CÁLCULO PARALELO DEL ESPACIO DE LAS CONFIGURACIONES PARA ROBOTS REDUNDANTES

Cálculo Paralelo del Espacio de las Configuraciones para Robots Redundantes

Roberto Therón Sánchez

2002
Universidad de Salamanca
Facultad de Ciencias
Departamento de Informática y Automática

Departamento de Informática y Automática

Universidad de Salamanca

Impreso en el Departamento de Informática y Automática de la Universidad de Salamanca.

Primera edición: 1 de junio de 2002

Los abajo firmantes **CERTIFICAN** que, bajo su dirección, D. **Roberto Therón Sánchez** ha realizado el trabajo de investigación que se recoge en esta memoria, titulada ***Cálculo Paralelo del Espacio de las Configuraciones para Robots Redundantes***, que se presenta para optar al grado de Doctor por la Universidad de Salamanca.

Salamanca, 27 de mayo de 2002

D. **Vidal Moreno Rodilla**

Prof. Tit. de Universidad
Área de Ing. de Sistemas
Universidad de Salamanca

D.ª **Belén Curto Diego**

Pfa. Tit. Esc. Universitaria
Área de Ing. de Sistemas
Universidad de Salamanca

A María

Agradecimientos

Quiero expresar mi agradecimiento a todos los que me han sufrido durante el desarrollo de esta tesis.

En primer lugar, a mis directores, Vidal y Belén: sin ellos seguiría pensando que los robots articulados son C3PO y los móviles, R2D2. A Luis, a quién agradezco todos los comentarios y consejos, que no han sido pocos. A Javi, porque seguimos una trayectoria paralela y es el que mejor conoce nuestro día a día. Y, por supuesto, a todos los demás en el departamento.

No puedo desaprovechar estas líneas y no darles las gracias a mi familia, principalmente porque son muchas las ocasiones en que debería haberlo hecho, pero, desgraciadamente, en contra de como debería ser, reconocer todo lo que les debo no es algo cotidiano. En especial a mis padres, porque sé que esto supone un orgullo para ellos.

Tampoco quiero olvidar a mis amigos, a los que me dedicaré más a partir de ahora; a Fabio le debo las horas que ha dedicado a ayudarme con las animaciones. Siempre nos quedará Sapristi.

Y, finalmente, quiero recordar con todo el cariño a la Célula, porque me ha ayudado en todo lo que puede, que es, con mucho, más de lo que me merezco y porque se preocupa como nadie por mí.

Ah, y se me olvidaba, al teclado y al ratón, con los que he estado jugando.

Todos para uno, uno para todos

Los Tres Mosqueteros
ALEJANDRO DUMAS

1. Un robot no puede dañar a un ser humano, o por inacción, permitir que un ser humano resulte dañado;
2. Un robot debe obedecer las órdenes dadas por los seres humanos, excepto cuando tales órdenes entren en conflicto con la primera ley;
3. Un robot debe proteger su propia existencia, hasta donde esta protección no entre en conflicto con la primera o la segunda ley.

Círculo vicioso
ISAAC ASIMOV

robot. A reprogrammable, multifunctional manipulator designed to move material, parts, tools, or specialized devices through various programmed motions for the performance of a variety of tasks.

Robot Institute of America, 1979.

parallelization.(*n.*) Turning a serial computation into a parallel one. Also sometimes turning a vector computation into a parallel one. This may be done automatically by a parallelising compiler or (more usually) by rewriting (parts of) the program so that it uses some parallel paradigm.

A Glossary of Parallel Computing Terminology,
IEEE Parallel & Distributed Technology,
February 1993

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Robótica y Paralelismo: Un Matrimonio Perfecto

Calvin: Antes no me gustaba escribir redacciones, pero ahora disfruto.
He comprendido que se escribe con al intención de inflar ideas flojas, oscurecer un razonamiento pobre e inhibir la claridad.
¡Con un poco de práctica, puede convertirse en una niebla intimidante e impenetrable!

El Último Libro de Calvin y Hobbes
BILL WATTERSON

En tiempos pasados el grupo de usuarios de ordenadores paralelos estaba limitado a aquéllos que trabajaban para organizaciones con enormes presupuestos, capaces de invertir grandes cantidades de dinero para comprar ordenadores con esta especial arquitectura.

