mathbf{n_3}}$.

- Este proceso se repetirá $k-1$ veces. Como resultado acumulado de los pasos anteriores de tendrá que

$$C = \mathbf{C}_{\mathbf{1}}^{\mathbf{n_1}} \cup \mathbf{C}_{\mathbf{2}}^{\mathbf{n_2}} \cup ..... \cup \mathbf{C}_{\mathbf{k-1}}^{\mathbf{n_{k-1}}} \cup \overline{\mathbf{C}}_{\mathbf{k-1}}^{\mathbf{n_{k-1}}}$$

donde el subconjunto $\overline{\mathbf{C}}_{\mathbf{k-1}}^{\mathbf{n_{k-1}}}$ debe ser calculado a mayor resolución, que ahora será n_k.

$\overline{\mathbf{C}}_{\mathbf{k-1}}^{\mathbf{n_k}}$ se evaluará como la convolución de $\overline{\mathbf{B}}_{\mathbf{k-1}}^{\mathbf{n_k}}$ y $\mathbf{A}^{\mathbf{n_k}}$. Como $\overline{\mathbf{C}}_{\mathbf{k-1}}^{\mathbf{n_k}}$ se ha evaluado a la resolución n_k, por definición, será invariante hasta n_k, por lo que $\mathbf{C}_{\mathbf{k}}^{\mathbf{n_k}} = \overline{\mathbf{C}}_{\mathbf{k-1}}^{\mathbf{n_k}}$.

Así, se obtendrá la función C evaluada en

$$C = \mathbf{C}_{\mathbf{1}}^{\mathbf{n_1}} \cup \mathbf{C}_{\mathbf{2}}^{\mathbf{n_2}} \cup ..... \cup \mathbf{C}_{\mathbf{k-1}}^{\mathbf{n_{k-1}}} \cup \mathbf{C}_{\mathbf{k}}^{\mathbf{n_k}}$$

calculado como la unión de varios subconjuntos, cada uno de ellos con un nivel de resolución distinto y todos ellos invariantes hasta n_k.

Además, en el procedimiento de obtención de la convolución se puede observar que según se va aumentando la resolución, si se van obteniendo subconjuntos invariantes, la región en la que es necesario realizar los cálculos es menor. Por lo que, según el Lema 4.2 también, se requieren regiones menores de la función B.

4.4.2 Los *quadtrees* en la convolución jerárquica

En el método expuesto de la convolución jerárquica es necesario tener representadas las funciones A y B a diferentes niveles de resolución. Por esta razón es necesario un tipo de estructura en la cual estén representados los niveles de resolución necesarios. Una estructura que reúne estos requisitos es *quadtree*, además de ser una estructura bien conocida y permitir un gran compactación de los datos.

Por lo tanto, las funciones A y B se representan mediante *quadtrees*, por lo que se tendrán representadas las funciones con una resolución variable, y serán el punto de partida para realizar la convolución jerárquica. El resultado de la convolución también se evalúa con diferentes niveles de resolución, por lo que la función C también deberá ser representada por un *quadtree*.

Así, la convolución jerárquica tendrá como origen las representaciones en forma de *quadtree* de las funciones que convolucionan, A y B, para obtener un nuevo *quadtree* que represente el resultado de la convolución, sin necesidad de pasar por estructuras intermedias. Esto supondrá ventajas en cuanto al tiempo de cálculo y necesidades de memoria.

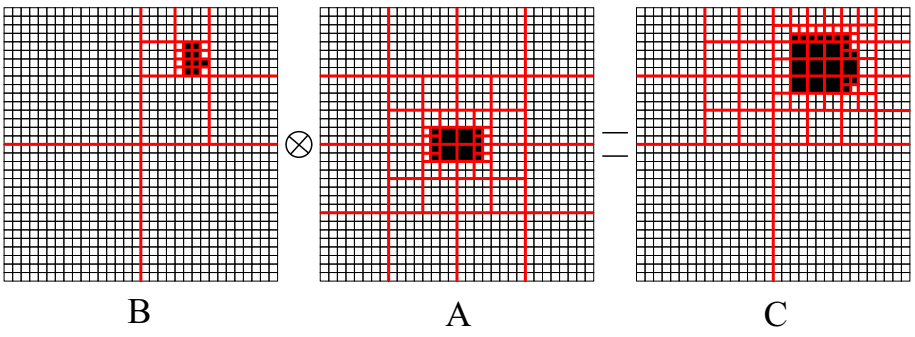


Figura 4.2: Convolución discreta de dos funciones

En la Figura 4.2 están representados las dos funciones que convolucionan en modo matricial, para un nivel de resolución dado. Realizando la convolución discreta de forma clásica se obtendrá el resultado de la matriz que representa la función C.

En la figura, se muestra también, mediante líneas rojas, como quedarían todas las representaciones matriciales si se hubiera utilizado una representación de *quadtrees*. Así, se pretende ilustrar cómo todas las funciones pueden representarse con diferente nivel de resolución. Por lo tanto, el planteamiento de evaluar regiones con diferentes niveles de resolución parece apropiado, puesto que la función C permanece constante en grandes intervalos.

4.5 Consideraciones finales

En este capítulo se ha propuesto un método para la realización de la convolución de dos funciones de forma jerárquica a través de convoluciones locales con diferentes grados de resolución.

Un concepto clave es el de conjunto invariante, permitiendo que según se aumenta la resolución, la región a evaluar sea cada vez más pequeña. Además, estos conjuntos invariantes están relacionados directamente con los *quadtrees*.

Por otro lado, el principio de localidad, que permite que no sea necesario el conocimiento de la totalidad de las funciones para evaluar una región de la convolución, impone un conjunto de restricciones que es necesario tener en cuenta cuando se pretenda utilizar para alguna aplicación.

Todo ello se ha realizado con el suficiente rigor matemático que soportará la calidad de las aplicaciones que se realicen con este planteamiento.

5

Obtención del C-espacio Utilizando la Convolución jerárquica

Según se vio en los antecedentes, en [Bro89] se propone la evaluación del C-espacio como la convolución de dos funciones, una que representa el espacio de trabajo y la otra a un robot móvil. Posteriormente, [Kav95] propone la evaluación de dicha convolución de forma discreta mediante el uso de la FFT para robots móviles. En trabajos más recientes, [CVM98] se formaliza matemáticamente este punto de vista, permitiendo también la obtención del C-espacio de manipuladores como una convolución discreta.

Estos métodos para la evaluación del C-espacio como una convolución discreta tienen una eficiencia comprobada y producen excelentes resultados. Al ser métodos discretos, es factible el uso de la convolución jerárquica para la evaluación de la convolución.

Así, en este capítulo se va a realizar una primera aplicación de la convolución jerárquica para la evaluación del C-espacio de un robot móvil sin la posibilidad de girar. Este caso pretende servir como un primer paso para poder aplicar la convolución jerárquica a casos más complejos.

5.1 Comprobación de condiciones previas

Para que la convolución jerárquica pueda ser aplicada a la evaluación de un C-espacio es necesario comprobar que las funciones que representan al robot y al espacio de trabajo cumplen las condiciones impuestas en los Lemas en los que se basa la convolución jerárquica.

Consideremos que la función que representa al robot es A, y B la que representa al espacio de trabajo. Ambas funciones toman valor 1 en los puntos que pertenecen al robot o a los obstáculos, respectivamente, y 0 en caso contrario.

La condición que expresa el Lema 4.1 indica que la función A tiene únicamente valores distintos de 0 en una región limitada. Si la función A representa al robot cumplirá esta condición puesto que el robot tiene dimensiones finitas.

También es necesario comprobar que se cumplen las condiciones del Lema 4.2. La primera de ellas es que $A(0,0) = 1$, este punto corresponderá con el lugar donde se fija el sistema de referencia asociado al robot. Como la forma habitual de elegirlo corresponde con un punto del propio robot, entonces se cumple la condición.

La segunda de las condiciones indica que el conjunto de los elementos de A con valor 1 debe ser conexo. Esta condición la cumplirá el robot, puesto que si no fuera conexo se trataría de dos piezas independientes sin ninguna unión entre ellas, y por lo tanto sería imposible moverse conjuntamente.

Puesto que hemos comprobado que se cumplen las condiciones impuestas para la aplicación de la convolución jerárquica estamos en disposición de aplicarla para la evaluación del C-espacio.

5.2 Descripción del algoritmo

En el método de la convolución jerárquica expuesto en el capítulo previo no se hacía ninguna referencia expresa a la estructura utilizada para la representación de las funciones que convolucionan. Únicamente era necesario conocer las funciones que convolucionan con diferentes niveles de resolución, y contar con una estructura para poder almacenar el resultado evaluado en cada paso, y que de igual modo permita utilizar diferentes niveles de resolución para cada región. Sin embargo, se consideró la posibilidad de utilizar representaciones de las funciones en forma de *quadtrees*, puesto que permiten conocer la función con cualquier nivel de resolución.

En este apartado se desarrollará el algoritmo para evaluar el C-espacio aplicando la convolución jerárquica mediante el uso de representaciones con *quadtrees* tanto para el espacio de trabajo y el robot, como para almacenar el resultado del C-espacio.

El algoritmo comenzará con el nivel de resolución más bajo, 1, con el que se representará el espacio de trabajo y el robot. A este nivel de resolución se realizará la convolución para obtener el resultado del C-espacio a resolución 1.

Una vez calculado el C-espacio es necesario encontrar el subconjunto invariante $\mathbf{C_1^1}$ y su complemento $\overline{\mathbf{C}}_\mathbf{1}^\mathbf{1}$. El conjunto $\mathbf{C_1^1}$, por el Teorema 4.1, está constituido por todos aquellos pixels de valor 0 cuyos 4-vecinos en la dirección sur, oeste y suroeste también han sido evaluados con valor 0 (de manera que se cumple la condicion de invarianza del Teorema 4.1). Al ser un conjunto invariante, representa una zona del *quadtree* del C-espacio a nivel de resolución 1 que no será necesario dividir, y, por lo tanto, se puede incluir como resultado en el primer nivel del *quadtree* que representa el C-espacio. Así, estos nodos serán hojas del árbol.

El conjunto complemento $\overline{\mathbf{C}}_\mathbf{1}^\mathbf{1}$ estará integrado por todos aquellos nodos que no se consideraron invariantes. Este subconjunto será necesario calcularlo con mayor resolución, por lo que en la representación en el *quadtree* los nodos correspondientes se dividirán para poder representarlos con mayor resolución.

Definición 5.1
Denominaremos región de interés (RDI) a un nivel de resolución a la región representada por el conjunto de pixels no invariantes evaluados a dicha resolución.

En el ejemplo de la figura 5.1, se puede observar que en el primer paso el subconjunto invariante $\mathbf{C_1^1}$ no contiene ningún elemento, y por lo tanto su complemento $\overline{\mathbf{C}}_\mathbf{1}^\mathbf{1}$ es todo el espacio. De este modo la RDI es todo el espacio.

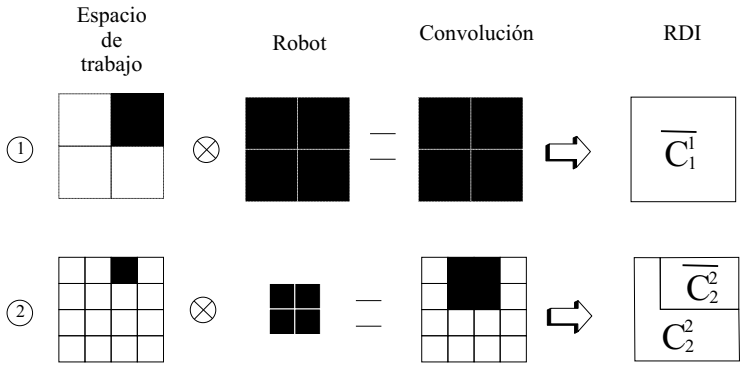


Figura 5.1: Representación de los conjuntos considerados en los dos primeros pasos del algoritmo

En la siguiente iteración del algoritmo será necesario calcular el subconjunto $\overline{\mathbf{C}}_\mathbf{1}^\mathbf{1}$ con un nivel de resolución 2, que denominaremos $\overline{\mathbf{C}}_\mathbf{1}^\mathbf{2}$. Por el Lema 4.2 la región de interés en el espacio de trabajo coincide con la del C-espacio $\overline{\mathbf{W}}_\mathbf{1}^\mathbf{2} = \overline{\mathbf{C}}_\mathbf{1}^\mathbf{2}$. Realizando la convolución con el robot (también representado a nivel de resolución 2) se obtiene el subconjunto $\overline{\mathbf{C}}_\mathbf{1}^\mathbf{2}$. De este subconjunto ya calculado, se podrán obtener a su vez el subconjunto invariante $\mathbf{C_2^2}$ y el no invariante $\overline{\mathbf{C}}_\mathbf{2}^\mathbf{2}$ siguiendo con el mismo criterio que anteriormente.

De forma análoga al paso anterior, el conjunto $\mathbf{C_2^2}$ puede pasarse al *quadtree* resultado, ahora aportando información al segundo nivel del *quadtree*. Y $\overline{\mathbf{C}}_\mathbf{2}^\mathbf{2}$ (nueva RDI) tiene que ser evaluado a resolución 3, $\overline{\mathbf{C}}_\mathbf{2}^\mathbf{3}$. Por lo que será necesario determinar el subconjunto del espacio de trabajo $\overline{\mathbf{W}}_\mathbf{2}^\mathbf{3} = \overline{\mathbf{C}}_\mathbf{2}^\mathbf{3}$ necesario para calcularlo. En el ejemplo de la figura 5.1 se puede observar que en este segundo paso la RDI se reduce y deja de representar todo el espacio

Se seguirán realizando estas operaciones de forma iterativa, hasta que se

calcule el subconjunto $\overline{\mathbf{C}}^{\mathbf{n}}_{\mathbf{n-1}}$. Ahora, por definición este subconjunto será invariante hasta n ($\overline{\mathbf{C}}^{\mathbf{n}}_{\mathbf{n-1}} = \mathbf{C}^{\mathbf{n}}_{\mathbf{n}}$), por lo que se pasará directamente al *quadtree* del C-espacio. En este último paso se aportará al C-espacio información en el nivel n.

El C-espacio así obtenido es válido hasta el nivel de resolución n, puesto que el último conjunto evaluado solamente era invariante hasta n.

Una muestra del algoritmo en pseudo código se muestra en el Algoritmo 5.1.

Algoritmo 5.1

```
quadtree del espacio = obtener espacio()
RDI = todo el espacio
quadtree del robot = obtener robot()
quadtree del C-espacio = nuevo()
para i = 1 hasta i =resolución total
  matriz del espacio = matriz del quadtree del espacio de RDI a resolución i
  matriz del robot = matriz del quadtree del robot a resolución i
  resultado = convolucionar(matriz del espacio, matriz del robot)
  si i = resolución total entonces
    invariante = resultado
  si no
    invariante = obtener invariante(resultado)
    no invariante = resultado\invariante /*complemento relativo a*/
    RDI = determinar región de interés(no invariante)
  pasar información de invariante al quadtree del C-espacio
```

5.2.1 Ejemplo

Para comprender el algoritmo con mayor claridad vamos a ilustrarlo con un ejemplo. En concreto vamos a considerar como espacio de trabajo y robot las funciones representadas en la figura 5.2, B y A, respectivamente. Estas funciones están discretizadas y representadas en forma matricial, de modo que se puede realizar la convolución discreta de matrices para obtener la correspondiente función C que representa el C-espacio. Además de esta representación matricial, en la figura se ha representado, mediante lineas rojas, la descomposición de las

matrices en *quadtrees* de tal modo que se pueda seguir con mayor facilidad la explicación del algoritmo. Por añadidura, al representar en forma de *quadtree* el C-espacio calculado se puede comparar con el resultado que se obtenga de aplicar el algoritmo.

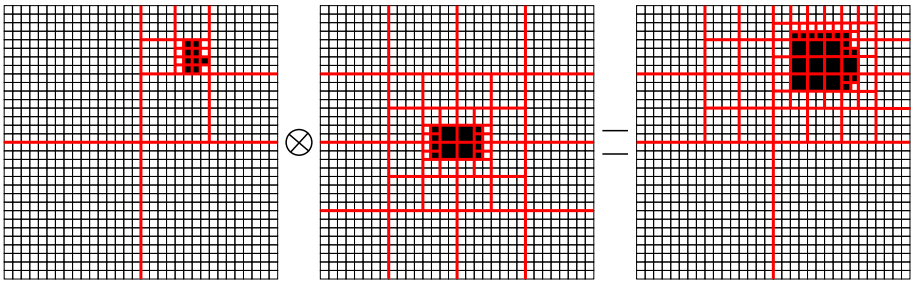


Figura 5.2: Cálculo del C-espacio mediante convolución

En la primera iteración del algoritmo, se considera la resolución 1, por lo que tanto el espacio de trabajo como el robot se representan mediante matrices 2×2. Al realizar la convolución (figura 5.3①) se obtiene que todo el C-espacio a esta resolución es susceptible de ser ocupado. Como no hay ningún pixel libre la región invariante es el conjunto vacío ($\mathbf{C_1^1} = \emptyset$). Por lo tanto, el conjunto no invariante o RDI que hay que evaluar a mayor resolución es el C-espacio completo ($\overline{\mathbf{C}}_\mathbf{1}^\mathbf{1} = C$). Antes de proceder con la siguiente iteración se debe trasladar toda la información obtenida al árbol (figura 5.4①), donde se tendrá que el nodo raíz se ha dividido en sus cuatro hijos, pero como ninguno era invariante no serán nodos hoja sino que podrán dividirse en la siguiente iteración. Por esta razón estos nodos se han representado en la figura 5.3① mediante un círculo.

En la segunda iteración (figura 5.3②) se tiene que $\overline{\mathbf{C}}_\mathbf{1}^\mathbf{2}$ se representa con una matriz 4×4. Para realizar la convolución hay que utilizar la representación del robot también a resolución 2 (donde únicamente se han representado los pixels no vacíos), y como resultado se obtiene una matriz 4×4 (en la figura se han marcado en color negro los pixels ocupados, y en color gris y blanco los no ocupados[1]). Como en el paso anterior se tienen que buscar los pixels invariantes, que serán aquellos no ocupados y cuyos vecinos en el lado izquierdo,

[1]Se ha hecho una distinción entre los pixels no ocupados para diferenciar aquellos que

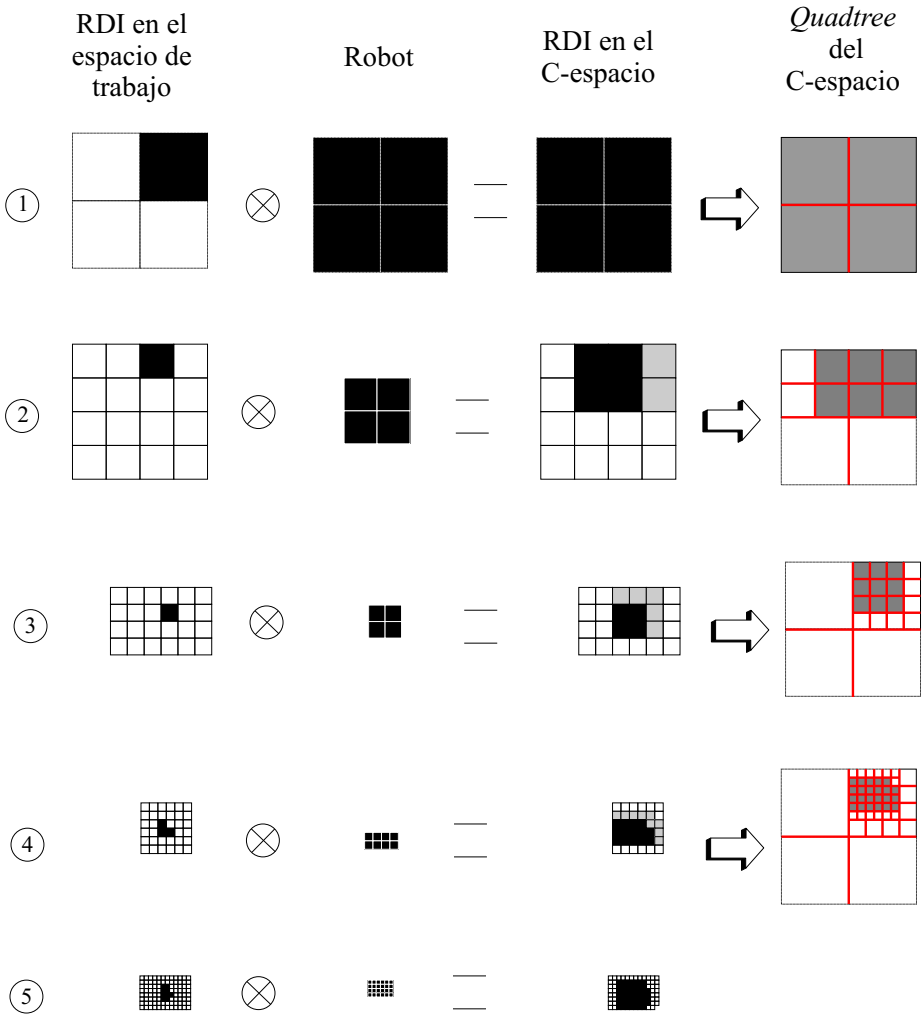


Figura 5.3: Ejemplo de ejecución del algoritmo

inferior y en la esquina inferior izquierda tampoco están ocupados (en la figura son los marcados en color blanco). En la siguiente iteración del algoritmo se deberán evaluar los no invariantes (pixels en color negro y gris), que son los que determinan la RDI.

Al introducir la información obtenida en la estructura de árbol (figura 5.4②), hay que tener en cuenta que en dos de los nodos que antes podían tener descendencia, todos sus hijos son invariantes, por lo cual para mantener una estructura

son invariantes (blancos) y los que no los son (grises) de modo que se puedan separar en el siguiente paso del algoritmo.

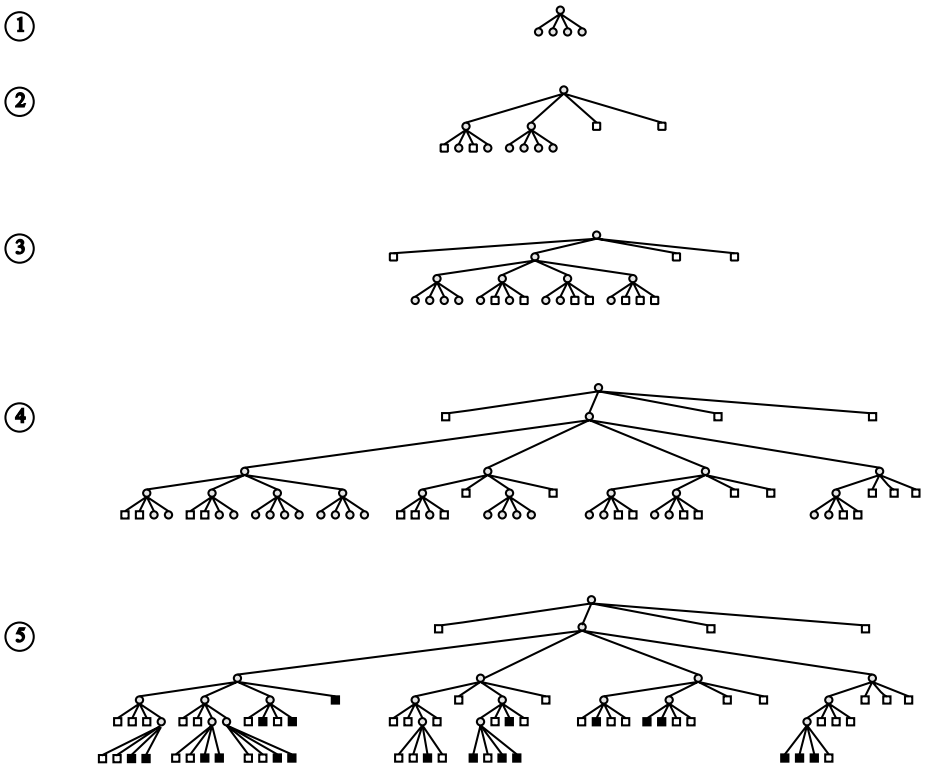


Figura 5.4: Crecimiento del árbol en cada paso del algoritmo. Se representan mediante ○ los nodos que deben ser divididos para alcanzar mayor resolución, y con □ los que ya no se deben dividir más

real de *quadtree*, esos dos nodos no se dividen y pasan a ser hojas. En la figura 5.4② estos dos nodos del nivel anterior se han representado por lo tanto mediante un cuadrado. De los nuevos nodos introducidos en el nivel que se está evaluando algunos están representados mediante cuadrados y círculos, indicando si son invariantes (nodos hoja, por lo tanto) o no, respectivamente.

Al llegar a la tercera iteración, ya no se necesita evaluar el C-espacio al completo, sino únicamente la RDI. Y solamente es necesaria la misma región del espacio de trabajo (a nivel de resolución 3) que la que se va a evaluar del C-espacio. En la figura 5.3③ se ve como ahora el espacio de trabajo necesario se representa con una matriz 4×6 en vez de la matriz 8×8 que correspondería para representar todo el espacio. De forma análoga a los pasos anteriores se realiza la convolución con el robot, trasladándose después el C-espacio local calculado a

la representación en forma de *quadtree* del C-espacio total. El algoritmo seguirá como en los pasos anteriores hasta llegar a la mayor resolución posible (iteración 5).

Al finalizar la quinta iteración, como no será posible continuar puesto que se desconoce el espacio de trabajo a mayor resolución, el resultado de la convolución se trasladará directamente al *quadtree*, donde todos los nuevos nodos introducidos serán hojas.

5.3 El interior de los obstáculos

De la misma forma que se puede eliminar de los cálculos parte de las regiones libres mediante las regiones invariantes, también es de gran interés eliminar aquellas zonas que pertenecen al interior de los obstáculos, y así limitar enormemente la región sobre la que es necesario realizar la convolución.

Analizando el C-espacio desde el punto de vista conceptual, los límites de los C-obstáculos se pueden encontrar, según se vio en apartado 2.1.2, mediante el desplazamiento del robot por los límites del obstáculo. Si el origen del sistema de referencia fijo al robot se encuentra en su interior, los C-obstáculos incluirán los obstáculos. Y por lo tanto, el C-obstáculo está realmente determinado por los límites del obstáculo en el espacio de trabajo, y no por su interior.

Desde el punto de vista de la convolución discreta, si los pixels $A(0,0) = 1$ y $B(i,j) = 1$, el correspondiente elemento de la convolución $C(i,j)$ al menos tiene un término del sumatorio distinto de cero, por lo que $C(i,j) \neq 0$. Vamos a analizar qué ocurre al aumentar la resolución cuando el pixel $B(i,j) = 1$ está en el interior del obstáculo o en su frontera.

Si la región que determina el pixel $B(i,j)$ está completamente ocupada por un obstáculo, los pixels que forman $B(i,j)$ a resoluciones mayores también lo están, por lo que también $C(i,j)$ estará completamente ocupado por un C-obstáculo a cualquier nivel de resolución. De esta forma $C(i,j)$ cumple la definición de invarianza.

Si $B(i,j) = 1$ pero no representa una región completamente ocupada por un obstáculo, es decir, se encuentra en una frontera del obstáculo y parte de la región está ocupada y parte libre, entonces, alguno de los pixels que lo forman a

menor resolución no estará ocupado. Por esta razón, no se puede asegurar que en el espacio de las configuraciones la región ocupada por $C(i, j)$ esté completamente ocupada por un C-obstáculo.

Por lo tanto, los pixels que están completamente ocupados por obstáculos son invariantes y no necesitan ser evaluados a mayor resolución.

Sin embargo, es necesario tener en cuenta que lo que determina los límites del C-obstáculo son las fronteras del obstáculo, por lo tanto éstas no deben ser eliminadas.

Teniendo en cuenta estas consideraciones, el algoritmo se puede modificar eliminando de la RDI en cada iteración todos los pixels que forman parte del interior de los obstáculos y no de su frontera. De este modo, la RDI se irá limitando únicamente a la franja alrededor de la frontera de los obstáculos donde aparecerá el C-obstáculo.

5.4 Uso de la FFT para evaluar la convolución

Puesto que la evaluación de la convolución discreta constituye un elemento fundamental es necesario determinar cómo se va a evaluar dicha convolución.

Una posibilidad sería utilizar la propia definición de convolución: evaluar en cada punto del resultado el sumatorio de los correspondientes productos. Este procedimiento, sin embargo, es poco eficiente puesto que es necesario evaluar múltiples sumas y productos para evaluar un único resultado. Por ejemplo, para calcular la convolución de dos matrices $N \times N$ se requiere un tiempo $O(N^4)$.

Otra posibilidad, es utilizar el teorema de convolución que permite evaluar la convolución como un producto simple utilizando la transformada de Fourier.

Teorema 5.1 (Teorema de Convolución)
Si dos funciones f y g definidas en R son integrables entonces

$$\mathscr{F}(f \otimes g) = \mathscr{F}(f) \times \mathscr{F}(g)$$

donde $\mathscr{F}$ representa la Transformada de Fourier, $\times$ el producto habitual de funciones y $\otimes$ el producto de convolución de funciones.

Aunque este teorema está enunciado para funciones continuas, también es válido para funciones discretas si se utiliza la transformada de Fourier discreta

(DFT). Por lo tanto se tendrá,

$$DFT(f \otimes g) = DFT(f) \times DFT(g)$$

por lo que realizando la transformada de Fourier inversa en los dos términos se tiene

$$f \otimes g = DFT^{-1}(DFT(f) \times DFT(g))$$

Cuando las matrices utilizadas para representar las funciones tienen tamaños potencia de 2, en vez del algoritmo de la DFT se puede utilizar el de la Transformada Rápida de Fourier (FFT) que es mucho más eficiente. Por ejemplo, para una matriz de dimensión $N \times N$, el cálculo de la DFT requiere un tiempo $O(N^4)$ y, sin embargo, si N es potencia de 2, la FFT lleva un tiempo $O(N^2 log_2(N))$. Sin embargo, también se puede utilizar la FFT cuando la dimensión de la matriz no es potencia de 2 con resultados intermedios [CT65].

Por lo tanto, para evaluar la convolución utilizaremos la FFT puesto reduce el tiempo de cálculo. Así, sera necesario evaluar la expresión siguiente

$$f \otimes g = FFT^{-1}(FFT(f) \times FFT(g))$$

Sin embargo, al utilizar la FFT se necesita que las funciones estén representadas mediante matrices rectangulares y además que dichas funciones sean periódicas. Estos condicionantes nos llevarán a proponer las soluciones que se presentan en los dos apartados siguientes.

5.4.1 Mínimo rectángulo contenedor

En este apartado vamos a estudiar qué zonas del C-espacio se van a evaluar utilizando la FFT para realizar la convolución. Teniendo en cuenta lo presentado en el apartado 5.2, se tiene que en cada iteración únicamente será necesario evaluar un conjunto de pixels (aquellos no invariantes). Para evaluarlos mediante la convolución, son necesarios los pixels que representan al robot y el mismo conjunto de pixels del espacio de trabajo.

En general, ese conjunto de pixels no invariantes no presentará una geometría rectangular. Sin embargo, para utilizar la FFT son necesarias matrices, y por lo tanto regiones rectangulares.

Definición 5.2
Denominaremos *mínimo rectángulo contenedor* (*mrc*) de un conjunto de pixels C, al menor de los rectángulos que contienen todos los pixels del conjunto.

De este modo, para la evaluación de la convolución de la RDI se puede considerar su *mrc*. Esto es válido puesto que según se vio en el Lema 4.2 el valor de los pixels fuera de la RDI no afectan al resultado de la convolución.

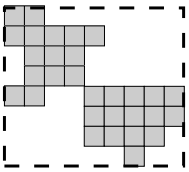


Figura 5.5: *mrc* (líneas discontinuas) de un conjunto **C** de pixels (gris)

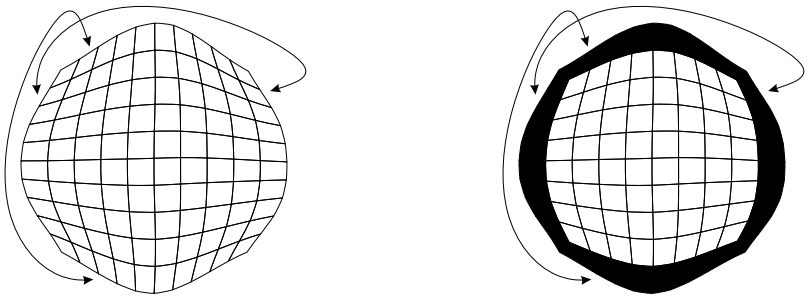
5.4.2 Matrices cíclicas

El hecho de utilizar la FFT para evaluar la convolución, además de requerir el uso de regiones rectangulares, presenta una característica que es necesario tener en cuenta. En concreto, el algoritmo de la FFT requiere que las funciones sean periódicas, lo cual no se cumple en el caso considerado.

Si se considera que las matrices que se van a someter a la FFT caracterizan a una función periódica sería equivalente a pensar que la matriz es cíclica. De una forma más gráfica (figura 5.6(a)), salir por la frontera derecha implica volver por la frontera izquierda y viceversa. Lo mismo sucede con el norte y el sur.

En el cálculo del C-espacio esta periodicidad provocaría que los C-obstáculos que quedaran fuera del C-espacio considerado se introdujeran dentro por efecto de la periodicidad (figura 5.7).

Este problema ya lo consideró Kavraki en su trabajo [Kav95]. Para solventar este inconveniente propone que el espacio de trabajo del robot debe estar delimitado (figura 5.6(b)), es decir, los pixels de las fronteras en el espacio de trabajo tienen que tomar valor 1 (en negro en la figura 5.6(b)). De este modo



(a) (b)

Figura 5.6: (a) Matriz cíclica. (b) Propuesta de Kavraki

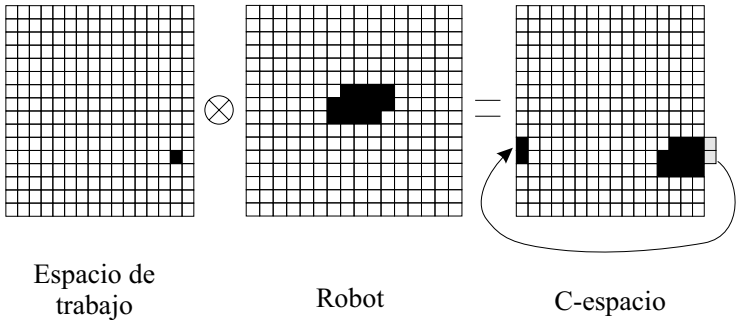


Figura 5.7: Efecto de la periodicidad al evaluar la convolución mediante la FFT

se garantiza que el robot no pueda salirse de la región por ninguna frontera, y por lo tanto se elimina cualquier problema derivado de la periodicidad.

En nuestro caso, no es posible utilizar la solución propuesta por Kavraki, puesto que las convoluciones se han de realizar sobre determinadas regiones dentro del espacio de trabajo completo. Estas regiones no pueden delimitarse con fronteras, puesto que el robot se moverá por todo el espacio y no se limitará únicamente a estas zonas locales. Sin embargo, es necesario proponer alguna medida para que no aparezcan C-obstáculos por efecto de la periodicidad.

La solución que proponemos consiste en introducir un marco vacío (relleno de 0s o pixels libres de obstáculos) al espacio de trabajo. De tal modo que todos aquellos C-obstáculos situados fuera de la región evaluada, entren dentro del marco, por lo que se evita que se reintroduzcan por efecto de la periodicidad. Para que el marco pueda albergar todos los posibles C-obstáculos que quedan

fuera de la región, el grosor del marco en cada eje debe venir determinado por las dimensiones del robot en el correspondiente eje. Un ejemplo se muestra en la figura 5.8. Si al resultado de la convolución se le elimina el marco introducido anteriormente, el resultado corresponde con el esperado.

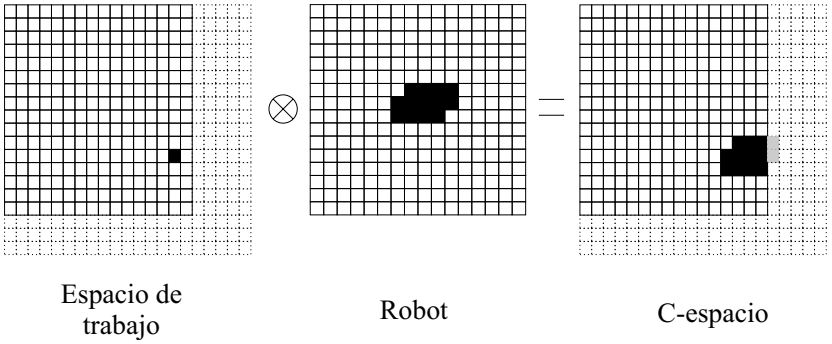


Figura 5.8: Eliminando el efecto de la periodicidad con un marco

5.5 Segmentación[2]

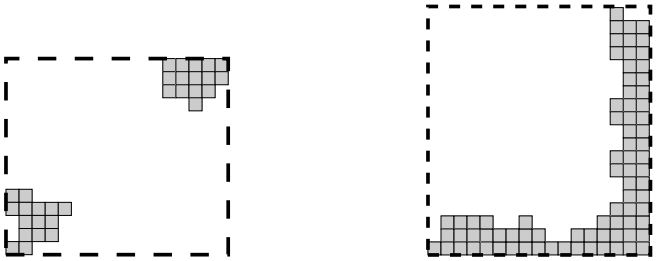
Una segunda dificultad derivada del uso de la FFT proviene de la necesidad de utilizar matrices, y por tanto regiones rectangulares para la evaluación de la convolución. Es decir, lo que anteriormente habíamos definido como *mínimo rectángulo contenedor*.

Al ser necesario realizar la convolución para el *mrc* de la región de interés, también se evalúan un conjunto de pixels (todos aquellos que no pertenecen a la RDI) que no era necesario evaluar, a los cuales denominaremos pixels innecesarios. En principio, esto no debería suponer un gran inconveniente debido a las ventajas que supone el uso de la FFT en cuanto al tiempo de cálculo. Pero en ciertas situaciones la cantidad de pixels innecesarios en la RDI puede hacer que el uso de la FFT sea ineficiente.

Vamos a ilustrarlo con los ejemplos de la figura 5.9. En ella, los pixels en gris representan la RDI. Es necesario recordar que esta RDI se corresponde con

[2]Una segmentación de imágenes consiste en la división de una imagen en un conjunto de regiones no superpuestas cuya unión es la imagen completa. El propósito de la segmentación es la descomposición de la imagen en partes con un significado respecto a una aplicación particular. Del cap. 10 de [HS92]

la región del C-espacio que se desea evaluar, que está rodeada completamente por 0s (el complemento es la región invariante con valor 0). Al obtener el *mrc* de la RDI (en líneas discontinuas), es necesario incluir gran cantidad de pixels que no pertenecen a la RDI.



(a) (b)

Figura 5.9: Ejemplos en los que el mínimo rectángulo contenedor contiene un número muy elevado de pixels añadidos

La forma de evitar este inconveniente será dividir la RDI en trozos más pequeños de modo que sea posible evaluar cada fragmento por separado. Se consigue de ese modo minimizar la inclusión de regiones externas a la RDI.

Esta razón es la que nos ha motivado a definir este proceso como segmentación, aunque, como se analizará más adelante, no coincida completamente con la definición de segmentación. Sin embargo, el propósito es similar, puesto que se desea dividir la RDI en partes más pequeñas siguiendo ciertos criterios para optimizar la evaluación de la convolución utilizando la FFT.

5.5.1 Criterios de segmentación

En esta sección se va a determinar qué criterios se van a utilizar para determinar los diferentes trozos en los que se segmente la RDI. Para especificar estos criterios es necesario analizar en primer lugar qué es lo que se busca con la segmentación.

Con la RDI se había conseguido que únicamente fuera necesario utilizar la misma región del espacio de trabajo que la que se desea evaluar del C-espacio. Con la segmentación se desea dividir la RDI en partes más sencillas, donde se siga cumpliendo la premisa anterior, es decir, solamente con cada una de esas partes en el espacio de trabajo se puede evaluar la región del C-espacio correspondiente.

Por lo tanto, es necesario determinar cuáles son las condiciones que se deben cumplir en la segmentación para que sea válida. Estas condiciones, y por tanto la forma de segmentar, van a depender de la naturaleza de la RDI, por lo que se va a clasificar en dos grupos. En primer lugar veremos el caso en el que la RDI está formada por diferentes conjuntos no 8-conexos entre sí, para pasar posteriormente al planteamiento para una RDI 8-conexa.

RDI no 8-conexa

Al tratarse de una región no 8-conexa (figura 5.9(a)), es posible identificar cada uno de los conjuntos aislados que forman la RDI (no están conectados entre ellos). Una vez que ya se tienen los diferentes fragmentos que forman la RDI, se puede comprobar que cada uno de ellos está rodeado completamente por pixels con valor 0 en el C-espacio. Esto es cierto puesto que lo estaba la RDI, y dado que los fragmentos no son 8-conexos entre sí, éstos también estarán rodeados de 0s.

Así, por el Lema 4.2 se puede evaluar el C-espacio correspondiente a cada fragmento de manera independiente, sin necesidad de conocer los valores de los pixels que pertenecen al resto de fragmentos. Y por lo tanto, para evaluar la convolución utilizando la FFT únicamente es necesario considerar los *mrc* para cada fragmento en el momento de evaluar su convolución.

El resultado de aplicar esta segmentación al ejemplo de la figura 5.9(a) se muestra en la figura 5.10.

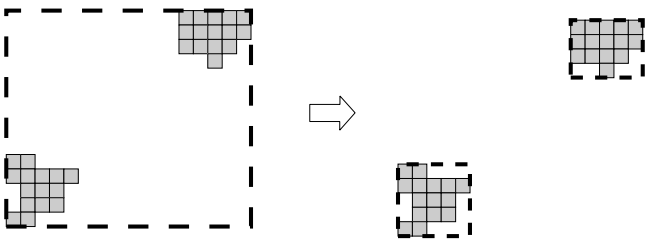


Figura 5.10: Segmentación de una RDI no 8-conexa

RDI 8-conexa

En este caso no es posible separar la RDI en fragmentos separados. Un ejemplo de este tipo es el mostrado en la figura 5.9(b), donde se puede observar que es imposible fragmentar la región, al ser conexa, en trozos aislados. Y, sin embargo, el *mrc* incluye gran cantidad de pixels innecesarios. Este tipo de RDIs se corresponden por lo tanto con RDIs convexas.

Antes de proponer una solución vamos a recordar cómo se había llegado a la RDI que se está considerando. Se había obtenido la RDI como la misma región en el espacio de trabajo que la deseada en el C-espacio, gracias a que se encontraba completamente rodeada por pixels con valor 0, aplicando el Lema 4.2.

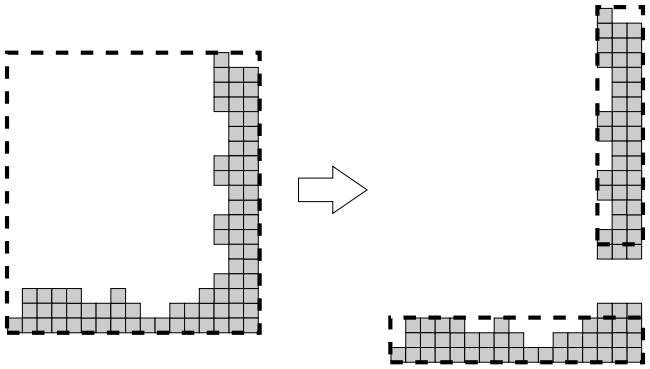


Figura 5.11: Segmentación no válida de una RDI 8-conexa

Si la región de C-espacio que se desea conocer es conexa, y se fragmenta, ninguno de los trozos cumplirá las condiciones del lema anterior, puesto que por alguna parte de su frontera el valor no será 0 (figura 5.11), de tal modos que no es posible identificar cada fragmento con una RDI. Por lo tanto, la segmentación planteada anteriormente no es válida, por lo que es necesario recurrir a otras formas de realizarla.

Para que se pueda identificar cada fragmento con una RDI se propone una segmentación en la que los diferentes fragmentos no son disjuntos[3], es decir, están solapados. De este modo, cualquier pixel estará en una de estas dos

[3]La segmentación de imágenes supone la división en un conjunto de regiones no superpuestas, por lo que este proceso no coincide exactamente con una segmentación de imágenes. Aun así, seguiremos llamando a este proceso segmentación por las similitudes que presenta.

opciones: pertenece a uno solo de los fragmentos o está contenido en más de uno.

- **El pixel únicamente pertenece a un fragmento**

  Al pertenecer únicamente a un fragmento, el valor que toma el pixel en el C-espacio se obtiene al convolucionar el fragmento en el espacio de trabajo con el robot. Por lo tanto, todos los pixels del espacio de trabajo que influyan en el sumatorio de la convolución deben encontrarse en el mismo fragmento en el que se encuentra el pixel. Si no fuera así, pixels que influyen en el valor de la convolución no son considerados.

  Esta condición impone una restricción sobre el modo en que se debe realizar la segmentación, de modo que se debe garantizar que cumpla la condición anteriormente expuesta.

- **El pixel está contenido en dos o más fragmentos**

  Al realizar el sumatorio de la convolución es necesario considerar todos los pixels que influyen en el resultado (todos ellos contenidos en la RDI). Se podría pensar que, como en el caso anterior, al convolucionar cada fragmento, para obtener un valor correcto del pixel, el fragmento tiene que contener todos los pixels necesarios para evaluarlo. Según el Lema 4.2 esto sólo se puede asegurar si el fragmento está completamente rodeado por pixels desocupados. Sin embargo, es imposible escoger fragmentos que estén completamente rodeados por pixels desocupados puesto que la RDI era 8-conexa.

  En contra de lo que cabe esperar, esto no supone un gran problema si el conjunto de los fragmentos que contiene al pixel a evaluar, incluye a todos los pixels necesarios para su evaluación. Entonces, de una forma u otra todos los pixels que afectan al valor de la convolución han sido considerados.

  De esta forma, si en todos los fragmentos se ha evaluado el pixel con valor 0 entonces el valor para la convolución global es 0 puesto que todos los términos del sumatorio (cada uno evaluado en su correspondiente fragmento) tomaban ese valor. De igual modo el valor es distinto de 0 si lo es en alguno de los fragmentos.

Un ejemplo de partición se puede observar en la figura 5.12. Aquí, se ha segmentado la RDI en dos fragmentos, la parte inferior y la lateral. Si estos fragmentos cumplen las condiciones impuestas anteriormente para los pixels que pertenecen únicamente a un fragmento o a ambos, se puede realizar la convolución de forma independiente. Por lo tanto, será necesario considerar el *mrc* de cada uno de ellos por lo que se reduce el número de pixels innecesarios incluidos. Por supuesto, después de realizar la convolución el valor que toman los pixels comunes será cero si el calculado en cada uno de los fragmentos lo era.

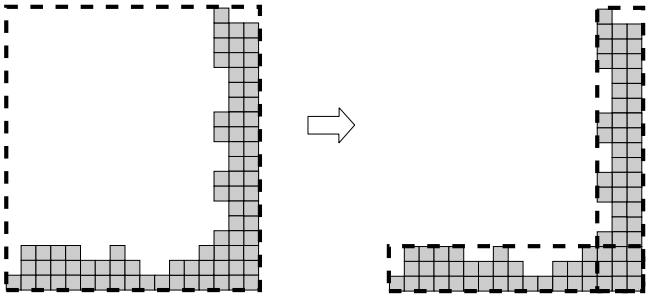


Figura 5.12: Segmentación de una RDI 8-conexa

5.5.2 Automatización de la segmentación

Existen una gran cantidad de técnicas para realizar la segmentación de imágenes ([GW87]) debido a que es un campo ampliamente estudiado en visión artificial. Sin embargo, todas ellas están enfocadas a la identificación de regiones con características diferenciadas. Y, en ningún momento se considera la posibilidad de que pueda haber superposición de las regiones.

Estas técnicas podrían ser aplicadas al caso de RDIs no conexas, pero no cuando se trate de RDIs conexas. Además, según se comentó en el apartado anterior, en este segundo caso, la segmentación requiere de una serie de condiciones que indican que se debe proponer una técnica de segmentación apropiada para esta situación. Por esta razón se ha desarrollado un método basado en la eliminación de las regiones innecesarias que se incluyen con el *mrc*.

5.5.2.1 Método de eliminación de regiones innecesarias

En esta sección comentaremos la filosofía general del método propuesto y en secciones sucesivas ahondaremos en sus aspectos claves.

Definición 5.3
Denominaremos *Máximo Rectángulo Innecesario* (*MRI*) de un *mrc*, al mayor de los rectángulos (de área máxima) dentro del *mrc* que únicamente contiene pixels innecesarios.

Utilizando el *MRI* es posible realizar una segmentación en cuatro fragmentos, donde únicamente intersectan 2 a 2. Los cuatro fragmentos se corresponden con las bandas horizontales superior e inferior al *MRI* y las verticales en los laterales. Un ejemplo se muestra en la figura 5.13, donde se tienen los cuatro fragmentos (① y ③ las bandas horizontales, ② y ④ las bandas verticales). Además, se puede observar que las intersecciones de los fragmentos se corresponden con las zonas correspondientes a los 4 vértices. En el caso de que alguno de los lados del *MRI* coincida con el del *mrc* el fragmento correspondiente no existirá.

A continuación, será necesario determinar el *mrc* de la porción de la RDI contenida en cada uno de los fragmentos.

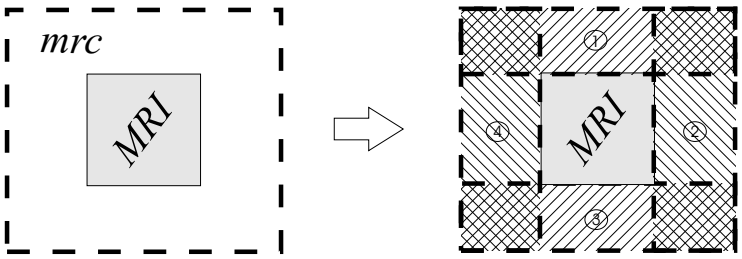


Figura 5.13: Segmentación utilizando el *MRI*

En las figuras 5.14 y 5.15 se muestra el proceso de segmentación propuesto aplicado a los ejemplos de la figura 5.9 contemplados anteriormente. En ellas se puede observar cómo en ciertas situaciones no es necesario considerar los cuatro fragmentos. En la figura 5.14 también se puede observar cómo solamente una parte de los fragmentos contienen RDI, por lo que es necesario determinar el *mrc*.