En la actualidad vivimos una situación en la que el cálculo paralelo ya no se ve como una rara subárea de la informática; de hecho, los ordenadores con más de un procesador son bastante populares y las predicciones dicen que este tipo de máquinas va a dominar el mercado de los ordenadores personales en un futuro próximo. A este panorama de computadores con arquitectura paralela hay que sumar una tendencia cada vez más extendida: el uso de un grupo de estaciones de trabajo conectadas en red para realizar trabajos colaborativos y concurrentes.

Mientras somos testigos de los pasos de gigante que se dan en el mundo de la computación paralela, todavía estamos muy lejos del mundo soñado por Asimov en 1942, cuando en el número de marzo de la revista *Astounding Science Fiction* apareció el relato corto *Círculo vicioso (Roundaround)*, en el que se establecían las Tres Leyes de la Robótica[1]. Sin embargo, también se han producido grandes avances en este campo y se puede decir que la sociedad está cada día más habituada a tratar con robots en la vida cotidiana —aunque estos no sean humanoides como en la literatura o en el cine[2].

1.1 Robots Redundantes

En una primera gran división se puede hablar de dos tipos de robots: móviles y articulados. El número de movimientos espaciales independientes entre sí —cada uno de ellos denominado **Grado de libertad**(*Degree of freedom*, **DOF**)— determinará la versatilidad del robot. Por ejemplo, un coche se mueve en un espacio 2D con tres grados de libertad: posición (x, y) y orientación. Similarmente, se necesitan 6 variables para definir la posición y orientación de la pinza de brazo manipulador (x, y, z, balanceo (*roll*), cabeceo (*pitch*) y guiñada (*yaw*)). Aparece el concepto de **Redundancia** cuando se tiene un número ma-

[1] Fue el propio Isaac Asimov (1920?–1992) el que acuñó la palabra *robótica*, como ciencia que estudia los sistemas que permiten la construcción de robots, y el que, a través de su extensa producción escrita, impulso su avance (Engelberger, creador del primer robot industrial, Unimate, en 1958, confesó que su fascinación por los robots era consecuncia de la lectura de *Yo, Robot* (*I, Robot*, 1968) —colección de relatos cortos que incluye *Círculo vicioso*— en su infancia [AWG83]. Sin embargo, Karel Capek (1890–1938) debe su reputación, por encima de todo, a haber inventado la palabra *robot* para su obra teatral R.U.R. (*Rossum's Universal Robots*, 1920), aunque varios biografos coinciden en que el verdadero creador de la palabra fue su hermano Josef, también escritor, que la propuso a petición de Karel cuando éste no lograba encontrar un nombre para los ingenios de vida artificial —ilustrando el título de este capítulo aparecen Sulla y Primus, en la primera edición del guión de R.U.R. (Praga, 1920)— que constituyen el eje central del texto.

[2] Existe una amplia caterva de robots procedentes de la literatura, el cine o la televisión, algunos de los cuales se han convertido en iconos generacionales como María, la robot de *Metropolis* (Friz Lang, 1927), R2D2 y C3PO, de la exitosa serie de películas *La Guerra de las Galaxias* (*Star Wars*, George Lucas) o Bender, el robot de la serie de dibujos animados *Futurama* (del creador de *Los Simpson*, Matt Groening).

yor de grados de libertad que el número de variables necesarias para definir un cuerpo en el espacio de trabajo.

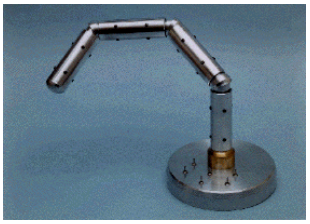


Figura 1.1: Robot redundante tipo serpiente

Es indiscutible la potencia que se añade con cada grado de libertad redundante. Así, en el caso del coche se estarí introduciendo redundancia si se permitiera que éste se desplazara lateralmente, además de girar, moverse hacia delante y moverse hacia atrás, facilitando enormemente el aparcamiento, por ejemplo; en el caso de un robot manipulador, basta pensar en un brazo humano, con sus 7 grados de libertad (3 en el hombro, 1 en el codo y 3 en la muñeca), capaz de adaptarse a distintas condiciones utilizando variadas configuraciones que aprovechan la fuerza de diferentes músculos: acercándolo al cuerpo cuando carga un peso o doblándolo hacia arriba por el codo y apoyando la palma de la mano cuando empuja un objeto.