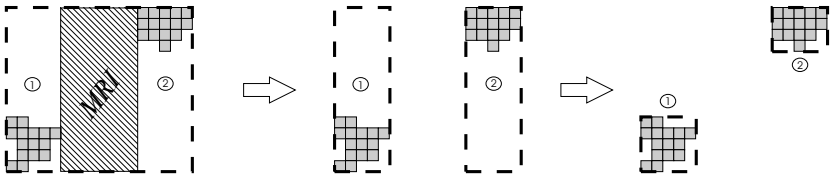


Figura 5.14: Procedimiento de la segmentación para el ejemplo de la figura 5.9(a)

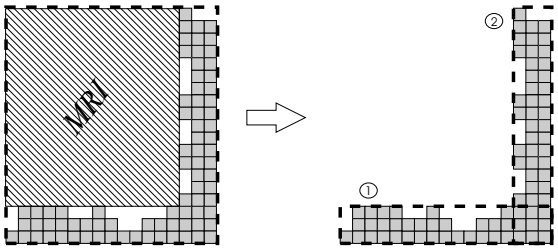


Figura 5.15: Procedimiento de la segmentación para el ejemplo de la figura 5.9(b)

Si consideramos el tercer ejemplo mostrado en la figura 5.16, se puede observar que la segmentación en cuatro fragmentos puede ser insuficiente. Es decir, alguno de los fragmentos obtenidos aún puede contener una región innecesaria cuya superficie sea elevada. En este caso, el procedimiento puede ser aplicado de forma recursiva sobre cada uno de los *mrc* obtenidos, eliminando de esta forma tantas regiones innecesarias como sea pertinente. Así, se puede ver en la figura 5.16 que aplicando el procedimiento de la segmentación una segunda vez se obtienen los resultados deseados.

5.5.2.2 Determinación del *MRI*

Como se ha descrito, encontrar el *MRI* consiste en elegir el mayor de los rectángulos que únicamente contienen elementos innecesarios. Sin embargo, ésta no es una tarea fácil, puesto que pueden existir infinidad de rectángulos contenidos, y, en el caso de una RDI suficientemente dispersa, distribuidos por diferentes posiciones.

Para determinar el *MRI* proponemos utilizar un método aleatorio, que consiste en generar un número determinado de semillas diseminadas de forma alea-

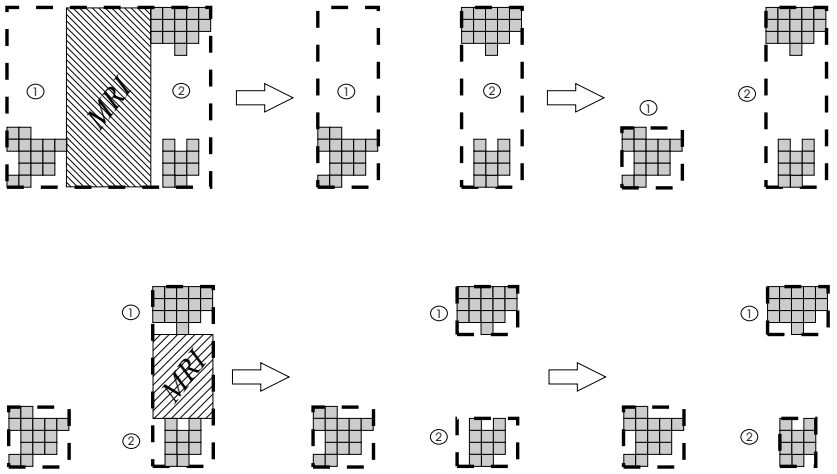


Figura 5.16: Segmentación recursiva

toria por el *mrc*. De estas semillas, únicamente serán válidas aquellas que se sitúen en pixels innecesarios. Todas las semillas válidas comenzarán a crecer, expandiéndose en forma de rectángulo hacia aquellas direcciones en las que puedan alcanzar una mayor área, con la condición de que únicamente contengan pixels innecesarios. De esta forma, escogiendo la semilla que mayor área ha alcanzado se puede obtener el *MRI*.

Evidentemente, al ser un método aleatorio no tiene por qué encontrarse el máximo de todos los posibles rectángulos. Pero si el número de semillas es suficiente se encontrará uno de los mayores. Lo cual no es un inconveniente puesto que simplemente se desea el mayor para obtener la mejor segmentación posible y eliminar el máximo de pixels innecesarios.

5.5.3 ¿Cuándo segmentar?

El criterio para decidir cuándo realizar una segmentación de la RDI va a ser un punto clave a la hora de conseguir un funcionamiento óptimo del algoritmo. De tal modo que, si los criterios son excesivamente restrictivos apenas se realizará la segmentación, con el inconveniente de que se van a realizar evaluaciones en lugares donde no sea necesario.

Por el contrario, si los criterios son muy amplios se tenderá a segmentar en un gran número de casos. Esto, aunque en principio pueda parecer beneficio-

so, puede acarrear grandes perjuicios. En primer lugar, realizar el proceso de segmentación puede suponer mayor esfuerzo que el que supondría evaluar los pixels innecesarios. Y en segundo lugar, es necesario considerar que al realizar la segmentación, si la RDI es conexa, se produce un solapamiento entre la regiones a evaluar. Realizar segmentaciones de forma masiva sobre RDIs que ya han sido segmentadas anteriormente acarrea que existan regiones que deban ser evaluadas múltiples veces (una región en la que se solapan varias RDIs) con el consiguiente perjuicio.

Por lo tanto, es necesario llegar a una solución de compromiso. También, se deben buscar criterios que se evalúen de una forma rápida, de tal modo que la comprobación para realizar la segmentación o no sea lo más rápida posible.

Se ha optado por considerar como criterio de segmentación el porcentaje de la región ocupada por los pixels innecesarios en el *mrc* de la RDI. De tal forma que, si el porcentaje de la región innecesaria supera un umbral se realizará la segmentación. Este es un procedimiento extremadamente sencillo que puede realizarse rápidamente.

5.6 Algoritmo completo

Si se añade el proceso de segmentación descrito anteriormente al Algoritmo 5.1 resulta el Algoritmo 5.2. En las líneas 1-5 se realiza la inicialización de variables. Es necesario destacar la aparición de las listas RDIs_i que representan el conjunto de RDIs que deben ser evaluadas en la iteración i. Esta lista aparece debido a que en el proceso de segmentación es necesario considerar cada una de las RDIs resultantes independientemente. No obstante, en la primera iteración solamente es necesario considerar una única RDI que es todo el espacio.

En la línea 6 se realiza la iteración donde se va refinando el nivel de resolución, mientras que la línea 7 se incluye para que se evalúe la convolución en todas las RDIs que se han fragmentado en las iteraciones anteriores. El resto del algoritmo es similar al mostrado en el Algoritmo 5.1 salvo en las líneas 19-22 donde se comprueba si la RDI obtenida para ser evaluada en la siguiente iteración debe ser segmentada o no según los criterios que se hayan establecido. En el caso de que se requiera la segmentación esta se realiza incluyendo cada

uno de los fragmentos resultantes como las RDIs que es necesario evaluar en la siguiente resolución.

Algoritmo 5.2 Algoritmo completo

```
1: quadtree del espacio = obtener espacio()
2: RDI = todo el espacio
3: RDIs_1 ← RDI
4: quadtree robot = obtener robot()
5: quadtree C-espacio = nuevo()
6: para i = 1 hasta i =resolución total
7:   para todo cada RDI de RDIs_i
8:     matriz del espacio = matriz del quadtree del espacio de RDI
9:                          a resolución i
10:    matriz del robot = matriz del quadtree del robot
11:                       a resolución i
12:    resultado = convolucionar(matriz del espacio, matriz del robot)
13:    si i = resolución total entonces
14:      invariante = resultado
15:    si no
16:      invariante = obtener invariante(resultado)
17:      no invariante = resultado\invariante /*complemento relativo a*/
18:      RDI = determinar región de interés(no invariante)
19:      si es necesario segmentar (RDI) entonces
20:        RDIs_{i+1} ← segmentar(RDI)
21:      si no
22:        RDIs_{i+1} ← RDI
23:    pasar información de invariante al quadtree del C-espacio
```

5.7 Consideraciones finales

Una de las principales diferencias entre el algoritmo completo y el básico aparece en las líneas 19-20, donde se realiza la segmentación en el caso de ser necesaria. Como se comentó en el apartado 5.5.3, el criterio que se va a utilizar para determinar si es necesario segmentar o no, es el porcentaje del *mrc* de la RDI

ocupado por pixels innecesarios. Si supera un determinado porcentaje umbral será necesario segmentar.

Por lo tanto ahora se tiene una situación de compromiso en el valor del porcentaje umbral. Este valor no debe ser fijo, puesto que la segmentación también debe estar de acuerdo con la dispersión de la RDI. No es lo mismo una RDI completamente agrupada donde con la segmentación se puede eliminar gran parte de los pixels innecesarios que una muy dispersa donde la segmentación no elimina apenas las zonas innecesarias. Indudablemente, la dispersión de la RDI estará asociada a la dispersión del espacio de trabajo, puesto que espacios de trabajo dispersos producirán RDIs dispersas y espacios de trabajo organizados RDIs agrupadas. Por lo tanto tenemos con el porcentaje umbral un parámetro de sintonización del algoritmo dependiendo del espacio que se esté considerando en cada momento.

En cuanto a la línea 20 del algoritmo, donde se realiza la segmentación, también es necesario realizar una serie de consideraciones. El procedimiento propuesto para realizar la segmentación estaba basado en la determinación del *MRI* a partir de un conjunto de semillas lanzadas al azar. El número de semillas que se utilice tampoco debe ser fijo, puesto que para una RDI dispersa será mucho más complicado encontrar el *MRI* que para una RDI agrupada. Por lo tanto, el número de semillas a utilizar también es un parámetro que es necesario determinar para un correcto funcionamiento del algoritmo.

Como se ha explicado, estos dos parámetros (porcentaje umbral y número de semillas) no se pueden determinar con un carácter general, puesto que sus valores óptimos dependen de la RDI a considerar, y en concreto de su dispersión.

Puesto que la RDI está directamente relacionada con el C-espacio, ya que es la región no invariante, la medida de la dispersión se puede realizar sobre el C-espacio. Pero, el C-espacio es lo que se está evaluando, por lo que es desconocido a priori. Por otro lado, la dispersión en el C-espacio está íntimamente relacionada con la dispersión en el espacio de trabajo, que sí es conocida a priori. De este modo se debe establecer una dependencia funcional entre la dispersión del espacio de trabajo y los valores óptimos de sintonización de la segmentación.

6

C-espacio de un Robot Móvil con Giros

Cuando se considera un robot móvil con la posibilidad de girar, según se comentó en el apartado 2.1.3, la dimensión del C-espacio aumenta para introducir una nueva variable en la configuración correspondiente a la orientación. De este modo, una configuración del robot viene definida por la tupla (x_r, y_r, θ_r), donde (x_r, y_r) determinan (igual que cuando no se consideran giros) la posición de un punto fijo del robot respecto un sistema de referencia asociado al espacio de trabajo y θ_r la orientación.

Si se plantea la evaluación del C-espacio como una convolución discreta [Kav95] se tiene que

$$C(x_r, y_r, \theta_r) = \sum_{i,j} W(i,j) A'_{\theta_r}(x_r - i, y_r - j) = W \otimes A'_{\theta_r}(i,j)$$

donde W es la matriz que representa al espacio de trabajo y A'_{θ_r} al robot en la orientación θ_r. En este caso, las únicas variables que convolucionan son las de posición (x, y), mientras que la orientación es independiente.

Debido a la independencia de la orientación, algunos autores proponen la discretización del eje correspondiente a la orientación en el C-espacio (entre ellos [Kav95]). De tal manera que, se puede evaluar el C-espacio completo como un conjunto de planos con orientación constante. Para la evaluación de cada uno de estos planos se puede aplicar el mismo procedimiento que para la evaluación del C-espacio de un robot móvil sin giros.

Teniendo en cuenta esto, se puede aplicar la convolución jerárquica de la misma forma que se propuso en el capítulo 5, realizándola sobre cada una de las orientaciones. El resultado es una representación de *quadtree* para cada una de las orientaciones consideradas.

Sin embargo, puesto que las variaciones entre la representación del robot para dos orientaciones consecutivas van a ser pequeñas, también lo serán las variaciones que se produzcan entre dos planos consecutivos en el C-espacio. Por lo tanto, al ser nuestro objetivo la utilización de estructuras jerárquicas, es posible la agrupación de los datos en la dirección angular, obteniéndose una estructura de *octree*, apropiado para representar un espacio tridimensional. De este modo se compactarán en gran medida los datos resultantes.

Una posibilidad sería construir el *octree* una vez que se han obtenido cada uno de los planos del C-espacio en forma de *quadtree*. Así, se pueden ir combinando parejas de planos para agrupar regiones similares en ambos, formando una estructura *d-slice*. Y luego, de forma reiterativa combinar parejas de los *d-slices* obtenidos anteriormente [SJL94] para obtener uno que incluya ambos. Un ejemplo ilustrativo se muestra en la figura 6.1.

Otra posibilidad es integrar la variable que no convoluciona en el algoritmo que evalúa el C-espacio de forma jerárquica. De modo que, el refinamiento se realice también en la orientación a medida que se alcanza una mayor resolución. Así, según se tiene mayor precisión en las variables espaciales, también se adquiere respecto al espacio que ocupa el robot en una determinada orientación.

Esta última posibilidad es la que se va a plantear en este capítulo, debido a que se pueden ir eliminando regiones que quedarán vacías para grandes intervalos de ángulos, por lo que se obtendrá una mayor reducción de memoria, ya que no es necesario evaluarlo para cada uno de los ángulos. La utilización de los

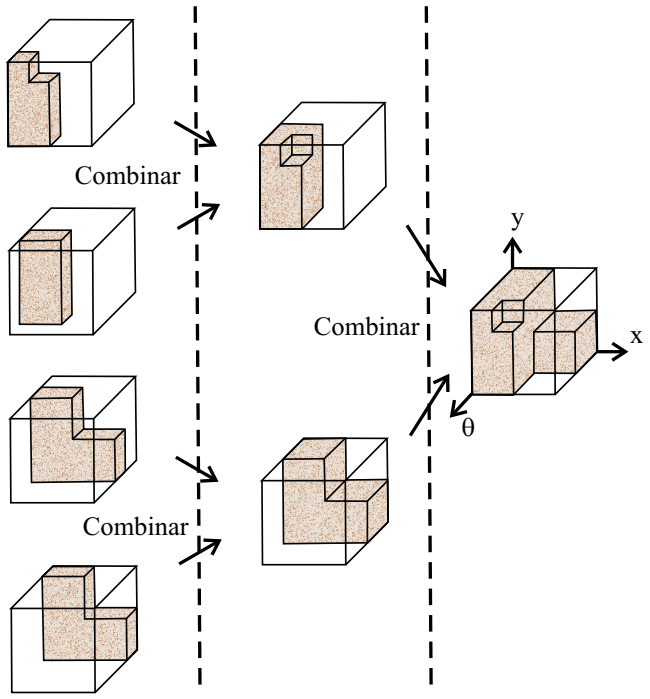


Figura 6.1: Combinación de *quadtrees* para producir un *octree*

d-slices sería similar a, cuando se calcula el C-espacio de un robot móvil sin giros, realizar la convolución en forma matricial y luego calcular el *quadtree*.

En el siguiente apartado se determinará cuál es la estructura de datos más apropiada para la representación del C-espacio, teniendo en cuenta que el refinamiento en el ángulo se pretende realizar junto con las variables espaciales. Posteriormente, se procederá a describir el modo de introducir la nueva dimensión en el procedimiento de evaluación del C-espacio.

6.1 Definición de la estructura de datos

Antes de comenzar a definir la estructura de datos se necesario determinar las características de las funciones que van a intervenir en la evaluación del C-espacio:

- El espacio de trabajo es un espacio bidimensional con las coordenadas (x, y) y por lo tanto representado mediante un función bidimensional

- El robot es una región bidimensional (x, y) para cada ángulo (θ), representado por una función bidimensional. Sin embargo, como se requiere una función para cada valor de ángulo, éste se puede introducir en la función resultando una función tridimensional con las variables (x, y, θ)
- El C-espacio es un espacio tridimensional con las coordenadas (x_r, y_r, θ_r), por lo tanto se representa mediante una función tridimensional

El espacio de trabajo, al representarse mediante una función bidimensional, se puede representar en forma discreta mediante *quadtrees*. En cuanto al robot y al C-espacio, al ser tridimensional, la estructura que parece más apropiada para su representación es el *octree*. Así, se puede obtener una descomposición regular en todas las coordenadas, por lo que se pueden obtener diferentes niveles de discretización en cualquiera de ellas.

Sin embargo, la estructura de *octree* impone que se realice el mismo número de descomposiciones en los tres ejes. Esto es útil cuando los tres ejes representan variables de la misma naturaleza, y además, abarcan intervalos semejantes, como sería el caso de tres coordenadas espaciales. En nuestro caso, mientras que dos ejes corresponden con coordenadas espaciales, el tercero corresponde con una coordenada angular, y por lo tanto limitada al intervalo $[0, 2\pi]$. Si la superficie por donde se mueve el robot es suficientemente grande, para alcanzar una precisión aceptable es posible que se necesite llegar a un nivel de resolución elevado. Pero, ese nivel de resolución puede llegar a no tener ningún sentido en la coordenada angular, puesto que se llegarían a diferenciar orientaciones que sería imposible distinguirlas con los sistemas de percepción del robot.

Así, por ejemplo, supongamos que se desea representar el C-espacio correspondiente a un entorno de trabajo de dimensiones $100 \times 100 m^2$ con una precisión de al menos $1,5cm$. Entonces se necesitaría una resolución de 8.192×8.192. Utilizar la misma resolución en la coordenada angular llevaría a distinguir entre diferencias de $7,7^{-4} rad = 0,044°$, lo cual un sistema robótico es incapaz de diferenciar.

Es necesario, por lo tanto, utilizar algún tipo de estructura que permita trabajar con diferentes niveles de resolución para las coordenadas espaciales que para la angular. La estructura que se ha elegido es una estructura híbrida de *octree* y *quadtree*.

6.1.1 Árbol multigrado-2^k

Proponemos utilizar una estructura, que hemos diseñado en este trabajo denominada árbol "multigrado-2^k", y que es similar a la propuesta en [SJL94] (*d-slice*), aunque la finalidad es completamente distinta, como se comentará a continuación.

La estructura de árbol multigrado-2^k tiene como característica especial que permite trabajar con espacios de dimensión n y con diferentes niveles de resolución para cada una de las dimensiones o ejes del espacio. Para ello, cada dimensión debe subdividirse un número de veces distinto al resto. Para conseguirlo se utilizará una estructura de árbol con diferentes grados. Es decir, cada uno de los nodos que forma el árbol se dividirá en un número distinto de hijos dependiendo del nivel en el que se encuentre dentro del árbol.

Puesto que se desea una estructura que represente una descomposición regular del espacio, cada vez que se realiza una subdivisión en una de las dimensiones ésta debe generar dos trozos del mismo tamaño. Por esta razón, el grado del árbol en cada nivel no puede ser cualquiera, sino que tiene que ser una potencia de 2. De este modo, el grado en un nivel será 2^k, siendo k el número de dimensiones que se subdividen en dicho nivel.

Para que esta estructura pueda reconstruir el espacio, es necesario que incluya información de las dimensiones que se subdividen en cada uno de los niveles, de modo que sea posible determinar qué región del espacio representa cada nodo.

Por otro lado, para evitar la necesidad de incluir toda esa información y para conseguir que el árbol sea más organizado es conveniente optar por sistematizar las divisiones, de modo que el grado del árbol disminuya según se profundiza en el árbol. Así, en los primeros niveles se dividirán todas las dimensiones, hasta llegar a un nivel en el que alguna no necesite dividirse más. En este punto el grado para ese nivel se reducirá, dividiéndose el resto de dimensiones, hasta llegar de nuevo a un nivel en el que otra dimensión alcance su máximo nivel de resolución, por lo que de nuevo se reducirá el grado del árbol para ese nivel. De este modo, se continuará reduciendo el grado hasta que ninguna dimensión requiera mayor nivel de resolución. Por lo tanto, únicamente será necesario especificar en qué nivel del árbol deja de dividirse cada una de las dimensiones. Un ejemplo de este tipo de árbol se muestra en la figura 6.2.

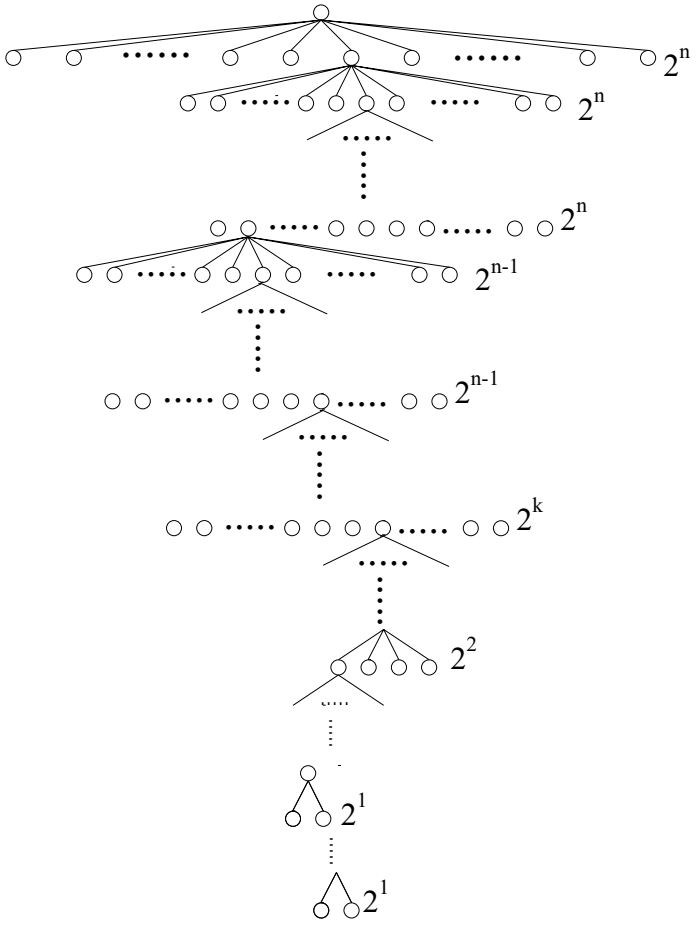


Figura 6.2: Árbol multigrado-2^k

Una diferencia del árbol multigrado-2^k con el *d-slice* radica en la ordenación del árbol. Mientras que en el árbol multigrado-2^k se comienza con el mayor grado posible y se va disminuyendo según se aumenta de nivel en el árbol, en el *d-slice* se comienza con el menor grado y se aumenta según se aumenta de nivel. Por otro lado, el *d-slice* únicamente considera espacios tridimensionales, pues es propuesto como una estructura intermedia para agrupar *quadtrees* y así formar un *octree*.

6.1.2 Árbol multigrado-2^k para representar el C-espacio y el robot

En el caso de estudio que nos ocupa, para representar el robot y el C-espacio tenemos que considerar espacios tridimensionales correspondientes a los tres grados de libertad. En ambos casos el grado máximo del árbol será 2^3.

Dos de las dimensiones corresponden a coordenadas espaciales, mientras que la tercera se corresponde con una coordenada angular por representar una orientación. Por no necesitarse una precisión elevada en la coordenada angular, no se requiere alcanzar un alto nivel de resolución. Sin embargo, las dos coordenadas espaciales serán comparables por su naturaleza, por lo que ambas alcanzarán el mismo nivel de resolución, en general alto pues pueden representar un espacio amplio. Teniendo en cuenta estas características, la coordenada angular se dejará de dividir en niveles anteriores que las espaciales, mientras que las espaciales permanecerán hasta el nivel más profundo del árbol.

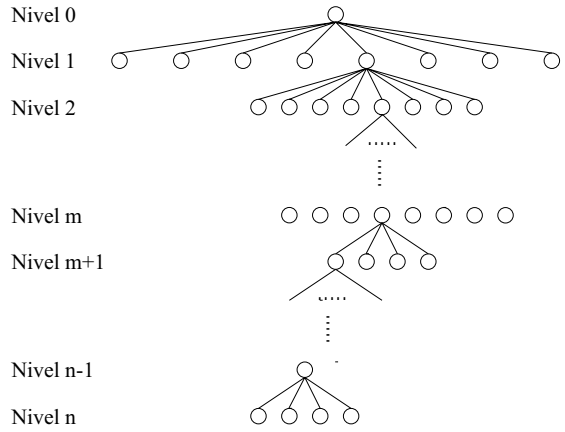
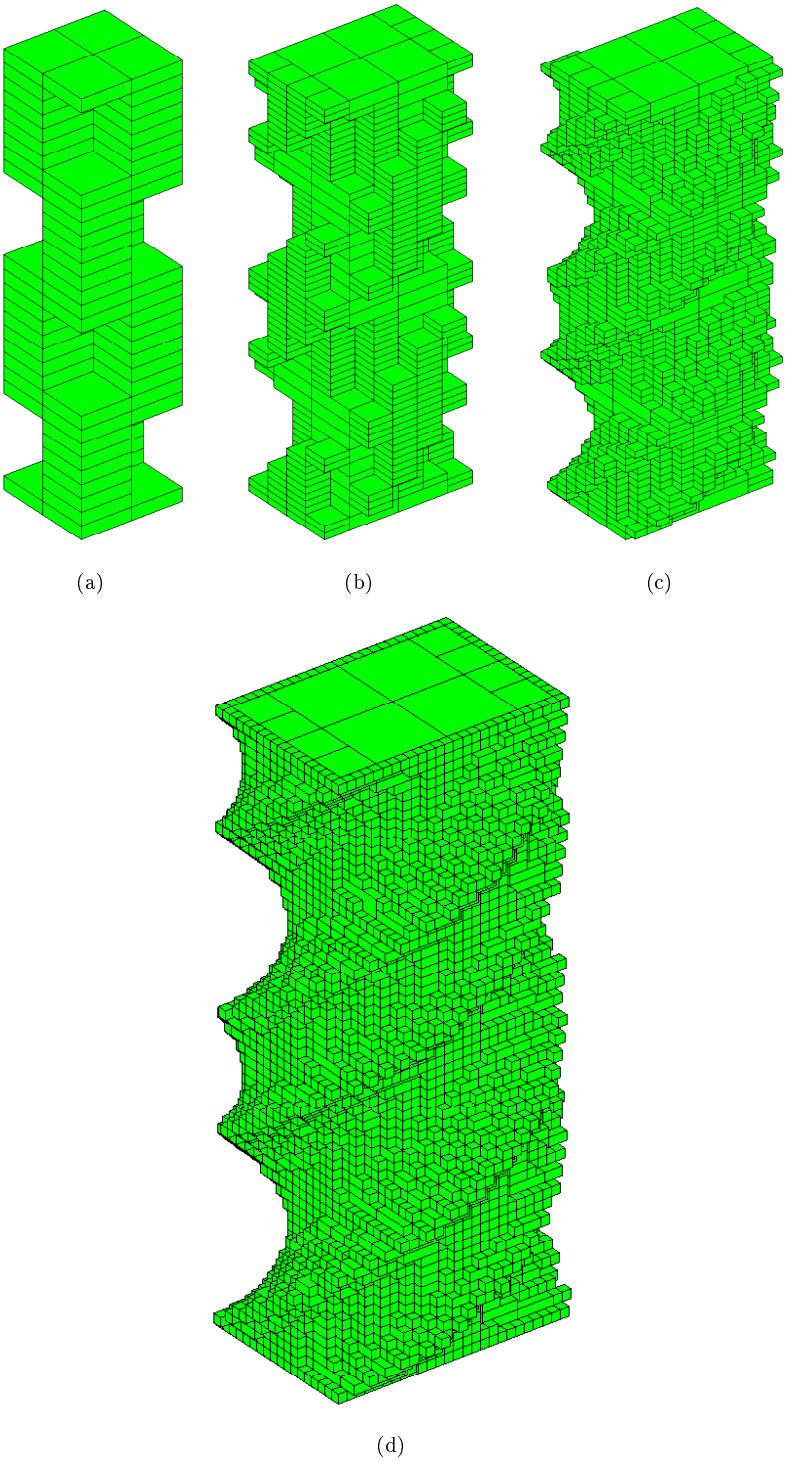


Figura 6.3: Árbol multigrado-2^k para representar el C-espacio

En la figura 6.3 se presenta un esquema de la estructura de árbol multigrado-2^k para representar el C-espacio y el robot. En ella se puede observar cómo los primeros niveles del árbol son de grado 2^3. Al llegar al nivel m la coordenada angular deja de dividirse pasándose a tener niveles de grado 2^2, correspondiéndose con divisiones en las dos coordenadas espaciales. De este modo, al aplicarse



(a) (b) (c)

(d)

Figura 6.4: Representación espacial de una plataforma rectangular con un árbol multigrado-2^k en diferentes niveles de resolución

esta estructura para la representación del C-espacio y el robot, únicamente es necesario especificar el nivel m en el que deja de dividirse la coordenada angular.

En la figura 6.4 se muestra la representación espacial de una estructura de este tipo para el robot. Para una mejor visualización de los diferentes niveles del árbol se ha representado de forma incremental. Esto se ha mostrado así, ya que según se profundiza en el árbol, nodos en niveles inferiores pueden llegar a ocultar nodos de niveles anteriores. En estas representaciones, el eje vertical se corresponde con la coordenada angular, mientras que los ejes del plano horizontal corresponden a los ejes espaciales.

En la figura 6.4(a) aparecen representados los nodos ocupados cuando se trabaja con la misma resolución en las tres direcciones, por lo que todos ellos pertenecen al mismo nivel del árbol. Al añadir a la representación los nodos correspondientes al siguiente nivel (figura 6.4(b)) aparecen los nodos de tamaño menor. Estos nuevos nodos respecto a las coordenadas espaciales tienen doble resolución, al igual que sucede con la coordenada angular. Si en este momento se añaden a la representación los nodos correspondientes al próximo nivel (figura 6.4(c)) se puede comprobar que ahora los nuevos nodos introducidos, aunque en las coordenadas espaciales, tienen tamaño mitad que sus antecesores, en el eje angular las dimensiones son iguales. Por lo tanto, en este nivel ya no se ha seguido dividiendo el eje angular, por lo que el árbol ha pasado de ser de grado 8 a ser de grado 4. Al introducir los nodos del último nivel (figura 6.4(d)) se observa el mismo comportamiento al seguir siendo de grado 4.

En la figura 6.4, una vez que se ha representado con el máximo nivel de resolución, se puede observar que se corresponde con un robot de forma rectangular que presenta todas las orientaciones posibles al rotar alrededor de su centro.

6.2 Aspectos a considerar con la inclusión de la orientación

Como se analizó en el capítulo 5, la aplicación de la convolución jerárquica en la evaluación del C-espacio de un robot móvil conlleva un refinamiento, a medida que se alcanza mayor resolución, únicamente en los lugares que es necesario. Con la introducción de la orientación en el C-espacio también será necesario el

refinamiento de ésta, y además, al ser una variable que no convoluciona, necesita una atención especial.

Por otro lado el concepto en el que está basada la convolución jerárquica es el de la región invariante. Éste, al estar definido para un espacio bidimensional, también debe ser reconsiderado al introducir la tercera dimensión correspondiente a la orientación.

Estos dos aspectos son los que serán expuestos en este apartado, antes de proceder a adaptar la convolución jerárquica para considerar la posibilidad de giros en el robot móvil.

6.2.1 Refinamiento de la orientación: robot virtual

Con el refinamiento de la orientación se van a obtener discretizaciones con diferentes niveles de resolución sobre esta variable angular. En cada una de estas discretizaciones el intervalo que incluye todas las posibles orientaciones $[0, 2\pi]$ será dividido en dos subintervalos de la misma amplitud. Por lo tanto, cada uno de los subintervalos resultantes representa un conjunto de orientaciones del robot.

La representación el robot en cada intervalo de orientaciones se puede realizar mediante el área barrida por el robot al girar en los valores de ángulos definidos por el intervalo. Así, aparece la idea de robot virtual, cuya geometría coincide con el área barrida por el robot real en un intervalo de ángulo. Un ejemplo de esta representación se muestra en la figura 6.5. En ella se representa un robot y el área barrida por éste cuando se consideran diferentes niveles de resolución. Así, para un determinado intervalo, el robot se representa mediante el contorno de la superficie barrida (trazo rojo en la figura 6.5).

Para cada orientación perteneciente a un determinado intervalo, la superficie del robot se encuentra incluida dentro del robot virtual para dicho intervalo. De este modo, si se considera el C-espacio para el robot virtual sin considerar giros, la región libre de C-obstáculos calculada también lo será para el robot en cualquier orientación del intervalo.

De este modo, según se aumenta el nivel de resolución en la variable angular, el intervalo de ángulos representado es menor, y por lo tanto, el área barrida por el robot se asemeja cada vez más a la región ocupada por el robot en

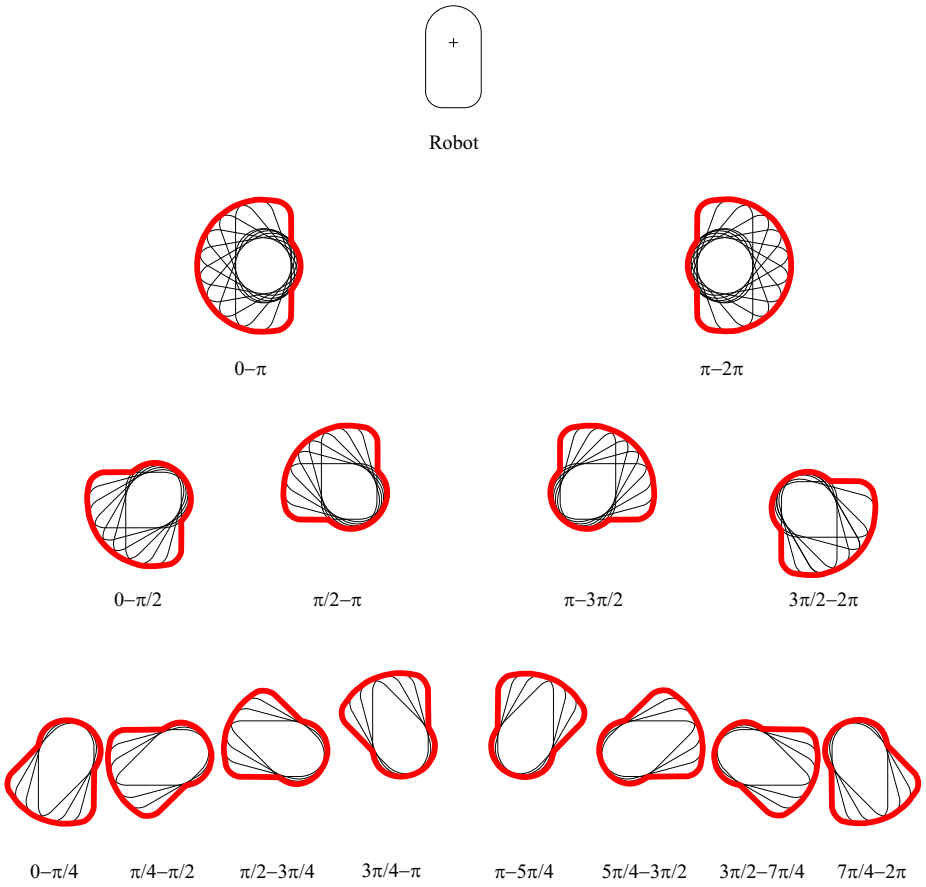


Figura 6.5: Superficie barrida por el robot en un intervalo de orientaciones (robot virtual)

cualquiera de las orientaciones pertenecientes al intervalo. Por esta razón, el C-espacio evaluado se irá asemejando cada vez más al C-espacio para cualquier orientación fija del intervalo.

6.2.2 Región invariante del C-espacio

El concepto principal en el que se sustenta la convolución jerárquica es el de las regiones invariantes. En aquel momento, el concepto de invarianza solamente tenía un significado espacial asociado a las dos variables que convolucionaban. Cuando se introduce la orientación del robot, como una dimensión más en el C-espacio, el concepto de región invariante debe ser extendido.

Una primera generalización podría incluir la tercera dimensión simplemente

pasando de región a volumen invariante. De tal modo que, un volumen en el C-espacio sería invariante si se mantiene libre u ocupado de obstáculos independientemente de cualquier resolución mayor que se utilice para su evaluación.

Esta generalización sería apropiada si las tres dimensiones convolucionaran, puesto que la evaluación del volumen se puede realizar con una única convolución. Sin embargo, según se explicó al principio del capítulo, la orientación es una variable que no convoluciona, y por lo tanto es necesario evaluar una convolución entre el espacio de trabajo y el robot para cada uno de los valores de ángulo. Por tanto, esta extensión del concepto no parece apropiada, puesto que al realizarse convoluciones independientes para los diferentes intervalos angulares del robot no es necesario que se determinen regiones invariantes que incluyan varios intervalos, sino a cada uno de ellos de forma separada.

Para una ampliación más acorde a esta circunstancia, se puede hacer uso del refinamiento de la orientación presentado en el apartado anterior, ya que apareció precisamente por ser una variable que no convoluciona. En concreto, la utilidad provendrá por el uso del área barrida por el robot en un intervalo de ángulos. Se puede considerar un robot virtual que ocupa el área barrida, y evaluar el C-espacio que corresponde a este robot.

A partir de esta región invariante se puede realizar la siguiente definición:

Definición 6.1 (Región invariante angular)
Denominaremos **región invariante angular** a la región invariante del C-espacio evaluado para un robot virtual en un intervalo angular, si la región se mantiene libre u ocupada independientemente de cualquier nivel de resolución mayor que se utilice en las variables espaciales y en la orientación.

Es interesante caracterizar cuáles son estas regiones invariantes angulares, a partir de las caracterizaciones de las regiones invariantes. En el capítulo 5 se llegó a dos caracterizaciones de las regiones invariantes: una se corresponde con el interior de los obstáculos, el cual lleva a una región del C-espacio ocupada; la segunda corresponde con la región libre del C-espacio que está rodeada por las fronteras inferior e izquierda también por C-espacio libre.

Respecto al interior de los obstáculos, como ya se explicó en su momento,

siempre va a formar parte del C-obstáculo por estar la coordenada de referencia del robot dentro de éste. Puesto que es un punto fijo del robot siempre estará dentro de él independientemente de la orientación. Por lo tanto, si una región es invariante para el robot virtual por este motivo, también lo será para cualquier otro intervalo de orientaciones, por lo que se trata de una región invariante angular.

Si se considera la segunda causa de invarianza, el área barrida por el robot al rotar en un intervalo de ángulos incluye al robot en cualquier orientación. Por lo tanto, el C-espacio libre evaluado para el robot virtual también lo es para el robot en cualquier orientación. De este modo la región libre de C-obstáculos invariante para el robot virtual lo será para cualquier orientación, y por lo tanto, también es una región invariante angular.

6.3 Algoritmo modificado

Una vez que se han realizado las consideraciones anteriores, se está en disposición de proceder a explicar el algoritmo propuesto cuando el robot móvil tiene la capacidad de girar.

Es necesario destacar que el procedimiento coincide básicamente con el de la convolución jerárquica, salvo que se añade un refinamiento en la orientación a medida que se obtiene una mayor resolución espacial. Como se trabajará con un menor nivel de resolución en la coordenada angular que en las espaciales, llegará un momento en el que no será posible continuar con el refinamiento angular, por lo que se seguirá únicamente alcanzando mayor resolución espacial.

Se comenzará con el nivel de resolución 1. Así, la orientación se dividirá en dos intervalos, cada uno de ellos representado por el correspondiente robot virtual. De este modo se suponen dos robots virtuales, para los cuales es necesario obtener el C-espacio sin giros en el nivel de resolución 1. Para cada uno de ellos se buscará la región invariante angular (RIA_1^1, RIA_2^1), que lo seguirá siendo para cada subintervalo angular, y por lo tanto no necesita ser evaluada con mayor precisión, ni angular ni espacial. Las complementarias de estas regiones se deben evaluar con mayor precisión ($\overline{RIA_1^1}$, $\overline{RIA_2^1}$). El resultado obtenido

de la convolución se puede trasladar al primer nivel del árbol que representa al C-espacio, dividiéndose el nodo raíz en 8 nodos.

Al proceder con el siguiente nivel de resolución, cada uno de los intervalos angulares debe ser dividido en 2, por lo que ahora se suponen 4 robots virtuales para los que es necesario obtener el C-espacio, en este caso al nivel de resolución 2. Cada uno de los 4 robots virtuales tiene asociada una región no invariante (aquella asociada al intervalo angular al que pertenece), por lo que la convolución no necesita realizarse en todo el espacio, sino solamente en la región no invariante. De forma análoga al paso anterior, se puede obtener para cada uno de los cuatro robots virtuales una región invariante angular ($RIA_1^2,\ RIA_2^2,\ RIA_3^2,\ RIA_4^2$), y sus correspondientes no invariantes, que será donde se evaluará la convolución en el siguiente nivel de resolución. Este resultado también se puede trasladar al árbol obtenido, siendo los niveles de grado 8.

Este procedimiento se repetirá hasta evaluar el nivel de resolución m, a partir del cual suponemos que ya no se requiere mayor precisión en la coordenada angular. En este punto se tendrán C-espacios para 2^m robots virtuales sin capacidad de girar. Así mismo, el árbol hasta este punto es de grado 8, puesto que se han producido divisiones en las tres coordenadas.

Para los iteraciones posteriores no se va a dividir la coordenada angular. Por lo tanto, como únicamente se dividen las coordenadas espaciales, para cada una de las orientaciones que se han alcanzado en el paso anterior se puede proceder como si se tratara del C-espacio de un robot sin giros. De este modo, el árbol pasará a ser de grado 4, como corresponde al *quadtree* utilizado para representar al C-espacio de un robot sin giros.

Una representación en pseudocódigo se muestra en los Algoritmos 6.1 y 6.2. El Algoritmo 6.1 se corresponde con la realización de divisiones en la variable angular, según se aumenta la resolución. Y el Algoritmo 6.2 comienza cuando ya no es necesario realizar más divisiones en la variable angular, puesto que ya se ha alcanzado la mayor resolución angular requerida, pero es necesario trabajar con mayor resolución espacial.

De las líneas 1 a la 5 se realiza la inicialización de las variables necesarias. Cabe destacar el uso de la lista $\text{RDIs}_{(i,j)}$ que representa a todo el conjunto de

regiones de interés que será necesario considerar a la resolución i. Puesto que la evaluación se hace por capas para intervalos de ángulos, se determina a qué capa pertenece mediante el índice j.

Algoritmo 6.1

```
1: quadtree del espacio = obtener espacio()
2: RDI = todo el espacio
3: RDIs(1,1) ← RDI
4: árbol multigrado del robot = obtener robot()
5: árbol multigrado del c-espacio = nuevo()
6: para i = 1 hasta i = m
7:    para j = 1 hasta j = 2^i
8:       para todo RDI de RDIs[i][j/2]
9:          intervalo angular = [(j − 1)2π/2^i, j 2π/2^i]
10:         matriz del espacio = matriz del quadtree del espacio de RDI
11:                              a resolución i
12:         matriz del robot = matriz del árbol multigrado-2^k del robot
13:                            a resolución i en el intervalo angular
14:         resultado = convolucionar(matriz del espacio, matriz del robot)
15:         invariante = obtener invariante(resultado)
16:         no invariante = resultado\invariante /*complemento relativo a*/
17:         RDI = determinar región de interés(no invariante)
18:         si es necesario segmentar (RDI) entonces
19:            RDIs[i+1][j] ← segmentar(RDI)
20:         si no
21:            RDIs[i+1][j] ← RDI
22:         pasar información de invariante al quadtree del C-espacio
```

En la línea 6 comienza el primer bloque de iteraciones en el que se va aumentando la resolución. La resolución máxima que se considera es m, que es donde deja de dividirse la variable angular. Como la evaluación se realiza por capas de resolución angular es necesario evaluar la convolución para cada capa en la línea 7. Al aumentar la resolución tanto en las variables espaciales como en la

angular, el número de intervalos angulares depende de la resolución a la que se trabaje.

Después es necesario realizar la convolución para todas las regiones de interés que se determinaron en el paso anterior (línea 8). En este punto es necesario comentar que, como el paso de una resolución angular a otra está basado en las regiones invariantes angulares, las RDIs a utilizar son aquellas que corresponden a los intervalos angulares previamente evaluados, que son la mitad en número.

Algoritmo 6.2

```
23: para i = m + 1 hasta i = resolución total
24:   para j = 1 hasta j = 2^m
25:     para todo RDI de RDIs[i][j]
26:       intervalo angular = [(j − 1)2π/2^m, j 2π/2^m]
27:       matriz del espacio = matriz del quadtree del espacio de RDI
28:                            a resolución i
29:       matriz del robot = matriz del árbol multigrado-2^k del robot
30:                          a resolución i en el intervalo angular
31:       resultado = convolucionar(matriz del espacio, matriz del robot)
32:       si i = resolución total entonces
33:         invariante = resultado
34:       si no
35:         invariante = obtener invariante(resultado)
36:         no invariante = resultado\invariante /*complemento relativo a*/
37:         RDI = determinar región de interés(no invariante)
38:         si es necesario segmentar (RDI) entonces
39:           RDIs[i+1][j] ← segmentar(RDI)
40:         si no
41:           RDIs[i+1][j] ← RDI
42:       pasar información de invariante al quadtree del C-espacio
```

El resto del pseudocódigo es similar al expuesto para el robot sin giros. La única consideración adicional a tener en cuenta es que ahora la matriz del robot se corresponde con la del robot virtual en el intervalo angular.

El siguiente bloque de pseudocódigo (Algoritmo 6.2) comienza en el primer

nivel de resolución $(m+1)$ en el que ya no se divide la coordenada angular. Termina cuando se alcanza el máximo nivel de resolución espacial requerido. Además, como se ha alcanzado la máxima resolución angular, los intervalos angulares que es necesario evaluar siempre son la misma cantidad 2^m. El resto es similar al último bloque del Algoritmo 6.1.

6.4 Caso de estudio

Para comprobar el correcto funcionamiento del algoritmo se va obtener el C-espacio para un plataforma con forma rectangular moviéndose libremente por un espacio de trabajo R^2 con tres obstáculos. Para el espacio de trabajo se ha considerado una discretización con una resolución de 1024×1024 y para el robot se ha tomado una resolución angular de 64. De este modo, el C-espacio tendrá una resolución de $1024 \times 1024 \times 64$.

En la esquina superior derecha de la figura 6.4 se muestra el espacio de trabajo, con los obstáculos en color negro y el robot en gris. En la misma figura se muestra el C-espacio evaluado para ambos en una estructura de árbol multigrado-2^k. Para una mejor visualización del resultado se muestra en la figura 6.4 únicamente los dos C-obstáculos que están próximos, donde se puede comprobar como llegan a unirse.

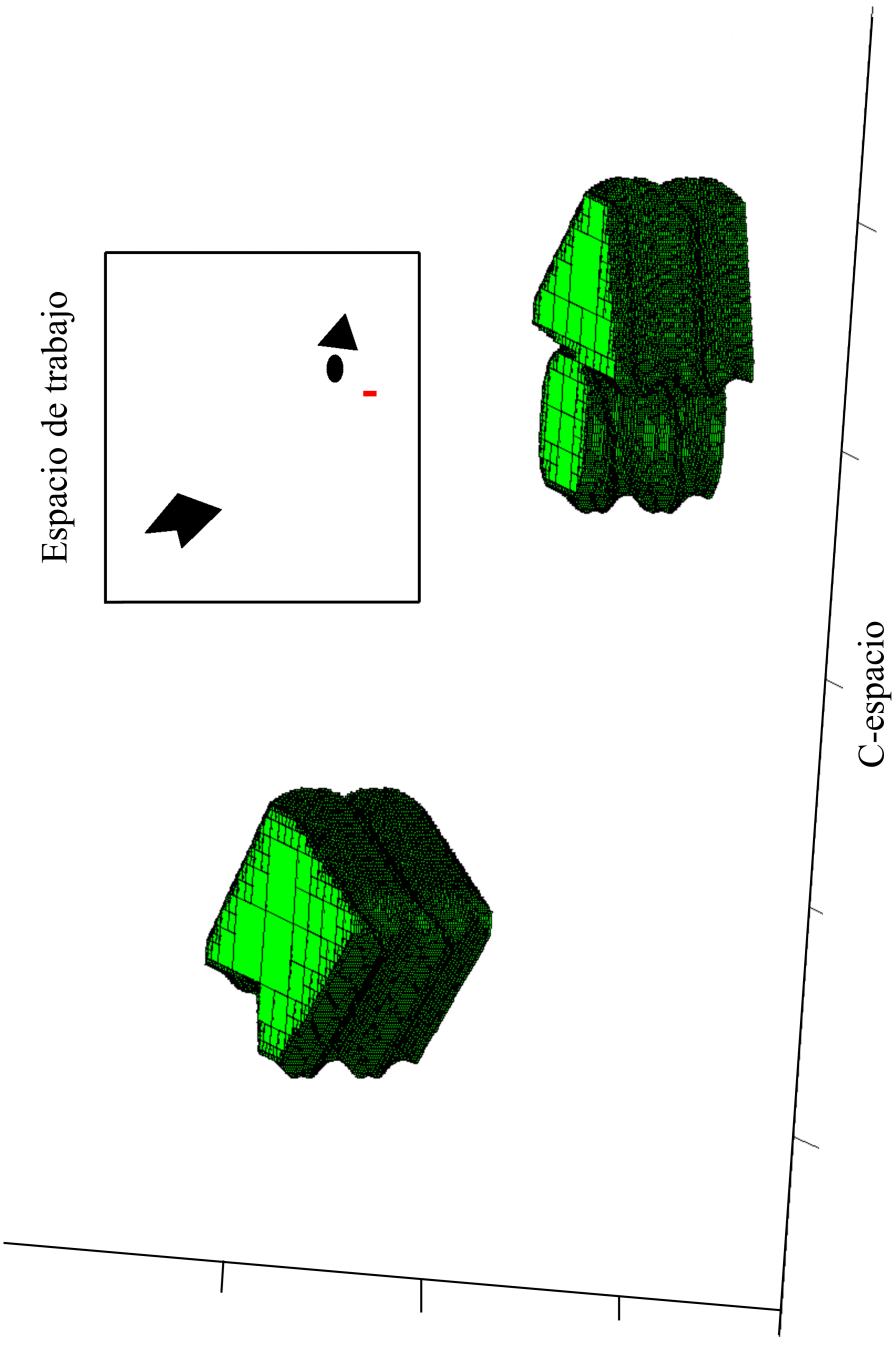


Figura 6.6: Espacio de trabajo con el robot y el C-espacio evaluado

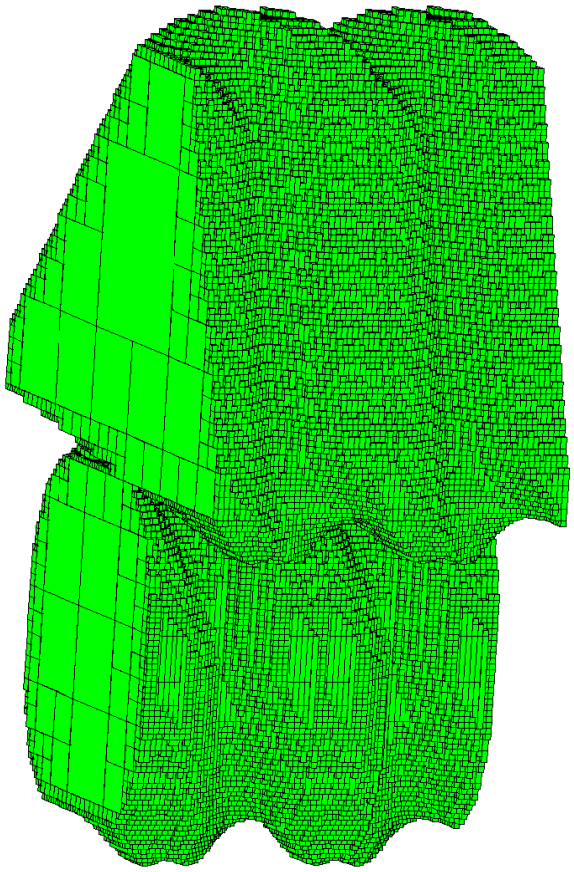


Figura 6.7: Detalle de dos C-obstáculos

7

Técnicas y Herramientas

Este capítulo se dedica a una breve exposición de las principales técnicas y herramientas que se han utilizado en la realización de esta tesis doctoral. Como técnica destacar las redes neuronales artificiales, que son métodos computacionales empleados para almacenar información de patrones y generalizar para patrones desconocidos. En cuanto a las herramientas, distinguir el SNNS como una herramienta para implementar redes neuronales artificiales y la biblioteca FFTW para realizar transformadas de Fourier discretas.