Figura 1.2: Brazo articulado sobre plataforma

Son muchas las situaciones en las que la mejor solución es adoptar una estructura robótica que incorpore redundancia; estos posibles robots redundantes van desde el caso simple de un

brazo planar de tres articulaciones hasta complejas combinaciones de manipuladores montados sobre móviles (Figura1.2), pasando por robots hiper-redundantes [LOH94], [KP99] con morfologías semejantes a serpientes (Figura 1.1) o incluso ASIMO[3], con sus 26 grados de libertad (Figura 1.3) que Honda presentó públicamente el 12 de noviembre de 2001 como el primer robot humanoide disponible para ser alquilado[4].

Figura 1.3: Robot humanoide redundante con 26 grados de libertad

Entre las optimizaciones que añaden las configuraciones redundantes destacan[5]:

- **Mecánicas y cinemáticas:**
  - maximizar la transmisibilidad de fuerza (elegir la solución que mejor transmita la fuerza al efector final);

[3] Es claro el homenaje que la compañía japonesa quiere dedicar al padre literario de la Robótica.

[4] http://www.world.honda.com/news/2001/c011112.html (revisado en junio de 2002).

[5] http://www.robotics.utexas.edu (revisado en junio de 2002).

  - maximizar la disponibilidad de rango de articulación (manteniendo cada articulación lo más cerca posible de su posición central);
  - minimizar la velocidad de las articulaciones (eligiendo la solución que requiera la menor cantidad de movimiento);
  - maximizar la destreza (eligiendo la solución que evite singularidades y maximice la destreza del efector final);
  - minimizar la energía (eligiendo soluciones que minimicen la velocidad y la inercia);
  - maximizar la rigidez (eligiendo soluciones que minimicen la desviación).

- **Sortear obstáculos:** Se puede utilizar la redundancia para evitar obstáculos que de otra manera impedirían al efector final llegar a su destino: la configuración del manipulador puede envolver al obstáculo.

- **Tolerancia a fallos:** En el caso de un robot tradicional con 6 grados de libertad, cuando uno de estos falla, la habilidad para completar una tarea se ve comprometida; un robot redundante puede reorganizarse para compensar la pérdida de un grado de libertad mecánico [RM96] (Figura 1.4).

Redundancia y Tolerancia a Fallos

El robot planar de la figura 1.4 puede llegar a cualquier punto objetivo a pesar de que se produzca un fallo mecánico en cualquiera de sus tres articulaciones.

Lamentablemente, no todo son ventajas, ya que a mayor número de grados de libertad mayor es la complejidad y el precio se dispara (se puede doblar el precio de un robot al pasar de 4 a 6 grados de libertad [ROS98]).

Por ello se han desarrollado numerosas soluciones matemáticas que tratan de resolver algunos de los complejos problemas cinemáticos que incorporan los robots redundantes ([HT95] o [KT96]).

1.2 El Espacio de las Configuraciones

A diferencia de los seres humanos, para los que realizar cualquier actividad supone poner en marcha un engranaje que fun-

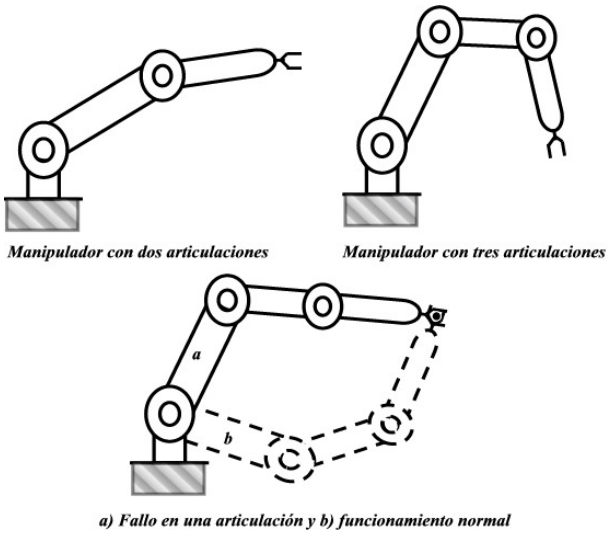


Figura 1.4: Tolerancia a fallos mecánicos en un robot redundante

ciona, principalmente, de forma inconsciente, los robots deben incorporar entre sus capacidades razonamiento automatizado, métodos de percepción del entorno y mecanismos de control de sus acciones [LGF86].

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.

En esta compleja estructura conviven numerosas técnicas procedentes de diferentes disciplinas: para que un robot pueda interaccionar de forma autónoma con un entorno de trabajo cambiante, es necesario dotarle de una capacidad sensorial que permita una descripción precisa de aquellos puntos del espacio que están ocupados por obstáculos; por otro lado, para que un robot se mueva de forma segura dentro de estas zonas parcialmente ocupadas por objetos en movimiento, se debe conocer, en cualquier instante de tiempo, el conjunto de parámetros que definen la posición y orientación de todos los puntos del robot

con respecto a un sistema de referencia fijo.

Geometría del robot

Dependiendo de su geometría, el robot podrá adoptar una serie de configuraciones, denominándose **espacio de las configuraciones** (**C-espacio**) del robot al conjunto de todas sus posibles configuraciones. Este concepto, que es esencialmente una herramienta de representación, fue introducido en la Robótica por Lozano-Pérez, y ocupará un papel fundamental en la descripción de este trabajo de investigación.

Algunos robots se pueden mover libremente dentro de un espacio de trabajo; otros tienen un punto de anclaje fijo. Asimismo, existen robots formados por un único elemento rígido, mientras que otros cuentan con una serie de elementos, rígidos o no, unidos por articulaciones.

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 la orientación del robot en su espacio de trabajo queda completamente determinada si se especifican los valores para los grados de libertad del robot.

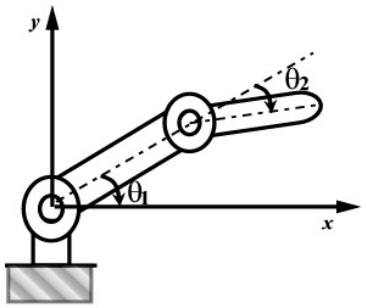


Figura 1.5: Grados de libertad de un robot articulado con dos elementos

En la figura 1.5 aparece un robot articulado en el plano, cuya cadena cinemática está formado dos elementos conectados mediante una articulación de revolución. Además, el primer elemento puede girar respecto a un eje perpendicular al plano. Una configuración arbitraria del robot viene especificada dando el valor del ángulo que forma el primer elemento con el eje x y el valor del ángulo que forma el segundo elemento respecto del primero. Por tanto, una configuración viene dada por la tupla bidimensional (θ_1, θ_2). En este caso el espacio de las configuraciones está formado por todas las tuplas (θ_1, θ_2) con $\theta_1, \theta_2 \in [-\pi, \pi)$.

Es fácil comprender la necesidad de métodos simples para la evalución del espacio de las configuraciones, pues, al contario que con el espacio de trabajo, cuya dimensión permanece constante, al aumentar el número de grados de libertad aumenta la dimensión del C-espacio. Como ejemplo de lo intratable que puede llegar a ser el problema, se considera [CGK93] un brazo robótico con k articulaciones, con cada grado de libertad discretizado en n niveles. Tal robot tendría un C-espacio con n^k configuraciones diferentes. Si tomamos $n = 90$, un brazo con cuatro articulaciones ($k = 4$) tendría más de sesentaicinco millones[6] de configuraciones, mientras que un brazo con seis articulaciones ($k = 6$) llegaría a superar los quinientos mil millones[7] de configuraciones.

1.2.1 Planificación de Movimientos

Planificación de Caminos

A diferencia de la planificación de movimientos, no se tiene en cuenta qué es lo que hay que hacer para lograr el movimiento, sólo busca un camino posible.

Un elemento crucial en la construcción de robots autónomos es el Planificador de Movimientos. Éste trata de encontrar un movimiento continuo que lleve al robot desde una configuración inicial a otra final, sujeto a la restricción de que en ningún momento de dicho movimiento se produzca colisión alguna con obstáculos en el espacio de trabajo o devolver un fallo, si no existe un camino que enlace ambos puntos.

Se han desarrollado muchos algoritmos de planificación de movimientos, que son de gran interés teórico, pero que no se utilizan en la práctica debido a su coste computacional [HA92]. Por otro lado, Un punto de vista incorrecto y muy extendido es el que considera que la planificación de movimientos consiste esencialmente en detectar colisiones. Además, se ocupa de calcular caminos libres de colisiones entre obstáculos móviles, de coordinar el movimiento de varios robots, de planificar movimientos para empujar y deslizar objetos con el fin de lograr relaciones exactas entre estos, de planificar la manera de coger los objetos de forma estable, etc.