7.1 Técnicas

En esta sección se expone la técnica que se ha considerado apropiada para la determinación de los parámetros óptimos a utilizar en el algoritmo para realizar la segmentación: las redes neuronales artificiales. A este respecto se muestra el

concepto de neurona artificial y red neuronal en general, particularizando para el tipo de red neuronal utilizada, el perceptrón multicapa.

7.1.1 Redes neuronales artificiales

Las redes neuronales artificiales (RNA) son un grupo de métodos computacionales, cuya denominación procede de la similitud que muestran con el modelo de procesamiento neuronal biológico. El cerebro humano almacena la información en forma de patrones con distintos niveles de complejidad. Este proceso de almacenar información en patrones, utilizarlos y resolver problemas con ellos es el que tratan de imitar las redes neuronales. Por esta razón, en primer lugar, se van a examinar conceptos elementales acerca de la neurona biológica, que son los que sustentan, en parte, la definición de las Redes Neuronales Artificiales (RNA).

7.1.1.1 La neurona biológica

La unidad funcional del cerebro, la neurona, está constituida por un cuerpo celular o soma (entre 10 y 80 μm) del que surge un denso árbol de ramificaciones (dendritas) y una fibra tubular o axón (entre 100 μm y 1 metro). Esta neurona es un elemento de proceso muy simple con un canal de entrada formado por las dendritas, un procesador que sería el cuerpo celular y un canal de salida constituido por el axón (Figura 7.1).

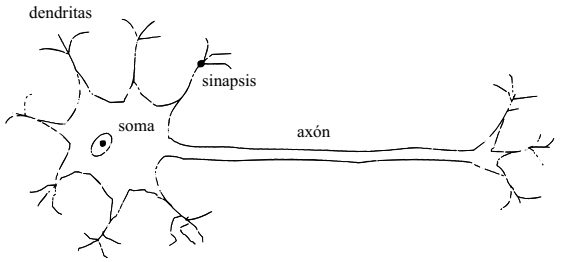


Figura 7.1: Esquema básico de la neurona biológica

Una neurona de la corteza cerebral recibe unas 10.000 entradas, y envía a su vez salida a varios cientos de neuronas. La transmisión de la información de una neurona a otra se lleva a cabo mediante una complicada reacción electroquímica

a través de las unión entre el axón y las dendritas, o sinapsis. Las sinapsis liberan sustancias químicas transmisoras que elevan o reducen el potencial eléctrico de la célula. Una vez que el potencial eléctrico rebasa cierto límite, se envía al axón un impulso eléctrico o potencial de acción que, finalmente, llega a las sinapsis y libera transmisores en los cuerpos de otras células. Las sinapsis que aumenten el potencial serán excitadoras y las que lo disminuyen inhibitorias. Dichas conexiones sinápticas muestran además plasticidad, es decir, puede alterarse la intensidad de las conexiones en respuesta al patrón de estimulación. Las neuronas establecen también nuevas conexiones con otras neuronas. Las neuronas en la corteza cerebral, además, se organizan en capas y en grupos especializados. Se puede decir que, el procesamiento de la información es muy complejo, no lineal y con un elevado grado de paralelismo.

7.1.1.2 Redes neuronales artificiales

Las Redes Neuronales Artificiales (RNA) intentan imitar el funcionamiento del cerebro humano donde el elemento básico de una RNA es la neurona artificial o elemento de proceso (EP).

Así una Red Neuronal Artificial se puede definir como una estructura de proceso de información, paralela y distribuida, formada por Elementos de Proceso (EP), interconectados a través de canales unidireccionales llamados conexiones, con una salida única que se distribuye sobre un número de conexiones (Figura 7.2). La salida puede ser de cualquier tipo numérico (binario, bipolar, entero, real, etc.). Pero a diferencia del cerebro donde las neuronas pueden interconectarse de manera incontrolada, las redes se disponen en un número limitado de capas donde existen interconexiones entre las neuronas de cada nivel.

Dentro de una capa, las neuronas suelen ser del mismo tipo y la mayoría de las redes neuronales van a estar formadas por tres niveles o capas:

- **Capa de entrada** recibe datos o señales procedentes del exterior y las propaga hacia la capa intermedia.
- **Capa intermedia o de neuronas ocultas** que no recibe, ni suministra información al entorno. Las neuronas o nodos de esta capa oculta contienen valores intermedios que son calculados por la red.

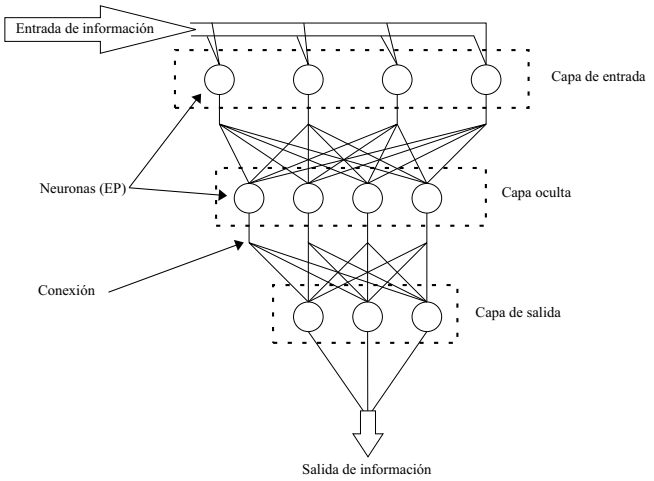


Figura 7.2: Arquitectura de Red Neuronal Artificial

- **Capa de salida** cuyas neuronas proporcionan la respuesta de la red.

Se puede plantear como conclusión que los tres conceptos clave que las RNA intentan imitar del cerebro humano son:

- El procesamiento en paralelo de la información. Las diferentes neuronas están operando de forma paralela, permitiendo tratar grandes volúmenes de información en poco tiempo.
- La memoria distribuida. La información almacenada se encuentra en las sinapsis.
- La capacidad de adaptación al entorno. Las sinapsis van variando con el tiempo, modificando su valor a partir de ejemplos.

El último aspecto señalado es clave, ya que es el que incorpora la capacidad de aprendizaje y, en definitiva, la posibilidad de obtener información a partir de los ejemplos que se le van a presentar. Para esto, es necesario, primero, plantear el modelo de RNA que se va a utilizar.

7.1.1.3 Modelo de neurona artificial

Las redes neuronales artificiales se han desarrollado bajo diferentes enfoques en los últimos 50 años. Sin embargo, la crítica sobre la experimentación sin

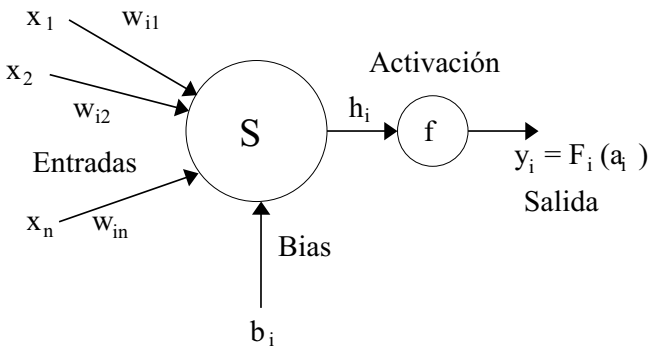


Figura 7.3: Modelo básico de funcionamiento de una nerurona artificial

directrices y la ausencia de rigor matemático que caracterizó una buena parte de los primeros trabajos, hizo que se perdiera el interés sobre estos métodos durante mucho tiempo. En 1986 [RHW86] se dan a conocer de nuevo las RNA, aumentando el interés por éstas dentro de la comunidad científica.

El modelo de Rumelhart y McClelland (1986) define un elemento de proceso, o neurona artificial, como un dispositivo que a partir de un conjunto de entradas, $x_i, i = 1, ..n$ (o vector x), genera una única salida y. En la Figura 7.3 se muestra el modelo básico funcionamiento de neurona en el que se distinguen:

- Un conjunto de entradas o vector de entrada x, de n componentes.
- Un conjunto de conexiones o sinapsis, caracterizadas cada una de ellas por un peso sináptico w_{ij}, que representa la intensidad de la interacción entre la neurona presináptica j y la postsináptica i. Este peso puede variar en un intervalo que incluye tanto a valores negativos (entrada inhibitoria) como valores positivos (entrada excitatoria). Asimismo incluye un valor de *bias* b_i que, de forma independiente a las entradas, determina el comportamiento de la neurona.
- Una regla de propagación $s(w_{ij}, x_j(t), b_i)$, que proporciona el potencial postsináptico, $h_i(t)$.
- Función de activación, $a_i(t) = f_i(a_i(t-1), h(t))$, que determina el estado de activación de la neurona en función del estado anterior $a(t-1)$, y del valor postsináptico $h_i(t)$. El objetivo de esta función es limitar la amplitud de la señal de salida de la neurona dentro de un intervalo de

valores normalizado. Habitualmente se utiliza una función sigmoidea con un intervalo de valores entre 0 y 1.

- Una función, $F_i(t)$ que proporciona la salida $y_i(t) = F_i(a_i(t))$

Por tanto, matemáticamente, el modelo básico de neurona se podría resumir:

$$y_i(t) = F_i(a_i(t)) = F_i(f_i(a_i(t-l), h_i(t))) = F_i(f_i(a_i(t-l), s(w_{ij}, x_j(t))))$$

7.1.1.4 Aprendizaje de la red neuronal artificial

La principal propiedad de una red neuronal artificial es la capacidad de aprender del entorno en el que opera y de realizar cambios a través de ese aprendizaje. Existen diferentes algoritmos para desarrollar la labor de aprendizaje. En general, los diferentes algoritmos se pueden englobar en dos grupos que se corresponden con dos tipos de aprendizaje ([RN95]):

- **No supervisado**. El propósito de un algoritmo de aprendizaje no supervisado es descubrir modelos o características a través de los datos de entrada.

- **Supervisado**. En el aprendizaje supervisado se introduce el comportamiento deseado en el proceso de enseñanza (por ejemplo el valor de la función a identificar). Se necesitan, pues, tanto las entradas como las salidas. La red procesa las entradas y compara los resultados obtenidos por las salidas con los valores esperados. La diferencia entre ambos valores se propaga hacia las capas intermedias haciendo que el sistema ajuste los pesos que definen el comportamiento de la red. Este tipo de aprendizaje supervisado aparece como el más adecuado a la hora de automatizar el procedimiento de aproximación de funciones. Dentro de este tipo, uno de los modelos de red más utilizado y que ha presentado mejores resultados es el denominado Perceptrón Multicapa (MLP, *Multi-Layer Perceptron*).

7.1.1.5 Perceptrón multicapa

El tipo de red neuronal Perceptrón Multicapa fue desarrollado por Rumelhart, Hinton y Willians ([RHW86]) en 1986. Es la extensión y generalización de otro tipo de red, el perceptrón simple, que estaba constituido sólo por dos

grupos de neuronas: unas de entrada y otras de salida, conectadas entre sí. Las redes neuronales perceptrón multicapa constan habitualmente de las tres capas descritas anteriormente: una primera capa de entrada, una capa intermedia u oculta y una capa de salida.

La información se propaga de capa en capa (de entrada a salida) por medio de las conexiones existentes entre las neuronas de cada una de ellas. De esta forma cada neurona de la capa de entrada se encuentra conectada con cada una de las neuronas de la capa intermedia, y cada una de estas últimas lo estará a su vez con cada uno de los nodos de la capa de salida. Cada par de neuronas o nodos es conectado por un peso o fuerza matemática que indica el grado y tipo de interacción entre los nodos, de forma que este peso o fuerza puede ser positivo o negativo. Por ejemplo, un peso +0,9 entre un nodo de entrada y uno oculto indicaría que el nodo de la capa oculta ha sido fuertemente estimulado por el nodo de entrada. Un peso de +0,1 indicaría una conexión mucho más débil. La salida de un determinado nodo representa la suma de los valores individuales de entrada multiplicados por los respectivos pesos de conexión entre estos nodos de entrada y un determinado nodo de salida. Los nodos de las capas intermedias calculan la suma de los productos entre los valores de las neuronas de entrada y los valores de los pesos asociados a las conexiones entre ambas capas de neuronas. Esta suma es utilizada por cada una de las neuronas de la capa intermedia para calcular el valor de una función sigmoidea (o función de similitud).

Método de aprendizaje de MLP

El primer paso para la utilización de una RNA se corresponde con una etapa de entrenamiento o aprendizaje. Para ello es necesario disponer de un conjunto de datos cuyo comportamiento es conocido y que va a ser utilizado en esta fase. Así, los datos utilizados para el aprendizaje se denominan "conjunto de entrenamiento". El entrenamiento es un proceso reiterativo en el que la red aprende a través de los ejemplos mostrados. Durante el aprendizaje se procesan varías veces los mismos pares de vectores de entrenamiento y en cada paso del aprendizaje se modifican los pesos de las conexiones. El tiempo de aprendizaje es muy variable y puede durar segundos o semanas.

Al principio del entrenamiento, los pesos de conexión asignados son valores

aleatorios. Los datos de entrada se presentan a la capa de entrada de la red. Esta información se utiliza para realizar varias operaciones a lo largo de las distintas capas de la red, hasta que se llega a la capa de salida, en la cual se genera un resultado. En una segunda etapa, el resultado que proporciona la red es comparado con el resultado esperado para cada uno de los patrones de entrenamiento. Si el valor estimado por la red en el entrenamiento se encuentra dentro del error de tolerancia, esta estimación se considera correcta y se presentarán las siguientes entradas. El error de tolerancia se refiere al error aceptado en la estimación para que la clasificación sea considerada correcta. Si la estimación no se encuentra dentro del error de tolerancia, la matriz de conexión de la red se cambia, para reducir el error. Se calcula una señal de error que se propaga hacia atrás a través de la red neuronal. Esta señal es un factor matemático que ajusta el valor de cada peso en la red neuronal para reducir la diferencia entre la salida estimada y la real. El hecho de que el error se propague desde la capa de salida hacia la entrada es lo que hace que este tipo de redes se denomine de retropropagación o de propagación hacia atrás (backpropagation).

El procedimiento de entrenamiento continúa hasta que la red clasifica correctamente a todos los patrones de entrenamiento. Si no se puede conseguir una clasificación correcta al 100%, se continúa el entrenamiento hasta un punto final que se ha definido de antemano.

El procedimiento básico para entrenar la red se resume en los siguientes pasos:

- Introducir los vectores de entrada en la red y calcular la salida.
- Comparar los valores proporcionados por la red con los vectores de salida correspondientes a las entradas, y así calcular el error cometido por la red.
- Determinar en qué dirección (+ ó -) hay que cambiar los pesos con el objetivo de reducir los errores.
- Determinar la cantidad en la que hay que modificar los pesos.
- Modificar los pesos.
- Repetir los pasos anteriores con todos los patrones de entrenamiento hasta

que el error medio para el conjunto de entrenamiento se reduzca hasta un nivel adecuado.

Las ecuaciones relevantes para el entrenamiento de una RNA perceptrón con una capa intermedia, que utiliza un aprendizaje hacia atrás, se pueden resumir de la siguiente forma:

- Presentar el vector de entrada, $x^p = (x_1^p, x_2^p, ..., x_N^p)^T$, en la capa de entrada.
- Calcular el valor de los niveles de excitación de las neuronas de la capa intermedia, de acuerdo con la expresión $s_i^p = \sum_{j=1}^{N} w_{ji} x_j^p + b_i$
- Calcular las salidas de las neuronas de la capa intermedia $y_i^p = F_i(s_i^p)$
- Calcular el valor de los niveles de excitación de las neuronas de la capa de salida, de acuerdo con la expresión $s_k^p = \sum_{i=1}^{L} w_{ik} y_i^p + b_k$
- Calcular las salidas de la red $y_k^p = F_k(s_k^p)$
- Calcular la sensibilidad de las neuronas de la capa de salida a partir del error apreciado en la salida con el vector de salida deseado $d^p = (d_1^p, d_2^p, \ldots, d_M^p)^T$

$$\delta_k^p = (d_k^p - y_k^p) F_k'(s_k^p)$$

- Calcular la sensibilidad de las neuronas de la capa intermedia de acuerdo con la expresión $\delta_i^p = F_i'(s_i^p) \sum_{k=1}^{M} w_{ik} \delta_k^p$
- Actualizar los pesos y los términos "*bias*" de las conexiones que unen las neuronas de la capa intermedia con las de la capa de salida. Los parámetros η y μ son los que van a determinar la velocidad y calidad del proceso de aprendizaje.

$$w_{ik}(t+1) = w_{ik}(t) + \eta \delta_k^p y_i^p + \mu(w_{ik}(t) - w_{ik}(t-1))$$

$$b_k(t+1) = b_k(t) + \eta \delta_k^p y_i^p + \mu(b_k(t) - b_k(t-1))$$

- Actualizar los pesos y los términos "*bias*" de las conexiones que unen las neuronas de la capa de entrada con las de la capa intermedia:

$$w_{ji}(t+1) = w_{ji}(t) + \eta\delta_i^p x_i^p + \mu(w_{ji}(t) - w_{ji}(t-1))$$

$$b_i(t+1) = b_i(t) + \eta\delta_i^p x_j^p + \mu(b_{ji}(t) - b_{ji}(t-1))$$

- Calcular el término de error:

$$E^p = \frac{1}{2}\sum (d_k^p - y_k^p)^2$$

Dado que este término refleja la capacidad de adaptación de la red, hay que tenerlo en cuenta para determinar si la red aprende de forma satisfactoria o no.

Validación de la RNA

El paso final en el diseño de la red de retropropagación es la validación. Se trata de evaluar la capacidad de la red entrenada para generalizar. En esta fase, se introduce otro conjunto de prueba o test, formado por patrones no utilizados previamente durante el entrenamiento de la red. El valor estimado por la red se comparará con el real de este conjunto. A diferencia del entrenamiento, la validación es rápida, ya que implica un sólo paso a través de la red neuronal, y la matriz de conexión no es manipulada. Esta validación se considera necesaria ya que uno de los principales problemas del entrenamiento de la RNA es el sobre-entrenamiento. Así, si la red se entrena "demasiado" pierde capacidad de generalización. Para evitarlo, normalmente lo que se hace es controlar el aprendizaje, evaluando periódicamente la red durante el entrenamiento, con un conjunto de datos independiente (un subconjunto que sea representativo y que no se haya utilizado en dicho entrenamiento).

En la Figura 7.4 se muestra la relación entre el error y la duración del entrenamiento. Durante el entrenamiento, el error de la red disminuye gradualmente hasta un mínimo. En el conjunto de prueba el error puede disminuir al principio y comenzar a aumentar si se ha producido un sobre-entrenamiento.

Otra causa de un falso aumento de clasificación durante el entrenamiento y, que puede observarse durante la fase de validación, es que el conjunto de entrenamiento tenga un tamaño pequeño. Para cualquier método multivariante, existe un aumento en la probabilidad de observar cambios en la relación entre los datos entrada y los de salida, a medida que disminuye la relación entre el

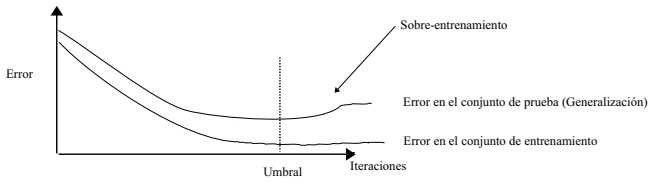


Figura 7.4: Problemas en el entrenamiento de la RNA

tamaño del conjunto de patrones y el número de parámetros (o pesos). Por otro lado, la diferencia entre el rango de clasificación durante el entrenamiento y el rango en la verdadera validación también parece depender del número de neuronas ocultas. Así, si el número de neuronas en la capa oculta es excesivo se producirá un "sobre-ajuste" que implica también pérdida de capacidad de generalización. Normalmente, se realizarán entrenamientos con distinto número de neuronas en la capa oculta para encontrar el número óptimo.

7.2 Herramientas

Esta sección está dedicada a la exposición de las principales herramientas software que han sido necesarias para la consecución del objetivo final del trabajo de investigación objeto de esta memoria. En primer lugar se presenta SNNS, una herramienta muy difundida para el desarrollo de gran cantidad de redes neuronales. En cuanto a la segunda herramientas, FFTW, se trata de una biblioteca de funciones que permite la evaluación de la Transformada Rápida de Fourier en varias dimensiones.

7.2.1 SNNS

A la hora de desarrollar una red neuronal, se puede plantear la codificación de los diferentes procedimientos de aprendizaje mediante un lenguaje de programación. Sin embargo, esta opción apenas es utilizada, dado que existe una gran cantidad de aplicaciones para el diseño, aprendizaje o entrenamiento mediante redes neuronales utilizando diferentes paradigmas.

De entre todas las disponibles, se ha elegido el SNNS (*Stuttgart Neural Net-*

work Simulator), que ha sido desarrollado en la Universidad de Stuttgart, y distribuida bajo licencia GNU (*GNU's Not Unix*).

Esta herramienta tiene una gran versatilidad, pues permite el diseño de gran cantidad de estructuras de redes neuronales, así como de un conjunto amplio para las funciones de activación de las neuronas artificiales. Para ello dispone de un interfaz gráfico donde se pueden incluir las diferentes neuronas que forman la red, así como las conexiones entre ellas.

Además, dispone de multitud de paradigmas de entrenamiento de la red, pudiendo experimentar con ellos para encontrar el que mejores resultados proporcione para el problema planteado. Para la etapa de entrenamiento dispone de una ventana gráfica en la que se muestra la evolución del error tanto para los datos de entrenamiento como para los de prueba, de tal modo que se puede seguir la evolución del entrenamiento de la red y evitar el sobre-entrenamiento.

Otro aspecto de gran importancia es que, a partir de la herramienta *snns2c*, permite la generación de código fuente en lenguaje C. Así, se consigue la explotación de una red neuronal ya entrenada como un módulo en cualquier aplicación. Esto será de gran utilidad en nuestro caso para la obtención de los parámetros de sintonización *on-line*.

Todas estas razones son las que han llevado a la elección de SNNS como la herramienta para la realización de la red neuronal, de entre todas las de libre distribución.

7.2.2 FFTW

Como es bien conocido, la Transformada Discreta de Fourier (*Discrete Fourier Transform*, DFT) se utiliza ampliamente para realizar análisis frecuencial de señales no periódicas. El algoritmo para el cálculo de la DFT es bastante complicado, ya que implica la realización de múltiples sumas y multiplicaciones con números complejos[1]. La Transformada Rápida de Fourier (*Fast Fourier Transform*, FFT) es una herramienta que permite alcanzar el mismo objetivo, pero con menos sobrecarga en los cálculos.

[1] Por ejemplo, la realización de la DFT para una señal de 8 muestras requeriría 49 productos complejos y 56 sumas complejas. Aunque este es un caso manejable, sin embargo, para un caso realista con señales de 1024 muestras, se requieren 2×10^7 sumas y productos complejos

La idea subyacente en la FFT es el conocido enfoque *divide y vencerás*, ya que divide la muestra original de N puntos en secuencias de tamaño más pequeño de forma recursiva. Así, la DFT requiere $(N-1)^2$ productos complejos y $N(N-1)$ sumas complejas frente al enfoque de la FFT, que para un tamaño que sea una potencia de 2, divide la señal en una serie de muestras de 2 puntos que simplemente requieren un producto y una suma y la recombinación de los puntos. Cuando el número de puntos de la muestra original no es una potencia de 2, la división de la secuencia se realiza para la obtención de secuencias más pequeñas que puedan ser evaluadas con menor cantidad de operaciones complejas.

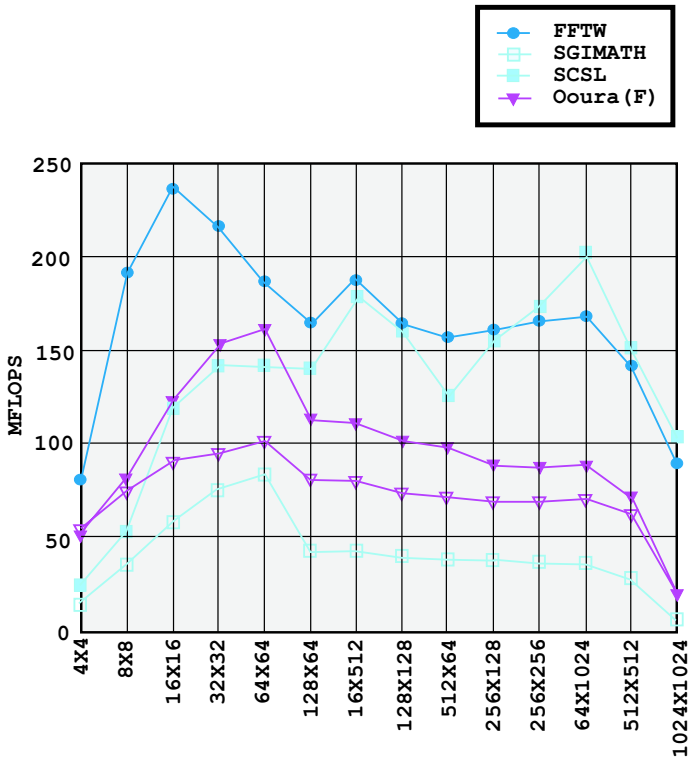


Figura 7.5: Transformadas de Fourier reales en 2D, Mflops para diferentes tamaños (potencia de dos) de muestra

Existen múltiples versiones del algoritmo para el cálculo de la FFT, tanto de libre distribución como propiedad de algún vendedor, y muchas de ellas utilizan el cálculo paralelo para acelerar los cálculos. Entre todas estas posibilidades se han hecho pruebas con varias de las bibliotecas disponibles. Entre ellas, destaca

la biblioteca FFTW, desarrollada por el MIT, y que como se muestra en la figura 7.5 presenta un rendimiento óptimo (usando Mflops[2] como medida de la velocidad); y la biblioteca *complib.sgimath* de Silicon Graphics, que obtiene resultados generalmente un poco inferiores, aunque en algunos casos mejores que FFTW.

La biblioteca de funciones que proporciona FFTW presenta algoritmos optimizados cuando se trata de secuencias cuyo tamaño no es una potencia de 2. Para obtener este rendimiento ajusta la descomposición de las muestras mediante programación dinámica (*plan*[3]) teniendo en cuenta el procesador, la memoria y el compilador ([FJ98]). Para la evaluación de cada uno de estos fragmentos se genera de forma dinámica código especializado (*codelet*) para ese tamaño. Además, tiene la funcionalidad de aprender de las planificaciones que ha realizado anteriormente para generar planificaciones de tamaños diferentes.

Por estas características, además de ser un software de libre distribución, se optó por la biblioteca FFTW, puesto que las transformadas que son necesarias realizar serían de tamaños muy diversos.

[2]Millones de operaciones de coma flotante por segundo.

[3]Las divisiones óptimas de las señales de un tamaño determinado para evaluar la FFT

8

Procedimiento de Evaluación del C-espacio

En este capítulo se va a presentar cómo deben ser utilizados los algoritmos anteriormente propuestos en busca de un óptimo rendimiento. Tal y como se comentó en el apartado 5.7, en los algoritmos propuestos para el cálculo del C-espacio, el proceso de segmentación tiene una gran dependencia con el espacio de trabajo. Además, la segmentación se puede ajustar mediante dos parámetros que rigen su comportamiento: el porcentaje máximo de región innecesaria permitida sin segmentar y el número de semillas que se utilizan para encontrar el *MRI*.

De este modo, este capítulo se va a estructurar de la siguiente forma. En un primer apartado, se va a estudiar el comportamiento del algoritmo para diferen-

tes espacios de trabajo con el objeto de analizar cómo afectan los parámetros de la segmentación en el rendimiento del algoritmo.

Después, se presentará el procedimiento para la aplicación del algoritmo, haciendo especial énfasis en el uso de una red de neuronas artificiales como herramienta que determine la dependencia funcional entre el espacio de trabajo y los parámetros que deben ser utilizados.

Como se ha señalado, en la convolución jerárquica una tarea fundamental es la realización de segmentaciones en el espacio de trabajo que determinen las regiones de interés. Se estudiará con detalle su realización para el caso de una plataforma móvil sin giros, dado que su extensión a una plataforma con giros se realiza de forma inmediata.

Por último, se compararán los resultados obtenidos con este nuevo método propuesto con el ya clásico de realizar la convolución en forma de matrices. Así, se realizará una comparativa entre ambos haciendo un gran hincapié en los requerimientos de memoria, lo cual era el principal objetivo de este trabajo.

8.1 Análisis del rendimiento del algoritmo

Con objeto de caracterizar el comportamiento del algoritmo frente a los dos parámetros utilizados en la segmentación (el porcentaje de región innecesaria en el *mrc* de la RDI y el número de semillas utilizadas para encontrar el *MRI*) se ha aplicado a diferentes espacios de trabajo. Estos espacios de trabajo tienen unas dimensiones de $6,4\ m \times 6,4\ m$ tomados con una resolución de $6,25\ mm$ en cada lado. Así, se pueden representar mediante matrices de 1.024×1.024 pixels. Además, se ha supuesto un robot de $37,5\ cm \times 21,25\ cm$, lo que supone aproximadamente el 2‰ del espacio de trabajo.

Para abarcar un amplio intervalo de posibilidades se han considerado espacios de trabajo con características bien diferenciadas. Así, se han escogido los siguientes espacios de trabajo con:

- 2 obstáculos que ocupan el 0,1 %
- 2 obstáculos que ocupan el 10 %
- 20 obstáculos que ocupan el 0,1 %

- 20 obstáculos que ocupan el 10 %

Es necesario resaltar, que aunque se considerará únicamente un espacio de trabajo para cada situación, el experimento se ha realizado para varios espacios de cada tipo, observando una tendencia similar.

A continuación se analizará el tiempo de cálculo y el consumo de memoria empleados por el algoritmo para cada uno de los cuatro escenarios y, posteriormente, se hará una especial incidencia en sus valores mínimos, pues son los que fijarán el rendimiento óptimo del algoritmo.

Escenario 1: Porcentaje de ocupación de 0,1% y 2 obstáculos

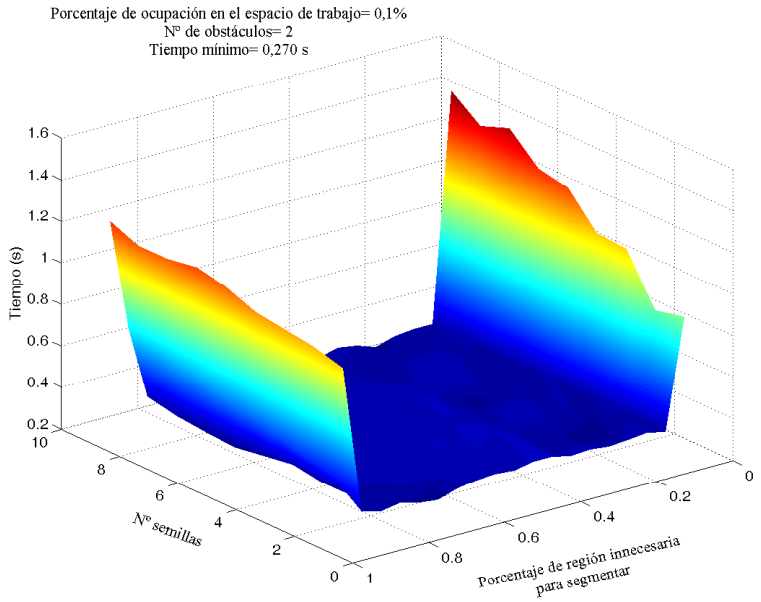
En esta situación, cada uno de los obstáculos ocupará el 0,05% del total del espacio de trabajo, lo que equivale a 524 pixels. Esto indica que tendrán un tamaño suficiente para "producir" agrupaciones de pixels. Al encontrarse el 99,9% del espacio libre de obstáculos se pueden realizar grandes agrupaciones de pixels libres. Además, una gran región no necesitará ser evaluada a gran resolución.

En la figura 8.1 se muestran los perfiles de tiempo consumido y memoria requerida en la evaluación del C-espacio para un espacio de trabajo con estas características. En dichos perfiles se muestra la dependencia con los dos parámetros utilizados para la segmentación.

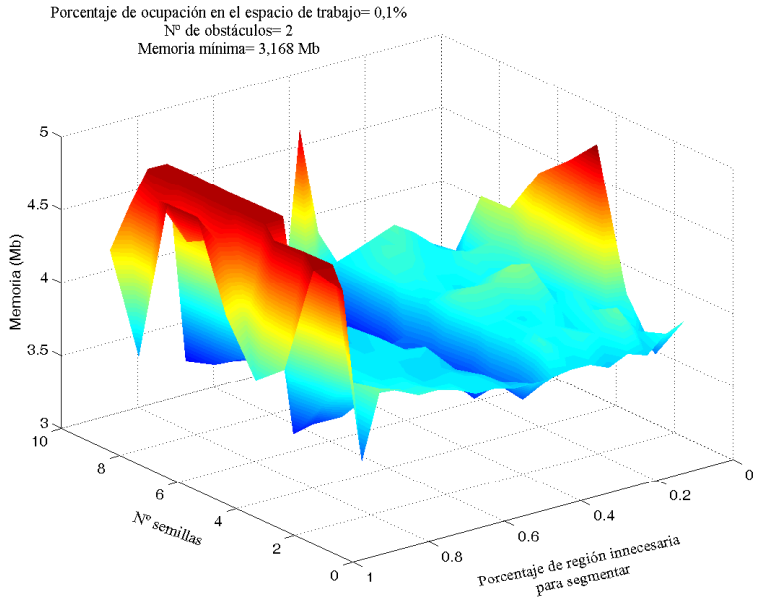
En cuanto al perfil de tiempos se puede observar que:

- Cuando el criterio de segmentación es muy restrictivo, lo que equivale a un umbral de región innecesaria muy grande (cercano al 100% de la región total), el tiempo tiende a un valor máximo. Esto es debido a que se consume mucho tiempo evaluando regiones sin información (completamente vacías o llenas).

- Cuando la condición de segmentación es poco restrictiva tambien se puede observar un gran aumento en el tiempo de cálculo debido a que se realiza la segmentación en cuanto existe la más mínima región innecesaria. Por ello, se puede deducir que la mayor parte del tiempo se pierde intentando realizar la segmentación.



(a)



(b)

Figura 8.1: (a) Tiempo y (b) memoria utilizados en el cómputo del C-espacio para un espacio de trabajo con únicamente 2 obstáculos que ocupan el 0,1% del espacio total

Otro aspecto importante a tener en cuenta es en qué iteración del algoritmo se inicia la segmentación. En este escenario, las regiones innecesarias, al ser tan grandes, se empezarán a eliminar en los primeros pasos del algoritmo, siendo únicamente necesario realizar pequeños refinamientos según se alcance mayor resolución. Por lo tanto, las segmentaciones siempre se realizan sobre regiones representadas por pequeñas cantidades de pixels. Así, la dependencia con el número de semillas utilizadas para encontrar el *MRI* es prácticamente inapreciable debido a que la búsqueda se realiza sobre un conjunto pequeño.

Observando el perfil de memoria se puede ver que, cuando el criterio de segmentación es muy restrictivo, se requiere una mayor cantidad de memoria. Esto concuerda con la conclusión obtenida del perfil de tiempos que indica que prácticamente no se segmenta y se realiza la convolución para todo el espacio.

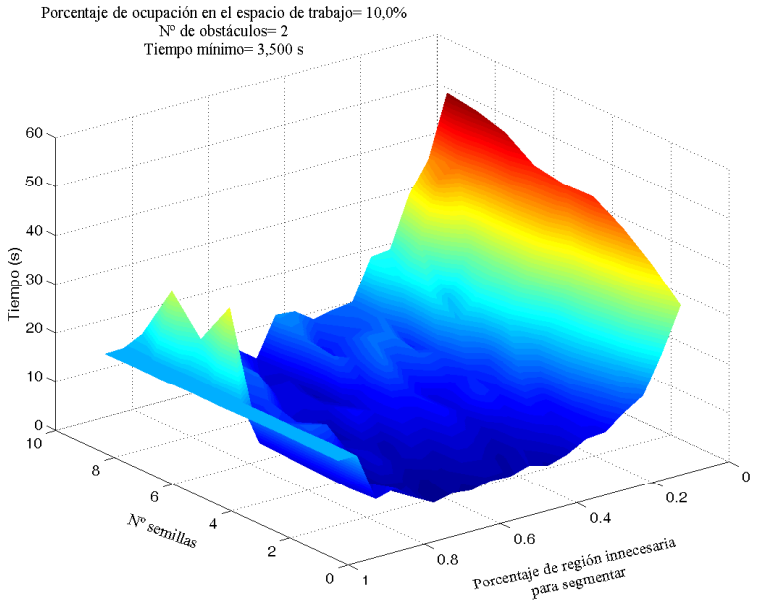
Escenario 2: Porcentaje de ocupación de 10% y 2 obstáculos

En este espacio de trabajo, cada uno de los obstáculos ocupa el 5% de la región. En cuanto a la región libre, aunque representa menor superficie que el caso anterior (el 90%) por haber únicamente dos obstáculos, sigue estando muy agrupada.

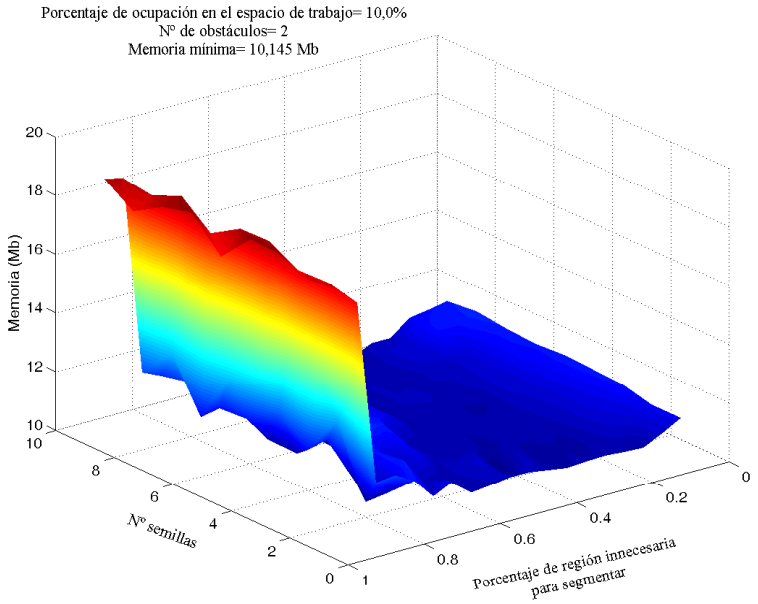
Si se analizan los perfiles con respecto a los parámetros de la segmentación (figura 8.2), se pueden observar los efectos anteriormente descritos de una forma más pronunciada.

El perfil de tiempos presenta una característica similar al caso anterior, encontrándose más acentuado el incremento en el tiempo cuando el criterio para segmentar es poco restrictivo. Esto es debido a que, como las regiones en el interior de los obstáculos también son innecesarias, y éstas tienen un tamaño considerable (cada uno de los obstáculos ocupa el 5%), el proceso de segmentación debe eliminarlas. Por esta razón, se tiene que realizar gran cantidad de segmentaciones para ajustarse al contorno del C-obstáculo, lo que implica un excesivo incremento de tiempo.

En cuanto a la iteración en la que se comienzan a realizar las segmentaciones, al considerar el interior de los obstáculos como regiones innecesarias los criterios de segmentación en dichas regiones no se cumplen hasta que no se alcanza una mayor resolución. Por lo tanto, las regiones a segmentar tienen un número mayor



(a)



(b)

Figura 8.2: (a) Tiempo y (b) memoria utilizados en el cómputo del C-espacio para un espacio de trabajo con únicamente 2 obstáculos que ocupan el 10% del espacio total

de pixels, por lo que es más costoso encontrar el *MRI*. Así, se puede observar cómo aparece una mayor dependencia con el número de semillas utilizadas.

Analizando la dependencia de la memoria, se observa la reducción drástica en el consumo de memoria cuando se permiten hacer segmentaciones.

Escenario 3: Porcentaje de ocupación de 0,1% y 20 obstáculos

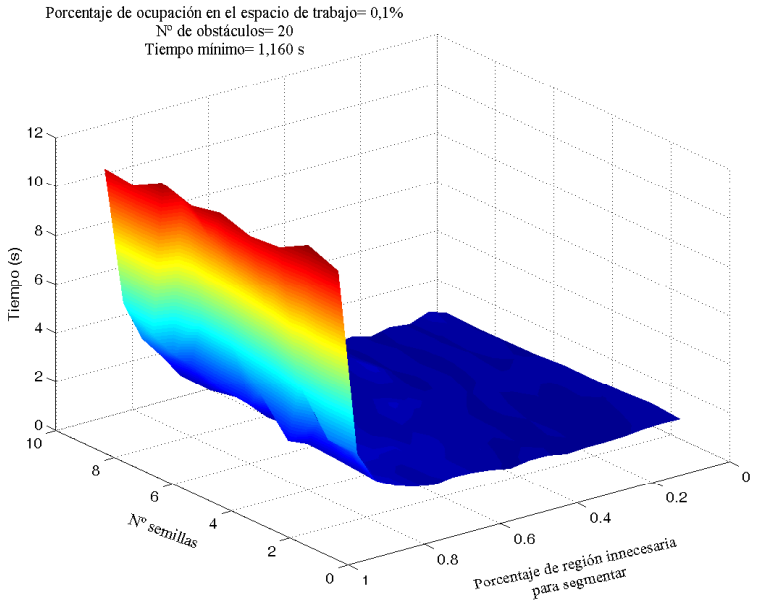
En este caso, al aumentar el número de obstáculos, la región ocupada por cada uno de los obstáculos es menor (0,005%). Así, cada uno de ellos estará representado, como media, por 52 pixels, por lo que las agrupaciones son muy pequeñas. La región vacía, aunque es muy grande, requerirá de numerosas segmentaciones para eliminarse, al existir múltiples obstáculos dispersos por el espacio.

La disminución en el tiempo de cálculo aparece (figura 8.3), como en los casos anteriores, cuando se suaviza el criterio de segmentación. Sin embargo, no se presenta el aumento que se observaba en los escenarios anteriores cuando el criterio de segmentación era tan débil que inducía a la segmentación en casi cualquier circunstancia. Este resultado se puede interpretar como el caso opuesto al segundo escenario: cuando los obstáculos tienen un tamaño muy pequeño, apenas aparecerán regiones innecesarias en su interior. Por esta razón la segmentación tiende a separar los diferentes C-obstáculos (eliminar la región libre que hay entre ellos), sin necesidad de tomar fracciones de cada uno de ellos para adaptarse a su contorno. En cuanto al consumo de memoria el análisis muestra la misma tendencia que en los escenarios anteriores.

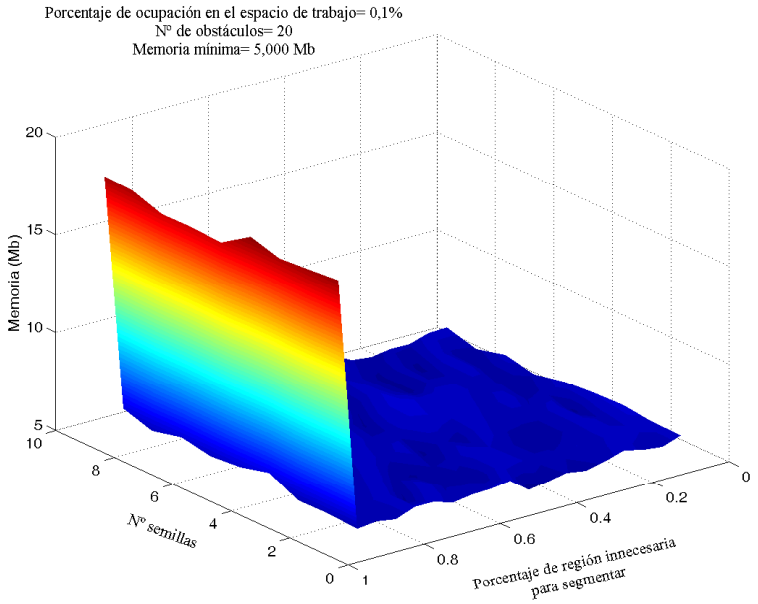
También se puede observar la escasa dependencia con el número de semillas. Este efecto también es atribuible al pequeño tamaño de los obstáculos: como su región interior es muy pequeña no se llega a segmentar nunca, mientras que la región libre comienza a eliminarse en los primeros pasos del algoritmo. Por lo tanto, como en el primer escenario, la evaluación del *MRI* se realiza de forma rápida al tener pocos pixels.

Escenario 4: Porcentaje de ocupación de 10% y 20 obstáculos

En este último caso, los obstáculos son de un tamaño lo suficientemente grande como para que su interior influya en el proceso de segmentación, por

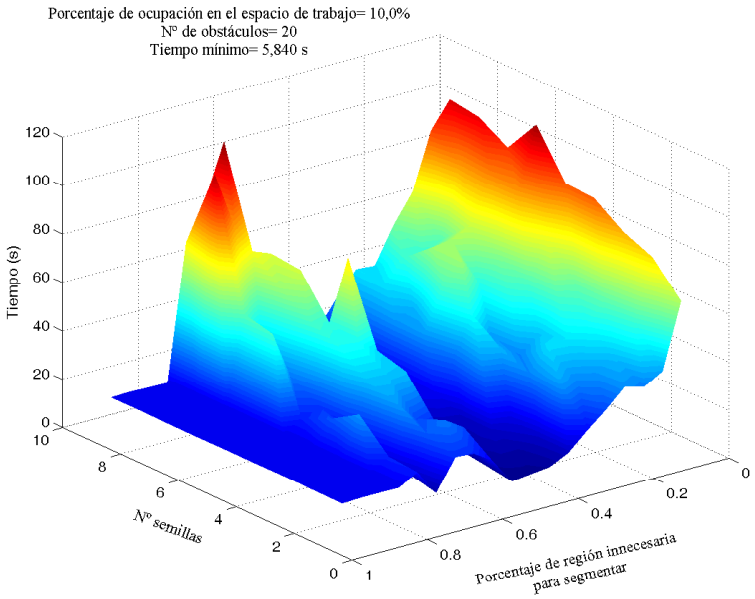


(a)

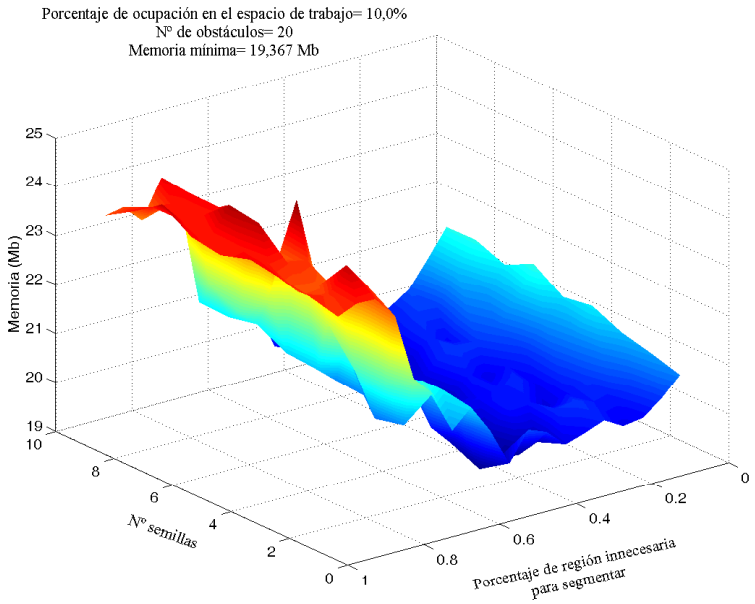


(b)

Figura 8.3: (a) Tiempo y (b) memoria utilizados en el cómputo del C-espacio para un espacio de trabajo con 20 obstáculos que ocupan el 0,1% del espacio total



(a)



(b)

Figura 8.4: (a) Tiempo y (b) memoria utilizados en el cómputo del C-espacio para un espacio de trabajo con 20 obstáculos que ocupan el 10% del espacio total

tratarse de regiones innecesarias. Por otro lado, el espacio vacío además de no ser muy elevado está muy disperso, ya que existen muchos obstáculos.

Si analizamos los perfiles obtenidos en la figura 8.4 se puede observar que el perfil de tiempos difiere de forma considerable con respecto a los de los escenarios anteriores. Se observa un gran pico cuando el criterio de segmentación es que se encuentre un 70% de región innecesaria. Para explicarlo es necesario tener en cuenta que el interior de los obstáculos también es región innecesaria. Por esta razón, al ocupar los obstáculos una región considerable, la *RDI* está muy dispersa, lo que implica que se requieran numerosas segmentaciones para obtener regiones que cumplan las propiedades exigidas. Además, en este punto, se puede apreciar un claro aumento del tiempo con el número de semillas utilizado que no se había apreciado anteriormente. Este aumento de tiempo se debe a que el interior de los obstáculos tiene un tamaño suficiente para ser eliminado, pero solamente alcanza los criterios de segmentación a resoluciones elevadas. Por lo tanto, se requiere una gran cantidad de tiempo para encontrar el *MRI*, que se ve agravado con el número de semillas.

Cuando el criterio de segmentar se hace algo menos restrictivo, la segmentación se comienza a realizar en etapas anteriores para el interior de los obstáculos reduciendo el tiempo necesario para ésta, y eliminando la dependencia con el número de semillas. El caso límite se obtiene de nuevo cuando se aplica la segmentación en cuanto existe la más mínima porción de región innecesaria. En este caso el tiempo vuelve a aumentar debido a que se requiere un gran tiempo en la segmentación.

Si se analizan las necesidades de memoria se puede comprobar que coincide con los resultados obtenidos previamente. Además es posible apreciar que el punto en el que se comienza a segmentar la memoria requerida comienza a disminuir.

Consideraciones finales

A partir de los escenarios anteriores se puede concluir que es necesario encontrar aquellos valores que consiguen que el proceso de segmentación se realice de forma eficiente. Si no es así, se pueden producir situaciones en las que el

consumo de tiempo de cálculo, o de memoria, se ve enormemente alejado de su valor óptimo.

Para determinar de forma analítica cuándo una segmentación es apropiada o no, es necesario determinar el tiempo que requiere la realización de la FFT de cada una de las *RDI*, así como el tiempo necesario para realizar la segmentación. El tiempo necesario en realizar la FFT de una matriz depende de forma compleja[1] con las dimensiones de las matrices.

Si a esto se une el hecho de que no hay una relación directa entre las características del espacio de trabajo y los parámetros óptimos del algoritmo, se puede deducir que la dependencia funcional es imposible determinarla de forma analítica.