1.2.2 Control de Movimientos

Una vez generada la secuencia de configuraciones por las que tiene que pasar un robot, desde la inicial a la final, un con-

[6] 65610000 exactamente.
[7] 531441000000 exactamente.

trolador tiene que realizar las operaciones oportunas para que el robot siga este camino. En definitiva, a partir de la secuencia generada por el planificador de movimientos, el controlador tiene que encontrar la fucnión temporal que define los pares a aplicar a los actuadores del robot en cada instante de tiempo y aplicarlos [SV89].

Este proceso se realiza en dos fases, generación de trayectorias, que define el perfil de velocidad a lo largo del camino, y seguimiento de la trayectoria, que calcula los pares que se deben aplicar a los actuadores en cada momento para poder realizar el movimiento requerido.

1.2.3 ¿Por qué el Espacio de las Configuraciones?

En esta sección se van a analizar las principales ventajas que se obtienen al trabajar en el espacio de las configuraciones en lugar de en el espacio de trabajo.

Como ya se ha comentado anteriormente, la característica fundamental que hace tan atractivo al espacio de las configuraciones, es la simplificación del problema que conlleva poder reducir la decripción del robot a un punto. Así, como consecuencia, algunas de las tareas que se benefician son la planificación de movimientos y el control de los mismos y la generación de trayectorias. En la otra cara de la moneda, aparece la dificultad añadida de evaluar cuál es el conjunto de configuraciones prohibidas (C-obstáculos), es decir, las configuraciones que producirían colisiones, ya sean entre partes del robot y obstáculos en el espacio de trabajo o entre los propios elementos del robot (autocolisiones), además de aquellas que violan alguna restricción mecánica.

A continuación se comparan cada una de estas tareas cuando se realizan a partir del espacio de las configuraciones o a partir del espacio de trabajo, comprobándose la mejor elección del primero.

Si se trata de realizar la planificación de movimientos dentro del espacio de trabajo será necesario planificar los movimientos de todos y cada uno de los puntos que forman parte de los distintos elementos del robot. Sin embargo, trabajando en el espacio de las configuraciones, se reduce drásti-

camente el problema, ya que cada una de las distintas configuraciones intermedias por las que tiene que pasar el robot desde su configuración origen a su configuración destino se corresponde con un punto en el C-espacio, por lo que la planificación a realizar es la del movimiento de un punto. Ejemplos de planificadores en el espacio de trabajo son [Nil69][HWW98][KK91], mientras que ejemplos del segundo caso son [LPW79][KSLO94][Bag96][BK00].

Una ventaja adicional viene dada por el hecho de que comprobar si existe o no colisión entre el robot y los obstáculos, o si se producen autocolisiones consiste simplemente en verificar si el punto correspondiente al robot en una configuración pertenece o no a los obstáculos representados en dicho espacio; De forma general, el punto que representa a una configuración conflictiva pertenece al C-obstáculo, siendo los demás puntos configuraciones libres de colisión.

En cuanto a las trayectorias, se puede decir que el hecho de conocer previamente las configuraciones que producen colisión o no con los obstáculos, supone un gran avance para facilitar el desarrollo de algoritmos generadores —se puede generar el perfil de velocidad teniendo en cuenta la distancia entre la configuración planificada y la que produce colisión, entre otros criterios—, mientras que realizar el seguimiento de trayectorias tambien recibe un beneficio por parte del espacio de las configuraciones, ya que las variables que definen una configuración constituyen las referencias para el controlador, simplificando el diseño de los algoritmos. Además, al tener las restricciones que los obstáculos imponen en las variables de control, se favorecen dichos algoritmos.

Finalmente, disponer explícitamente de una representación de los obstáculos en el espacio de las configuraciones permite abordar el caso de la planificación dinámica con mayores garantías de éxito, permitiendo no sólo considerar las restricciones en el control debidas a la naturaleza de los actuadores, sino tener también en cuenta las restricciones propiciadas por la presencia de obstáculos en el espacio de trabajo. Ciertamente, disponer de una representación explícita de los obstáculos en el espacio de las configuraciones no resuelve el problema, pero ayuda de forma sensible a hacerlo más abordable.

1.3 Cálculo Paralelo

Un computador paralelo es un conjunto de procesadores que son capaces de trabajar cooperativamente para resolver un problema computacional. Esta definición es lo suficientemente amplia como para englobar computadores paralelos que tienen cientos de procesadores, redes de estaciones de trabajo, estaciones de trabajo multiprocesadoras y sistemas empotrados.

Una de las ventaja que aportan los ordenadores paralelos es que ofrecen el potencial de concentrar gran cantidad de recursos —ya sea el número de procesadores, la cantidad de memoria o el ancho de banda de E/S— para importantes problemas computacionales.

Sin embargo, los programas concurrentes son inherentemente más complejos que los programas secuenciales; se podría decir que son a los programas secuenciales lo que el ajedrez es a las damas. Así, se puede considerar que los programas paralelos son programas secuenciales extendidos con mecanismos para especificar la concurrencia, las comunicaciones y la sincronización, con todas las complicaciones que esto conlleva para lograr diseños adecuados.

Además, los algoritmos paralelos que se diseñen deberán explotar tanto el paralelismo intrínseco del problema como la arquitectura del ordenador paralelo sobre el que se van a implementar.

El paralelismo tradicionalmente se ha dedicado a resolver problemas de clasificación, de búsqueda y de cálculo numérico. Generalmente, lo que se requiere es una computación tremendamente rápida y que se procesen enormes cantidades de datos en poco tiempo. En otros casos el tiempo de ejecución no es un parámetro que se desee optimizar, siendo, en estos casos, la capacidad de resolver problemas hasta el momento inabordables la aportación de la programación paralela.

El área de la robótica ha demostrado ser un excelente campo de cultivo donde sembrar la semilla paralela.

1.4 Estado del Arte

En esta sección se va a hacer un repaso de los trabajos más destacados, entre los que hacen uso de la tecnología disponible

para realizar cálculos en paralelo, que han propiciado algunos avances en la evaluación del espacio de las configuraciones.

En el campo de la Robótica hay muchas facetas susceptibles de paralelización, ya que es una rama en la que típicamente se deben manejar grandes cantidades de datos para obtener resultados intermedios que a su vez sirven de entrada para nuevos cálculos. Además, para que estos cálculos sean útiles, tienen que estar disponibles lo antes posible, ya que un robot es un sistema que, lógicamente, está fuertemente restringido en tiempo real, pues de su respuesta a determinados estímulos depende su buen funcionamiento.

Tradicionalmente, el campo dentro de la Robótica donde mayor ha sido la incursión del uso de técnicas paralelas es el que se dedica a realizar labores de planificación —y algunos de sus algoritmos auxiliares—, lo que, además, está íntimamente relacionado con el objetivo de este trabajo de investigación.

Cálculos previos

En la mayoría de los métodos de planificación de caminos existentes son necesarios algoritmos auxiliares que se encarguen de realizar algún preprocesamiento, como es la evaluación del C-espacio.

En [Hen97] se realiza una revisión detallada de los distintos enfoques paralelos utilizados en la planificación de movimientos. [LPP91][CGK93][Bag96][AG97] son ejemplos de trabajos que realizan planificación en paralelo y que trabajan sobre el C-espacio.

El prerrequisito principal para todos los algoritmos de planificación de movimientos que utilicen una representación específica del espacio de las configuraciones es la transformación de los obstáculos en dicho espacio. A continuación se comentan los trabajos más importantes que realizan esta tarea en paralelo.

En [DHS89], se presenta un algoritmo sistólico para el cálculo del C-espacio para un conjunto de obstáculos en el plano para un robot rectilíneamente convexo. Se asume que los obstáculos y el robot están representados mediante una imagen binaria $N \times N$. El algoritmo está diseñado para una red de tipo malla de $N \times N$ procesadores (utilizando la representación canónica de una imagen en una matriz de procesadores) y tiene un tiempo de ejecución asociado de $O(N)$, lo que es asintóticamente óptimo.

Un problema similar para robots de formas arbitrarias con intersección de ejes de unión se trata en [LPP91]. De esta forma se reconoce que se puede calcular una familia de mapas primitivos, que pueden ser combinados mediante superposición

basada en la distribución de los objetos reales. El algoritmo funciona sobre una Connection Machine de 8192 procesadores.

En [GJ91], los obstáculos del C-espacio para un robot de n DOF's se aproximan mediante conjuntos de rebanadas (*slices*) de dimensión $n-1$, construidas recursivamente a partir de rebanadas 1D. Las tareas para calcular las proyecciones de dichas rebanadas tienen una dependencia en forma de árbol y se planifican en una topología (triple) de matriz lineal, de estrella y de árbol. Se consigue un *speedup* cercano a 11 utilizando 12 Transputers para un manipulador de dos eslabones.