Así, al no ser posible obtener de forma analítica una dependencia funcional entre los espacios de trabajo y los parámetros óptimos para el algoritmo, sería adecuado utilizar herramientas de minería de datos. Éstas permiten la extracción de estas dependencias a partir de la experiencia. De modo que se puedan determinar los parámetros a utilizar para una situación concreta a partir de los obtenidos en experiencias anteriores.

Estos parámetros, además de depender del espacio de trabajo, también dependen de las dimensiones del robot. Sin embargo, esa dependencia funcional no tiene una gran importancia en cuanto que en una aplicación práctica el robot siempre será el mismo. Por lo tanto, únicamente es necesario determinar los parámetros óptimos para un robot determinado. De este modo se debe determinar el procedimiento a seguir para realizar la evaluación de los C-espacios utilizando este algoritmo.

8.2 Procedimiento de evaluación

La dependencia funcional que relaciona el espacio de trabajo con los parámetros óptimos del algoritmo, para un robot determinado, a partir de resultados experimentales se puede extraer utilizando diferentes técnicas. Entre ellas se encuentran las técnicas estadísticas, como pueden ser la regresión lineal multivariable, o la regresión no lineal multivariable para dependencias más complejas.

[1] Cuando las dimensiones son potencias de 2 es conocida, pero cuando las dimensiones no lo son depende en gran medida de la descomposición que se pueda realizar.

Otra posibilidad es la utilización de algoritmos genéticos, que también son ampliamente utilizados para la aproximación de funciones. Sin embargo, estos métodos consisten en la estimación de unos parámetros para que una determinada función se aproxime a los resultados. Y por lo tanto requieren la asunción de alguna dependencia determinada.

El método que se ha considerado más apropiado en nuestro trabajo es el de las redes neuronales, pues presentan una gran capacidad para aprender y generalizar, además de las posibilidades de experimentar con diferentes arquitecturas de redes o algoritmos de entrenamiento.

En la figura 8.5 se muestra un esquema del procedimiento a seguir que proponemos en este trabajo. Antes de proceder a aplicar el algoritmo es necesario extraer las características del espacio de trabajo (como puede ser la dispersión). A partir de dichas características una red neuronal artificial determinará los parámetros de la segmentación (% de región innecesaria, número de semillas). Una vez obtenidos estos parámetros ya se puede aplicar el algoritmo con garantías suficientes de que se están utilizando los valores óptimos.

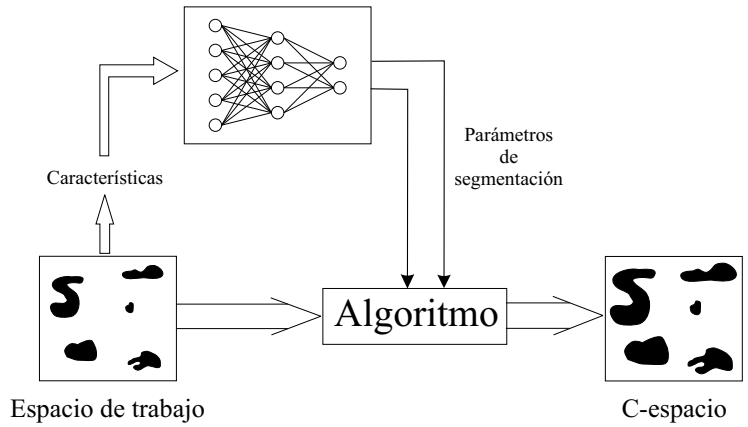


Figura 8.5: Procedimiento para la evaluación del C-espacio

Esta red neuronal determinará los valores que se deben utilizar para evaluar el C-espacio de un robot concreto. Si se desea evaluar para otro robot diferente, la red neuronal debe ser substituida por otra acorde al nuevo robot.

8.3 La red neuronal

Al pretender determinar una dependencia funcional mediante la red neuronal, ésta debe disponer previamente de los valores óptimos obtenidos para algunos espacios de trabajo. Por lo tanto, debe tratarse de una red neuronal supervisada. De entre las posibles, se ha elegido el perceptrón multicapa por ser un tipo de red neuronal que ha sido ampliamente utilizado para representar funciones con un alto grado de no linealidad.

En primer lugar es necesario determinar cuáles serán las entradas de la red neuronal, las cuales deben caracterizar el espacio de trabajo. Una vez que se tengan las entradas de la red, se deberá realizar el diseño de la misma. Posteriormente, para realizar la fase de aprendizaje es necesario establecer un conjunto de datos de entrenamiento que sea suficientemente significativo para que la red pueda extraer la información requerida. Y, por último, para verificar su correcto aprendizaje se tiene que analizar los resultados obtenidos por ésta, para un conjunto de datos desconocidos para la red.

8.3.1 Caracterización del espacio de trabajo

La caracterización de la dispersión del espacio de trabajo se puede abordar desde diferentes puntos de vista. Un método podría consistir en el análisis de la transformada de Fourier del espacio de trabajo. A partir de la transformada de Fourier se puede extraer una estimación de la dispersión mediante del ancho de banda y las separaciones entre los picos del espectro. Otra posible estimación de la dispersión puede venir dada por la entropía del espacio de trabajo, lo cual proporciona una medida del orden o desorden que presentan los datos. Cualquiera de estos métodos implica un alto procesamiento de los datos, lo que significa un alto coste para una tarea previa a la ejecución del algoritmo como es la sintonización óptima.

En la estructura del *quadtree* que representa al espacio de trabajo se encuentra implícita esta información. Esto es fácilmente explicable desde el punto de vista de que el *quadtree* consiste en la agrupación de pixels que presentan características similares. Una gran dispersión de los datos producirá una gran cantidad de hojas en el último nivel del *quadtree*, que representa la mayor reso-

lución. El caso extremo se corresponde con un tablero de ajedrez, en el que se obtendrían el mayor número posible de nodos hoja. Por el contrario, un espacio muy agrupado presentará un gran número de nodos hoja en los primeros niveles. En la figura 8.6 se muestra cómo varía el número de nodos hoja en los diferentes niveles para un espacio muy disperso, como es el tablero de ajedrez (figura 8.6(a)), o para otro más agrupado (figura 8.6(b)). De esta forma, a partir de las ramificaciones del árbol es posible estimar la dispersión del espacio de trabajo. Por lo tanto, lo único necesario es contabilizar la proporción de nodos hoja que existen en cada nivel del árbol.

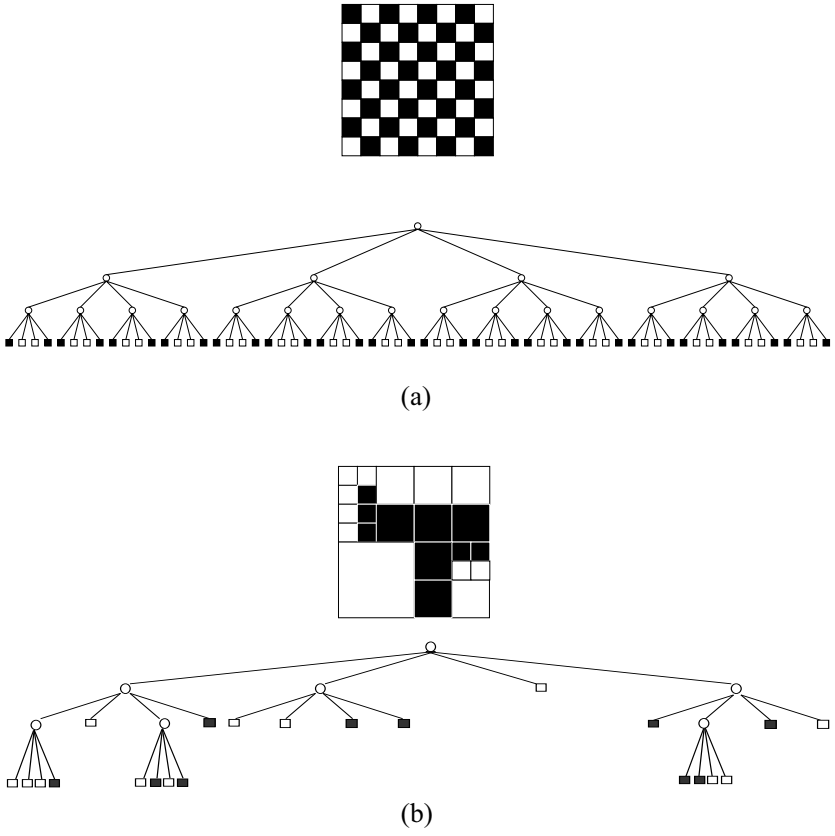


Figura 8.6: Presencia de los nodos hoja en los diferentes niveles del árbol dependiendo de su dispersión

Además, estas hojas pueden tomar dos valores (ocupado, libre). Es interesante tener información, no sólo de la dispersión sino, de la cantidad de espacio

libre y ocupado por ellas, por lo que se debe hacer un recuento separado de las hojas que representan zonas ocupadas o libres.

Entonces, es necesario contabilizar tanto las hojas vacías como las llenas en cada uno de los niveles del árbol. Para que estos valores puedan ser utilizados con mayor facilidad es conveniente que se encuentren normalizados, por lo que se pueden considerar en términos de porcentaje. Así, para cada nivel i se considerará el porcentaje de hojas libres y el porcentaje de hojas ocupadas:

$$\frac{\text{Nº hojas vacías}}{2^i}, \quad \frac{\text{Nº hojas llenas}}{2^i}$$

Los resultados así obtenidos, en los niveles de resoluciones elevados, van a tener por lo general valores muy pequeños. Esto es debido a que una gran parte del espacio ya se ha considerado en los niveles anteriores, y por lo tanto se van reduciendo las hojas que puede haber en niveles sucesivos. Al ser estos valores muy pequeños se requerirá una gran precisión para diferenciar unos de otros.

Otra forma de contabilizar las hojas, que elimina el problema de trabajar con valores tan pequeños, consiste en determinar el porcentaje de hojas libres y ocupadas únicamente respecto a aquellos nodos que existan en el nivel, sean hojas o no. Entonces para cada nivel se tiene

$$\frac{\text{Nº hojas vacías}}{\text{Nº nodos en el nivel}}, \quad \frac{\text{Nº hojas llenas}}{\text{Nº nodos en el nivel}}$$

Es fácil de comprobar que ambos contienen la misma información, pero, con la ventaja que los valores evaluados siguiendo el segundo procedimiento, en cada nivel, varían entre 0 y 1.

A continuación vamos a considerar diversos espacios de trabajo y vamos a interpretar como el conjunto de porcentajes de nodos hojas por nivel caracteriza cada espacio de trabajo (figura 8.7). Para un escenario con únicamente 2 obstáculos que ocupan un 0,1% del total (figura 8.7(a)) se tiene una gran región libre, lo que se traduce en nodos hojas vacíos en los primeros niveles del árbol. Como los obstáculos son muy pequeños, los nodos hojas ocupados tienen que aparecer en niveles muy profundos. Si se compara con un espacio de trabajo en el que los 2 obstáculos ocupan el 10% (figura 8.7(b)), se puede observar, en primer lugar, que los nodos ocupados aparecen mucho más pronto, puesto que las regiones que ocupan los obstáculos son mayores. En segundo lugar, la región

libre se ve reducida, por lo que los nodos en los primeros niveles se ven reducidos. En cuanto a la dispersión, como únicamente hay dos obstáculos la región libre está muy agrupada, lo que también hace que los nodos vacíos aparezcan en los primeros niveles.

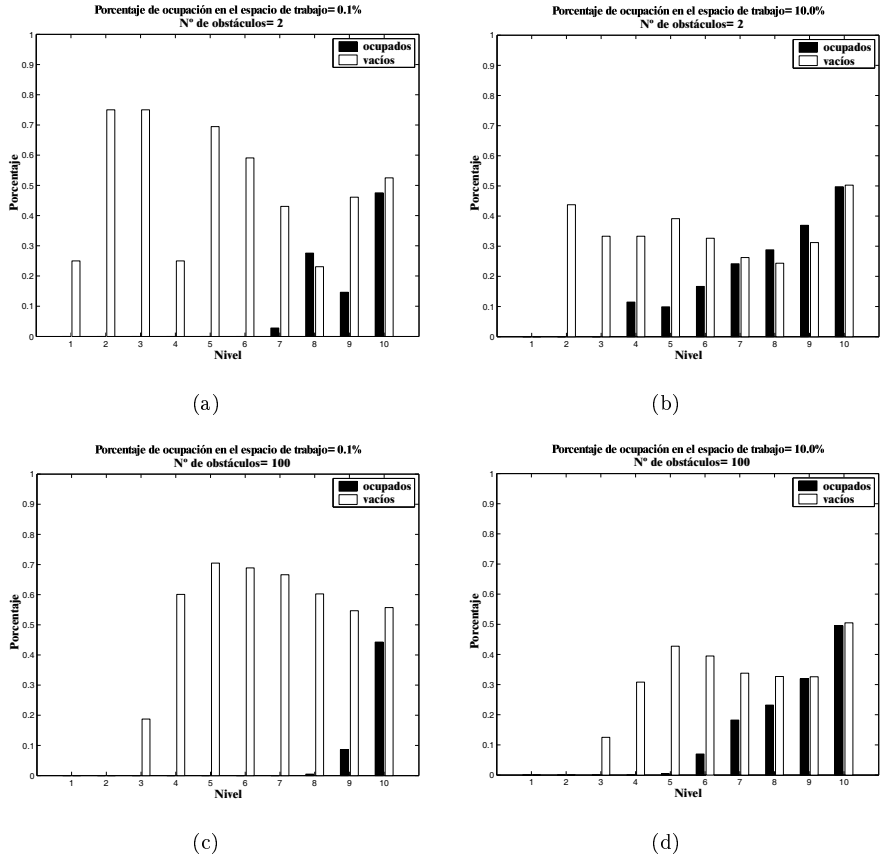


Figura 8.7: Distribución de nodos hoja ocupados y vacíos por nivel en el árbol para diferentes espacios de trabajo

Para ver claramente la repercusión de la dispersión se han considerado también espacios de trabajo compuestos por 100 obstáculos. Si estos obstáculos ocupan el 0,1 % del espacio total (figura 8.7(c)) se observaa, en primer lugar, que los nodos que representan los obstáculos son de tamaño muy pequeño, prácticamente solo aparecen en el último nivel. En segundo lugar, al estar muy dispersos, no existen grandes regiones vacías, lo que conlleva que los nodos vacíos aparezcan en niveles más profundos, en comparación con la figura 8.7(a) en la que el espacio libre es similar. Si consideraremos un espacio de trabajo con

100 obstáculos que ocupan el 10% (figura 8.7(d)), se puede apreciar como los nodos ocupados aparecen en niveles anteriores al escenario previo, por ser obstáculos de mayor tamaño. Respecto a los nodos vacíos se observa que, aunque en menor cantidad, también aparecen a partir del nivel 3 debido a la dispersión. Comparando este aspecto con el escenario de la figura 8.7(b) se puede ver la diferencia del nivel en que aparecen los nodos y, sobre todo, la gran diferencia de región que representan en dichos niveles.

Por lo tanto, parece claro que con estos valores se puede caracterizar la dispersión existente en el espacio de trabajo, no sólo en cuanto al espacio libre, sino también en cuanto al tamaño de los obstáculos y la región ocupada.

8.3.2 Diseño de la red neuronal

Así, para diseñar una red neuronal que determine los parámetros de sintonización óptimos en función del espacio de trabajo, los valores de los porcentajes de nodos hoja en cada nivel deben ser las entradas a la red neuronal. Así mismo, las salidas de la red neuronal serán los parámetros óptimos para realizar la segmentación.

Teniendo en cuenta las consideraciones anteriores se ha diseñado un perceptron multicapa con 20 neuronas en la capa de entrada[2], 50 neuronas en la capa oculta y 2 neuronas en la de salida. Como funciones de activación de las neuronas se han considerado sigmoides por ser las funciones que se suelen utilizar para este tipo de redes neuronales.

Para conseguir que la red neuronal aprenda las salidas que debe producir para cada conjunto de entradas se tiene que utilizar un tipo de entrenamiento supervisado. Entre los diferentes tipos que se pueden considerar se ha utilizado una variación del algoritmo de retropropagación, el algoritmo de retropropagación con momento. La ventaja de introducir el momento radica en que el aprendizaje es más rápido, puesto que incluye en cada paso de entrenamiento la tendencia observada en los pasos anteriores

La retropropagación con momento es el algoritmo de entrenamiento que se

[2]Se han considerado espacios de trabajo con discretizados con 1024×1024 pixels, por lo que el *quadree* tiene 10 niveles. Tomando dos entradas por cada nivel para determinar el porcentaje de nodos ocupados y libres se obtienen las 20 entradas de la red.

mostró en el apartado 7.1.1.5 en el que los valores de los pesos en cada paso de entrenamiento se modifican según la ecuación:

$$w_{ik}(t+1) = w_{ik}(t) + \eta\delta_k y_i + \mu(w_{ik}(t) - w_{ik}(t-1)) \tag{8.1}$$

donde δ_k es la sensibilidad de la neurona calculada de la forma

$$\delta_i = \begin{cases} (d_i - y_i)F_i'(s_i) \text{ si la neurona } i \text{ es de la capa de salida} \\ F_i'(s_i)\sum_{k=1}^{M} w_{ik}\delta_k \text{ si la neurona } i \text{ es de la capa de oculta} \end{cases}$$

Los parámetros η y μ son los que determinan la velocidad y calidad del proceso de aprendizaje. Así,

- η representa el parámetro de entrenamiento y determina el tamaño del paso en la dirección opuesta al gradiente de la superficie de error.
- μ es el momento, y determina la importancia que tiene el cambio anterior en cada paso de entrenamiento.

Además de estos términos, para mejorar el comportamiento del algoritmo de entrenamiento se incluyen en la equación 8.1 otros dos más:

- Un término constante (c) sumado a la derivada de la función de activación, para evitar que las regiones planas en las superficies de error.
- El error máximo propagable $d_k - y_k$, para evitar correcciones muy bruscas en los pesos.

8.3.3 Conjunto de datos de entrenamiento

El conjunto de datos de entrenamiento se ha creado con diferentes espacios de trabajo con unas dimensiones de $6,4\ m \times 6,4\ m$ y una resolución de $6,25\ mm$ en cada lado. De tal modo que se pueden representar mediante matrices de 1.024×1.024 pixels.

Cada uno de los espacios de trabajo se ha generado de forma aleatoria, de modo que contenga obstáculos con formas tales como rectángulos, elipses y triángulos, pudiendo superponerse para formar figuras más complejas. También, con el objeto de que estos espacios de trabajo reflejen la mayor cantidad de situaciones posibles se han creado teniendo en cuenta dos criterios: el espacio

ocupado por los obstáculos y la dispersión que presentan. De este modo, los espacios seleccionados presentan porcentajes de ocupación por los obstáculos de $0,1$, $0,2$, $0,5$, 1, 2, 5 y 10% con un número de obstáculos que varían entre 2, 5, 10, 20, 50 y 100. Se han generado 10 espacios con cada combinación de ocupación y número de obstáculos, obteniendose un total de 420 espacios de trabajo.

Para evaluar el C-espacio se ha considerado un robot de $37,5\ cm \times 21,25\ cm$, lo que supone aproximadamente el $2‰$ del espacio de trabajo.

Después, se han evaluado los C-espacios correspondientes a cada espacio de trabajo con el algoritmo propuesto utilizando gran cantidad de parámetros de sintonización con objeto de determinar los que proporcionan un comportamiento óptimo. De este modo, se obtiene la salida que debe proporcionar la red neuronal para cada uno de los espacios de trabajo.

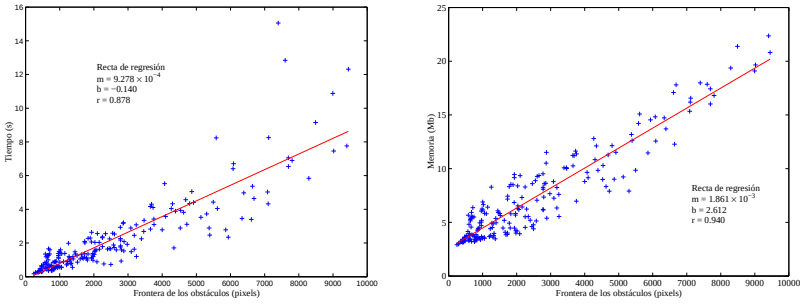
8.3.3.1 Representatividad de los datos de entrenamiento

Una vez que se ha establecido el conjunto de datos de entrenamiento, es decir, los espacios de trabajo con sus correspondientes parámetros óptimos, es necesario comprobar que es un conjunto de datos con un comportamiento que puede ser aprendido. Por lo tanto, en este apartado se pretende comprobar que existe algún tipo de relación entre los espacios de trabajo considerados y los valores mínimos de tiempo y memoria alcanzado para cada uno de ellos.

Según se comentó en el apartado 3.1, el teorema de complejidad del *quadtree* acota el número de nodos necesarios para representar una región en forma de *quadtree* como $O(p)$, siendo p el perímetro de la región a representar. Así, también está acotada la cantidad de memoria necesaria para su representación. Como la mayor parte de los algoritmos generales que hacen uso de estructuras de *quadtrees* suelen recorrer todos los nodos del árbol, la complejidad de estos algoritmos también está determinada por el perímetro de los objetos que representan.

Por estas razones vamos a comprobar si los resultados óptimos del algoritmo propuesto tienen algún tipo de dependencia con el perímetro de los obstáculos en el espacio de trabajo como indica el teorema de complejidad del *quadtree*.

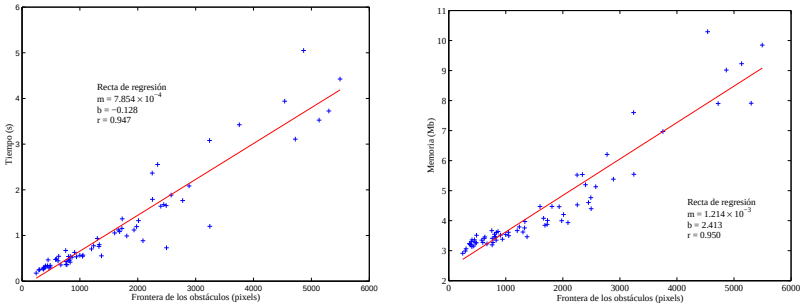
Para ello en la figura 8.8 se han representado los valores mínimos de tiempo



(a) (b)

Figura 8.8: Dependencia del tiempo de cálculo (a) y memoria (b) con la frontera de los obstáculos para todos los espacios de trabajo.

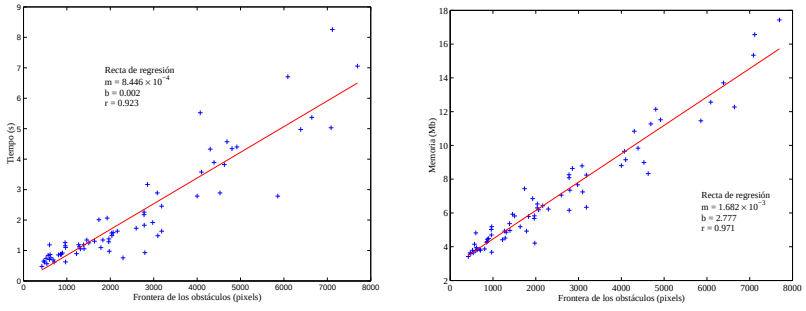
y memoria alcanzados para todos los espacios de trabajo por el algoritmo con respecto al perímetro de los obstáculos que contiene cada uno de ellos. En ella se puede observar el claro comportamiento lineal tanto para el tiempo como para la memoria. En la figura se muestra también la recta de regresión lineal con sus parámetros: m, la pendiente; b, la ordenada en el origen; y r, el coeficiente de correlación. Este comportamiento lineal, además de adecuarse al teorema de complejidad del *quadtree*, parece ser coherente con el hecho de que el algoritmo propuesto pretende limitarse únicamente al contorno de los obstáculos, evitando calcular a grandes resoluciones tanto la región libre como el interior de los obstáculos.



(a) (b)

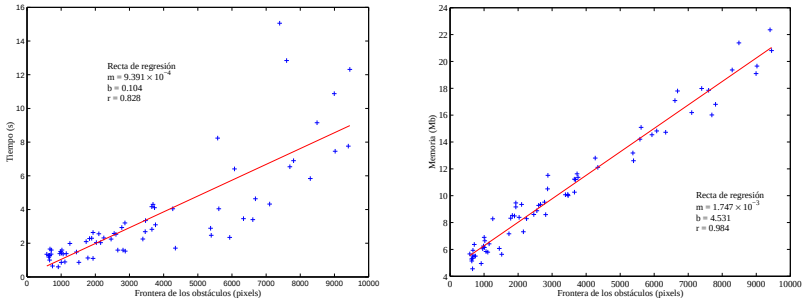
Figura 8.9: Dependencia del tiempo de cálculo (a) y memoria (b) con la frontera de los obstáculos para los espacios de trabajo con 2 obstáculos.

Si se analizan el tiempo de cálculo y la memoria, clasificando los espacios



(a) (b)

Figura 8.10: Dependencia del tiempo de cálculo (a) y memoria (b) con la frontera de los obstáculos para los espacios de trabajo con 10 obstáculos.



(a) (b)

Figura 8.11: Dependencia del tiempo de cálculo (a) y memoria (b) con la frontera de los obstáculos para los espacios de trabajo con 20 obstáculos.

de trabajo en función del número de obstáculos (figuras 8.9, 8.10, y 8.11) se puede comprobar cómo afecta el número de obstáculos en la recta de regresión. La dependencia en la recta de regresión con el número de obstáculos, y no únicamente con su frontera, puede deberse a dos factores:

- En el algoritmo no se está utilizando solamente el *quadtree* que representa el espacio de trabajo, sino que también está presente el *quadtree* del C-espacio. Por lo tanto, existirá una dependencia con el tamaño de la frontera de los C-obstáculos. Esto conduciría a una dependencia con el número de C-obstáculos.

  Al haberse generado los espacios de trabajo de forma aleatoria pueden aparecer obstáculos próximos que en el C-espacio se juntan para formar

uno sólo, y por tanto, se reduce la frontera, o por el contrario estar alejados, por lo que la frontera en el C-espacio aumenta.

- Si se tienen espacios de trabajo con varios obstáculos de pequeño tamaño, su frontera puede ser semejante al de un único obstáculo mayor. Como el proceso de segmentación tiene una gran influencia en el tiempo de ejecución del algoritmo, al aumentar la dispersión (mayor número de obstáculos) se incrementaría el tiempo de cálculo, aunque el tamaño de la frontera fuera semejante. La dispersión no viene reflejada en el número de pixels de la frontera.

De cualquier modo, la presencia de la linealidad es una garantía para considerar que los datos de entrenamiento de la red tienen consistencia y por lo tanto la red puede llegar a aprenderlos.

8.3.4 Resultados del entrenamiento

Para entrenar la red neuronal se han tomado entre el 70% y el 80% de los espacios de trabajo expuestos en el apartado anterior. El resto de los espacios de trabajo se han utilizado para comprobar la capacidad que ha alcanzado la red neuronal para encontrar los parámetros óptimos a utilizar en el algoritmo y la capacidad de generalización para espacios de trabajo desconocidos.

Una vez entrenada la red neuronal es necesario comprobar el porcentaje de aciertos y fallos para el conjunto de datos de prueba. Por lo tanto, debemos establecer el criterio que va a ser utilizado para determinar si la red neuronal acierta en la determinación de los parámetros o no. Al tratarse de una red neuronal que no clasifica patrones sino que aproxima una función desconocida, el criterio para determinar si la red funciona correctamente debe ser la certeza con la que proporciona un valor próximo al esperado.

En este caso para el que se está utilizando la red neuronal, no se puede olvidar que el objetivo último es encontrar unos parámetros con los que el algoritmo se comporte de forma óptima. Si se observan los perfiles de tiempo obtenidos en el apartado 8.1 se puede llegar a la conclusión que en muchos casos no es necesario obtener valores precisos de los parámetros a utilizar puesto que la diferencia de tiempos entre unos y otros es muy pequeña. Por lo tanto, para establecer el

criterio de acierto de la red neuronal es mucho más apropiado comprobar si el algoritmo tarda un tiempo próximo al óptimo con los parámetros propuestos por la red.

Para medir si el algoritmo tarda un tiempo aproximado al tiempo mínimo se ha optado por medir el porcentaje de error cometido con respecto al tiempo mínimo. También, es necesario establecer qué porcentaje de error es admisible como válido. Si se observan las figuras del apartado 8.1, y sobre todo la figura 8.4, se puede ver cómo pequeñas variaciones alrededor del mínimo pueden producir incrementos grandes sobre el tiempo de cálculo. En el escenario de la figura 8.4 el tiempo mínimo ($5,849s$) se encuentra alrededor del porcentaje de región innecesaria del 50%, por lo que pequeñas variaciones pueden hacer que se llegue fácilmente a un tiempo de $20s$.

Por las razones anteriormente mencionadas se han considerado dos criterios para determinar el acierto de la red:

1. Si el error es menor al 50% del tiempo mínimo se considera acierto, si no es fallo

2. Si el error es menor al 100% del tiempo mínimo se considera acierto, si no es fallo

Los resultados obtenidos para el conjunto de datos de prueba son los siguientes:

- Con el criterio 1 se han obtenido el 71% de aciertos

- Con el criterio 2 se han obtenido el 90% de aciertos

Con ambos criterios se observa que el entrenamiento de la red ha sido correcto

8.4 Tiempos de cálculo y memoria para la plataforma sin giros

Una vez que ya se ha establecido el procedimiento completo para evaluar el C-espacio, se va a realizar una comparación de los resultados obtenidos con los que proporciona el método de convolución matricial.

Puesto que el tiempo de cálculo para el procedimiento propuesto de evaluación del C-espacio depende de las características del espacio de trabajo se han considerado diversos casos. En la tabla 8.1 se encuentran desglosadas las medias de los tiempos de cálculo para los espacios de trabajo dependiendo del número de obstáculos que tengan y el porcentaje del espacio que ocupen. El método de evaluación del C-espacio mediante la convolución matricial con la FFT es independiente del espacio de trabajo y tarda 0,16 segundos.

Obst. \ Ocup.	0,1	0,2	0,5	1	2	5
2	0,30	0,41	0,50	0,67	1,30	1,59
5	0,62	0,68	1,07	1,60	2,26	3,39
10	0,74	0,91	1,37	1,81	2,51	3,74
20	1,26	1,30	1,85	2,25	3,25	4,22
50	5,19	6,13	8,36	10,78	13,37	16,28

Tabla 8.1: Tiempos (s) de cálculo del C-espacio con el procedimiento propuesto para diferentes tipos de espacios de trabajo. Evaluar el C-espacio mediante la convolución matricial con la FFT lleva 0,16 s.

Sin embargo, el propósito del procedimiento que se ha desarrollado pretende reducir la cantidad de memoria necesaria para realizar la evaluación. En este sentido, se muestra en la tabla 8.2 el porcentaje de memoria necesario frente al método matricial que requiere 21Mb.

Como se puede comprobar, se producen reducciones considerables en la mayor parte de los casos. Por lo tanto se ha conseguido el objetivo principal buscado al introducir el uso de *quadtrees* y *octrees* en la evaluación del C-espacio que consistía en la gran reducción de memoria. Sin embargo, se ha perdido la independencia con el espacio de trabajo que presentaba el método matricial puesto que evaluaba a gran resolución tanto las zonas libres como las que no.

Obst. \ Ocup.	0,1	0,2	0,5	1	2	5
2	15,62%	15,90%	16,57%	17,52%	21,57%	24,05%
5	20,90%	24,28%	25,05%	26,86%	29,05%	33,57%
10	22,62%	26,62%	28,05%	28,33%	36,57%	44,52%
20	26,10%	30,14%	38,38%	43,71%	52,76%	71,52%
50	40,67%	47,90%	57,80%	70,57%	77,00%	96,95%

Tabla 8.2: Porcentaje de memoria requerida por el procedimiento para evaluar el C-espacio en comparación con la requerida por el método matricial

9

Conclusiones y Trabajos Futuros

En este capítulo se van a destacar, las principales aportaciones realizadas en este trabajo de investigación, así como las conclusiones más importantes que de él se pueden extraer. Aunque éste tiene un carácter completo, a partir de él aparecen una serie de líneas sobre las que se pretende seguir realizando labores de investigación. Se comprueba así la validez de los resultados obtenidos y en qué medida podrán dar lugar, en un futuro, a otros prometedores resultados.

Así, se procede a presentar las principales conclusiones de este trabajo de tesis doctoral:

- Se ha propuesto una operación, que se ha denominado convolución jerárquica, para evaluar la convolución discreta de funciones utilizando estructuras jerárquicas de datos. La utilización de esta operación permite realizar evaluaciones parciales con resoluciones altas sólo en determinadas

regiones, lo que permite optimizar los consumos de recursos computacionales.

- Este método se ha fundamentado con una rigurosa base matemática lo que garantiza su viabilidad, y ha permitido llevarlo a la práctica con resultados cuya exactitud queda avalada por la base teórica desarrollada. Así, las definiciones realizadas de conjunto invariante junto con los teoremas que lo soportan constituyen la piedra angular de la convolución jerárquica y proporcionan la justificación de las operaciones que se llevan a cabo dentro del método propuesto.

- La convolución jerárquica se ha aplicado para evaluar la representación de los obstáculos en C-espacio para un robot móvil sin giros. Esto es posible debido a que este tipo de robots se pueden representar mediante funciones que cumplen las condiciones que aparecen impuestas en los teoremas enunciados y que permiten el uso de la convolución jerárquica. Su aplicación ha permitido obtener el C-espacio con una alta resolución en espacios de trabajos de grandes dimensiones. Este resultado tiene gran importancia práctica dado que los métodos existentes, que utilizan discretizaciones homogéneas, no eran capaces de obtener el C-espacio para entornos reales por los grandes requisitos de memoria para realizar las operaciones de convolución.

- Para poder llevar a cabo la evaluación del C-espacio con la convolución jerárquica de forma óptima se ha necesitado particionar o segmentar las regiones de interés durante el proceso de evaluación, de tal modo que se obtengan regiones compactas y de características uniformes. Para ello, se han analizado los requisitos que el proceso de segmentación debe cumplir de modo que éste no afecte al método fundamental de la convolución. Así, se ha mostrado que en cada instante el procedimiento para realizar la segmentación se puede parametrizar mediante dos factores: el porcentaje de región innecesaria que la desencadena y el número de semillas utilizadas.

- La aplicación de las definiciones y teoremas planteados en el método de convolución jerárquica y la propuesta de una técnica de segmentación han

permitido desarrollar e implementar algoritmos para la evaluación de C-espacios. Así, se ha realizado una importante labor experimental con un amplio conjunto de espacios de trabajo. Para analizar de forma exhaustiva el comportamiento de estos algoritmos con respecto a los parámetros que determinan el proceso de segmentación. Asimismo, se ha encontrado, aún en el caso de utilizar valores óptimos de dichos parámetros, una dependencia de los recursos computaciones (tiempo y memoria) con respecto al tamaño de la frontera de los obstáculos. Así, se comprueba la validez del algoritmo ya que el resultado está de acuerdo con los resultados teóricos establecidos por otros autores en la bibliografía.

- A la vista de los resultados experimentales del comportamiento de los algoritmos desarrollados, se ha propuesto un procedimiento completo para la evaluación del C-espacio, que incluye una red neuronal artificial como elemento predictivo de los valores óptimos de los parámetros de la segmentación. Así, se ha mostrado que los conjuntos de aprendizaje utilizados tienen las características adecuadas para conseguir realizar la tarea de aprendizaje propuesta. Además, esto requiere definir las entradas de dicha red. Con ello se ha conseguido tener como resultado del aprendizaje un elevado porcentaje de aciertos, por lo que se tiene como principal conclusión, la calidad y viabilidad del procedimiento propuesto para utilizar la convolución jerárquica en la evaluación práctica del C-espacio de un robot.

- También, se ha utilizado la convolución jerárquica para evaluar el C-espacio de una plataforma móvil con giros. La aparición de una variable de configuración que no convoluciona ha hecho necesario adaptar la convolución jerárquica para que incluya este nuevo tipo de variables, siempre siguiendo con el criterio de intentar reducir el consumo de memoria. Así, se ha propuesto una nueva estructura jerárquica de datos, el árbol multigrado-2^k, para la representación del su C-espacio. Ésta resuelve los problemas que surgieron para este tipo de estructura robótica, aunque es seguro que se podrá utilizar en otras aplicaciones donde esté presente la operación de convolución.

A partir de los resultados de este trabajo de investigación, se pueden plantear algunos puntos en los que es posible extender el trabajo presentado y que dan una idea de la importancia de esta línea de investigación para el futuro:

- Teniendo en cuenta la gran influencia que presenta el proceso de segmentación, se propone estudiar sus características y proceder a su formalización. Así, se espera obtener diferentes resultados teóricos y prácticos que lleven a un procedimiento más optimizado de la evaluación del C-espacio derivado de una mejor segmentación.

- Se propone introducir la operación de convolución jerárquica en la evaluación de C-espacios de manipuladores. Su aplicación será especialmente interesante en determinadas situaciones, como pueden ser los robots redundantes, en las que las limitaciones de recursos computacionales puedan comprometer la evaluación de esta representación con criterios restrictivos de exactitud.

- Se plantea aplicar los métodos de planificación existente sobre los C-espacios representados mediante estructuras jerárquicas. Se propone llevar a cabo la planificación, siempre que sea posible, a medida que se vaya obteniendo la información de zonas libres. Así, no sería necesario, por tanto, evaluar el C-espacio final, con alta resolución, para realizar la labor de planificación de caminos libres de colisiones.

Bibliografía

[AB88] F. Avnaim y J. D. Boissonnat. Polygon placement under translation and rotation. Informe Técnico 889, INRIA, Sophia-Antipolis, France, 1988.

[AIB] Aibo. http://www.aibo-europe.com/.

[AJ83] D.J. Abel y Smith J.L. A data structure and algorithm based on a linear key for a rectangle retrieval problem. *Computer Vision, Graphics, and Image Processing*, 24(1):1–13, Octubre 1983.

[BDFC98] W. Burgard, A. Derr, D. Fox, y A.B. Cremers. Integrating global position estimation and position tracking for mobilerobots: the Dynamic Markov Localization approach. En *Proc. of the IEEE/RSJ InternationalConference on Intelligent Robots and Systems*. 1998.

[BL83] R. A. Brooks y T. LozanoPérez. A subdivision algorithm in configuration space for findpath with rotation. En *Proceedings of the 8th International Conference on Artificial Intelligence*. 799-806, 1983.

[BM70] R. Bayer y E. McCreight. Organization and maintenance of large ordered indices. En *1970 ACM-SIGFIDET Workshop on Data Description and Access*, páginas 107–141. Noviembre 1970.

[Bro89] R. C. Brost. Computing metric and topological propierties of configuration-obstacles. En *Proceedings of the IEEE Conf. on Robotics and Automation*, páginas 170–176. 1989.

[BT94] Shawn Bohn y Erin Thornton. Environment reconstruction for robot

navigation. En *Proceeding of the SPIE*, tomo 2217, páginas 96–106. Abril 1994.

[Con84] C. I. Connolly. Cumulative generation of octree models from range data. En *Intl. Conf. Robotics*, páginas 25–32. Marzo 1984.

[Cra89] J. J. Craig. *Introduction to Robotics mechanics and control*. Addison Wesley Publising Co., Reading, MA, 1989.

[CSA88] C.H. Chien, Y.B. Sim, y J.K. Aggarwal. Generation of volume/surface octree from range data. En *The Computer Society Conference on Computer Vision and Pattern Recognition*, páginas 254–260. Junio 1988.

[CSU95] D.Z. Chen, R.J. Szczerba, y J.J. Uhran. Using framed-octrees to find conditional shortest paths in an unknown 3-d environment. Informe Técnico 95-9, Dep. of Computer Science and Engineering. University of Notre Dame, 1995.

[CSU97] D.Z. Chen, R.J. Szczerba, y J.J. Uhran. A framed-quadtree approach for determining euclidean shortest paths in a 2-d environment. *IEEE Trans. on Robotics and Automation*, 13(5):668–681, Octubre 1997.

[CT65] J.W. Cooley y J.W. Tukey. An algorithm for the machine computation of the complex fourier series. *Mathematics of Computation*, 19:297–301, Abril 1965.

[Cur98] B. Curto. *Formalismo Matemático para la Representación de Obstáculos en el Espacio de las Configuraciones de un Robot*. Tesis Doctoral, Dpto. de Informática y Automática. Universidad de Salamanca, Junio 1998.

[CVM98] B. Curto, P. Vega, y V. Moreno. Fast procedure to obstacle representation in the configuration space for mobile robots. En *Proc. Of Intelligent Components for Vehicles - ICV'98*, páginas 437–442. 1998.

[DCJ76] F. De Coulon y O Johnsen. Adaptative block schemes for source coding of blak-and-white facsimile. *Electornic Letters*, 12(3):61–62, Febrero 1976.

[DHS89] F. Dehne, A. L. Hassenklover, y J. R. Sack. Computing the configuration space for a robot on a mesh-of-processors. *Parallel Computing*, 12(2), 1989.

[DM90] J. R. Dooley y J. M. McCarthy. Parameterized descriptions of the joint space obstacles for a 5r closed chain robot. En *Proceedings of the IEEE International Conference on Robotics and Automation*, páginas 1542–1547. 1990.

[Don84] R. Donald. Motion planning with six degrees of freedom. Informe Técnico AI-TR-791, Artificial Intelligence Loboratory MIT, 1984.

[Eas70] C.M. Eastman. Representations for space planning. *Communications of the ACM*, 13(4):242, apr 1970.

[Fav84] B. Faverjon. Object level programming of industrial robots. En *Proceedings of the International Conference on Robotics*, páginas 504–512. Atlanta, Marzo 1984.

[Fav86] B. Faverjon. Object level programming of industrial robots. En *Proceedings of the IEEE International Conference on Robotics and Automation*, páginas 1406–1412,. 1986.

[FJ98] M. Frigo y S.G. Johnson. Fftw: An adaptive software architecture for the fft. En *ICASSP*, tomo 3, páginas 1381–1384. 1998.

[FS89] K. Fujimira y H. Samet. A hierarchical strategy for path planning among moving obstacles. *IEEE Trans. on Robotics and Automation*, 5(1):61–69, Febrero 1989.

[FT87] B. Faverjon y P. Tournassoud. A local based approach for path planning of manipulators with a high number of degrees of freedom. En *Proceedings of the IEEE International Conference on Robotics and Automation*, páginas 1152–1159. 1987.

[Gar82] I Gargantini. An effective way to represent quadtrees. *Communications of the ACM*, 25(12):905–910, Diciembre 1982.

[Gee95] C.V. Geem. Preparations with workspace information for meaningful robot motion planning in configuration space. Informe Técnico 95-14, RSIC-Linz Faculty, Marzo 1995.

[Gee96] C.V. Geem. *Fast Planning of a Good Path for a Manipulator Using Potential Fields on a Non-Uniform Grid in C-space*. Tesis Doctoral, Johannes Kepler Universität Linz, Mayo 1996.

[GM89] Q. J. Ge y J. M. McCarthy. Equations for boundaries of joint obstacles for planar robots. En *Proceedings of the IEEE International Conference on Robotics and Automation*, páginas 418–423. 1989.

[GM90] Q. J. Ge y J. M. McCarthy. An algebraic formulation of configuration-space obstacles for spatial robots. En *Proceedings of the IEEE International Conference on Robotics and Automation*, páginas 1542–1547. 1990.

[Gou84] L. Gouzènes. Strategies for solving collisions-free trajectories problems for mobile and manipulator robots. *International Journal of Robotics Research*, 3(4):51–65, 1984.

[GS86] L. Guibas y R. Seidel. Computing convolution by reciprocal search. En *Proceedings of the ACM Symposium on Computational Geometry*, páginas 90–99. 1986.

[Gut84] A. Guttman. R-trees: A dynamic index structure for spatial searching. En *ACM SIGMOD International Conference on Management of Data*, páginas 47–57. Junio 1984.

[GW87] R.C. Gonzalez y P. Wintz. *Digital Image Processing*. Addison-Wesley Publishing Company, 2^a edición, Noviembre 1987.

[HBM+02] T. Hong, S. Balakirsky, E. Messina, T. Chang, y M. Shneier. A hierarchical model for an autonomous scout vehicle. En *Proceedings of the SPIE*, tomo 4715. Abril 2002.

[Her86a] M. Herman. Fast path planning in unstructures, dynamic, 3-d worlds. *Proceedings of the SPIE*, 635:505–512, Abril 1986.

[Her86b] M. Herman. Fast, three-dimensional, collision-free motion planning. En *Proceedings of the IEEE Int. Conf. on Robotics and Automation*, páginas 1056–1063. Abril 1986.

[HS85] Tsai-Hong Hong y Michael O. Shneier. Describing a robot's workspace using a secuence of views from a moving camera. *IEEE Trans. on Patern Analisis and Machine Intelligence*, PAMI-7(6):721–726, Noviembre 1985.

[HS92] R.M. Haralick y L.G. Shapiro. *Computer and Robot Visión*, tomo I. Addison-Wesley Publishing Company, 1992.

[Hun78] G.M. Hunter. *Efficient computation and data structures for graphics*. Tesis Doctoral, Department of Engineering and Computer Science, Princeton University, 1978.

[Hwa90] Y. K. Hwang. Boundary equations of configurations obstacles for manipulators. En *Proceedings of the IEEE International Conference on Robotics and Automation*, páginas 298–303. 1990.

[JG97] D. Jung y K.K. Gupta. Octree-based hierarchical distance maps for collision detedtion. *Journal of Robotic Systems*, 14(11):789–806, Noviembre 1997.

[Kav95] L. E. Kavraki. Computation of configuration-space obstacles using the fast fourier transform. *IEEE Tr. on Robotics and Automation*, 11(3):408–413, 1995.

[KCN89] R. D. Klafter, T. A. Chmielewski, y M. Negin. *Robotic Enginering an integrated approach*. Prentice Hall, New Jersey, 1989.

[KE80] E. Kawaguchi y T Endo. On a method of binary picture representation an dits aplication to data compression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2(1):27–35, Enero 1980.

[Kha86] O. Khatib. Real-time obstacles avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5(1):90–98, 1986.

[KTKY95] Y. Kitamura, T. Tanaka, F. Kishino, y M. Yachida. 3-d path planning in a dinamic environment using an octree and an artificial potential field. En *Proceedings of the Int. Conf. on Intelligent Robots and Systems*, páginas 474–481. 1995.

[Lat91] J. C. Latombe. *Robot motion planning*. Kluwer Academic Publishers, Boston, MA, 1991.

[LC94] A. Li y G. Crebbin. Octree encoding of objects from range images. En *Pattern Recognition*, tomo 27(5), páginas 727–739. Mayo 1994.

[LGF86] C. S. Lee, R. C. Gonzalez, y K. S. Fu. *Tutorial on robotics*. IEEE Computer Society, Los Angeles, CA, 1986.

[Loz83] T. LozanoPérez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, 32:108–120, Febrero 1983.

[Loz87] T. LozanoPérez. A simple motion-planning algorithm for general robot manipulators. *IEEE Journal of Robotics and Automation*, 3(3):224–238, 1987.

[LP91] T. LozanoPérez y P.O'Donnell. Parallel robot motion planning. En *Proc. of the IEEE Int. Conf. on Robotics and Automation*, páginas 1000–1007. 1991.

[Mor66] G.M. Morton. *A computed oriented geodetic data base and a new technique in file sequencing*. IBM Ltd., Ottawa, Canada, 1966.

[Mor96] V. Moreno. *Planificación de trayectorias utilizando algoritmos de exploración de grafos en paralelo*. Tesis Doctoral, Universidad de Salamanca, 1996.

[NB91] W. Newman y M. Branicky. Real-time configuration space transforms for obstacle avoidance. *The International Journal of Robotics Research*, 6, Octubre 1991.

[Nil69] N.J. Nilson. A mobile automation: An application of artificial intelligence techniques. En *Proceedings of the Fith International Joint Conference on Artificial Intelligence*, páginas 509–520. Washintong D.C., 1969.

[RHW86] D. E. Rumelhart, G. E. Hinton, y R. J. Williams. Learning internal representations by error propagation. En D. E. Rumelhart, J. L. McClelland, et al., editores, *Parallel Distributed Processing: Volume 1: Foundations*, páginas 318–362. MIT Press, Cambridge, 1986.

[RI02] J. Rosell y P. Ifiiguez. A hierarchical and dynamic method to compute harmonic functions for constrained motion planning. En *Proceedings of the 2002 IEEE/RSJ*, páginas 2335–2340. Octubre 2002.

[RN95] S. Russell y P. Norving. *Artificial Intelligence. A modern approach.* Prentice-Hall International Editions, 1995.

[RSSW82] A. Rosenfeld, H. Samet, C. Shaffer, y R.E. Webber. Application of hierarchical data structures to geographical information sistems. Informe Técnico Computer Science TR-1197, University of Maryland, Junio 1982.

[Sam90] H. Samet. *The Design and Analisys of Spatial Data Structures.* Addison - Wesley, apr 1990.

[SH92] C.A. Shaffer y G.M. Herb. A real-time robot arm collision avoidance system. *IEEE Trans. on Robotics and Automation*, 8(2):149–160, Abril 1992.

[Sha87] M. Sharir. Efficient algorithms for planning purely translational collision-free motion in two an three dimensions. En *Proceedings of the IEEE International Conference on Robotics and Automation*, páginas 1326–1331. 1987.

[SJL94] L. K. Swift, T. Johnson, y P. E. Livadas. Parallel creation of linear octrees from quadtree slices. *Parallel Processing Letters*, 4(4):447–453, 1994.

[SRF87] T. K. Sellis, N. Roussopoulos, y C. Faloutsos. The r+-tree: A dynamic index for multi-dimensional objects. En *The Very Large Data Bases Journal*, páginas 507–518. Septiembre 1987.

[ST85] H. Samet y M. Tamminen. Bintrees, csg trees, and time. *Computer Graphics*, 19(3):121–130, Julio 1985.

[SV89] M. W. Spong y M. Vidyasagar. *Robot dynamics and control*. John Wiley and Sons, New York, 1989.

[Tan76] S. Tanimoto. Pictorial feature sistorsion in a pyramid. *Computer Graphics and Inage Processing*, 5(3):333–352, Septiembre 1976.

[The02] R. Therón. *Cálculo Paralelo del Espacio de las Configuraciones para Robots Redundantes*. Tesis Doctoral, Dpto. de Informática y Automática. Universidad de Salamanca, Junio 2002.

[Uhr72] L. Uhr. Layered "recognition cone" networks that preprocess, classify, and describe. *IEEE Transactions on Computers*, 21(7):758–768, jul 1972.

[VO95] J. Vleugles y M. Overmars. Approximating generalized voronoi diagrams in any dimension. Informe Técnico UU-CS-1995-14, Dep. of Computer Science. Utrecht University, Mayo 1995.

[YG00] Y. Yu y K.K. Gupta. Sensor-based probabilistic roadmaps: Experiments with an eye-in-hand system. *Journal of Advanced Robotics*, 14(6):515–536, Diciembre 2000.

[YSSB98] A. Yahja, A. Stentz, S. Singh, y B.L. Brumitt. Framed-quadtree path planning for mobile robots operating in sparse environments. En *IEEE Conf. on Robotics and Automation*. Mayo 1998.

[Zel92] A. Zelensky. A mobile robot exploration algorithm. *IEEE Trans. on robotics and automation*, 8(6):707–717, Diciembre 1992.