En [JL95], utilizando una superclase de los ordenadores tipo malla, un hipercubo $N \times N$, se consigue realizar un algoritmo similar al de [DHS89] para robot con forma de polígono convexo para imágenes $N \times N$ en un tiempo $O(logN)$

Finalmente, en [The00][TBCM01][TBC+02], se identifican diferentes niveles de paralelismo presentes en el formalismo matemático para la evaluación del espacio de las configuraciones de robots móviles y articulados de [Cur98]; Además, se proponen soluciones paralelas que, utilizando una técnica de *slicing*, explotan uno o varios de estos niveles. Por otro lado, se presentan excelentes resultados para implementaciones, en entornos distribuidos, de algoritmos para robots móviles.

Estos últimos trabajos son de especial relevancia para la presente investigación, pues sientan las bases de la problemática relacionada con el cálculo paralelo del C-espacio mediante productos de convolución.

1.5 Objetivos de la Investigación

El uso del espacio de las configuraciones para simplificar la planificación de movimientos es un concepto presente en la Robótica desde 1983, año en que lo popularizó Lozano Pérez [LP83].

Un paso especialmente relevante se da en [Kav93], donde se aproxima el problema a partir del hecho observado de que cuando un robot es un objeto rígido que sólo se puede mover de forma translacional, el espacio de las configuraciones es la convolución del espacio de trabajo y del robot. Se hace uso del algoritmo para la Transformada Rápida de Fourier (Fast Fourier Transform, FFT) para calcular esta convolución. El

método es particularmente prometedor para espacios de trabajo con muchos y/o complicados obstáculos, o cuando la forma del robot no es simple. Este método se puede beneficiar del hardware y la experiencia existentes para el cálculo de la FFT.

En 1998, Curto [Cur98] presenta un formalismo matemático que permite representar los obstáculos en el espacio de las configuraciones para un mayor número de tipos de robot, es decir, no se limita al caso de los móviles, sino que generaliza el método para que sea aplicable al caso de los robots articulados.

Es en este punto donde surge la justificación de este trabajo: superar las limitaciones de los métodos existentes y permitir contemplar cualquier tipo de robots, incluídos los redundantes, para los que ninguno de los métodos comentados es válido. Así, en [TMCB02] se comparan, para un robot planar de dos articulaciones, el método propuesto en [Cur98] con un nuevo método que, como se verá en sucesivos capítulos, supone la piedra angular de la generalización a cualquier tipo de robots.

Por tanto, el objetivo principal de este trabajo es proponer un nuevo método general que permita obtener una representación de los obstáculos en el espacio de las configuraciones de cualquier estructura robótica.

Además, el método debe estar bien fundamentado, de forma teórica, por un formalismo sólido y de carácter general.

El nuevo método debe intentar explotar la independencia de los distintos elementos que constituyen la estructura robótica, de forma que se pueda evaluar las configuraciones que produzcan colisión, asociadas a cada uno de ellos. Dada su vocación general, se procurará en lo posible utilizar procedimientos bien conocidos por la comunidad científica para realizar las tareas auxiliares necesarias.

Por otro lado, como ya se ha explicado en secciones anteriores, la evaluación del C-espacio supone un proceso muy costoso dado el volumen de información que se maneja, lo que es especialmente problemático en el caso de los robots redundantes. Por tanto, se analizará el problema teniendo en cuenta que la reducción tanto del tiempo de cálculo como de los recursos utilizados es una característica ideal a incorporar en la solución, por lo que se procurará explotar las ventajas de procedimientos existentes que proporcionan estas ventajas.

El método debe ser aplicable en entornos bidimensionales y tridimensionales, tanto para manipuladores conocidos como

el manipulador planar o un robot PUMA, como a estructuras redundantes. Así, se diseñarán y propondrán algoritmos que cubran estos casos. Un análisis detallado de los antecedentes bibliográficos en la materia permite poner de manifiesto la ausencia de representaciones de C-espacios para robots redundantes, por lo que la consecución de estos resultados y su interpretatción constituyen un objetivo importante para este trabajo.

Posteriormente, se diseñarán algoritmos completamente nuevos para ser explotados en un entorno paralelo que, dada la naturaleza del problema, permitirán obtener unos mejores resultados que satisfagan los objetivos planteados. Para realizar esta tarea, se explotarán previamente las ventajas de la simulación del rendimiento de los algoritmos paralelos en diferentes entornos, de forma que sea justificable su posterior implementación.

1.6 Organización de esta Memoria

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

El capítulo segundo está dedicado a la evaluación del espacio de las configuraciones. En él se presentan los fundamentos teóricos que sirven de base para desarrollar el método que se presenta en este trabajo de investigación. También se lleva a cabo un análisis detallado de algunas de las técnicas que se han desarrollado en el área, tanto para el caso de robots móviles como para el de los robots articulados, destacando sus ventajas e inconvenientes. De esta manera, se podrá valorar la novedad que supone la introducción del método propuesto, cuya formalización se expone en el capítulo que le sigue.

Así, en el capítulo tercero se propone un formalismo matemático para el nuevo método de evaluación de las configuraciones de un robot que presentarían colisiones con los objetos del espacio de trabajo. Dada la forma particular en que este método descompone el espacio de configuraciones en la unión de distintos conjuntos, que se evalúan por separado según un mismo procedimiento, se le ha denominado *Deconstrucción del espacio de las configuraciones.*

Es en el cuarto capítulo donde se justifica la denominación del método propuesto. Como ya se ha expuesto, el objetivo principal de este trabajo es proporcionar un método general para la evaluación del C-espacio, válido para cualquier estructura robótica. Esto incluye a los robots redundantes. Consecuentemente, este apartado se reserva para ilustrar, a modo de ejemplo, la aplicación del formalismo matemático a algunos manipuladores industriales tanto en un espacio bidimensional, como en uno tridimensional. Para una mejor comprensión del método, se parte del caso más sencillo, un robot planar de una única articulación, hasta uno más complejo, un robot PUMA redundante con cuatro articulaciones. Para todos los casos se proporciona un algoritmo y una respresentación gráfica de la aplicación de la deconstrucción.

A continuación, en el siguiente capítulo se desarrolla una revisión de las técnicas paralelas existentes, así como de la filosofía de trabajo en entornos distribuidos. En esta sección se pretende exponer las nociones básicas del área de la Programación Concurrente para que permita una absoluta comprensión del trabajo desarrollado. También se presenta una introducción al uso de las redes de Petri, y en particular a las redes de Petri coloreadas, que van a servir para describir los algoritmos paralelos propuestos y su simulación.

En ese mismo capítulo, se presentan las diferentes herramientas auxiliares que permiten el diseño y la validación teórica de los algoritmos paralelos (Renew), la consecución de una implementación mejor en términos de rendimiento y efectividad (MPI y FFW), y las que son indispensables para realizar un análisis de los experimentos y ayudan a depurar y mejorar el diseño de las distintas versiones de los algoritmos paralelos (XMPI). Además, se expone un estudio de las ventajas e inconvenientes que aportan cada una de esta herramientas, para poder así justificar su elección.

El siguiente capítulo se ocupa del diseño e implementación de los algoritmos paralelos. Para ello, antes se hace un profuso análisis del formalismo para el cálculo del espacio de las configuraciones bajo el prisma del paralelismo. Se identifican todas las oportunidades generales de introducir paralelismo y se proponen soluciones paralelas genéricas. Posteriormente, siguiendo una de las soluciones propuestas, se aplica al caso, que se considera más general (de los estudiados en el capítulo

cuarto): un robot PUMA redundante. Para ello se expone la metodología de diseño empleada para la mejor obtención de los algoritmos, analizando cada uno de los aspectos que conforman un algoritmo paralelo ideal, y justificando cada una de las decisiones de diseño tomadas. Asimismo, se detalla una descripción exhaustiva de cómo será la implementación del diseño final, y finalmente, se presenta el algoritmo en una red de Petri coloreada. Con ésta se hacen unos experimentos que validan teóricamente el algoritmo paralelo propuesto y justifican su implementación en varios entornos paralelos.

En el séptimo capítulo se presentan los resultados aportados por este trabajo y se valora la bondad de los mismos. Por un lado se analizan los resultados que proporcionan los algoritmos de la deconstrucción para todos los casos analizados en el capítulo cuarto. En concreto, la representación gráfica del C-espacio para los robots redundantes y su interpretación se pueden considerar una de las aportaciones más importantes de este trabajo.

Por otro lado, puesto que uno de los objetivos finales del trabajo es la implementación paralela de los algoritmos propuestos, se presentan los tiempos de cálculo asociados, analizando la especial casuística del cálculo concurrente.

Para finalizar, en el capítulo octavo se exponen las principales conclusiones a las que ha llevado el desarrollo de este trabajo, así como las líneas de investigación que abren los resultados alcanzados.

Además, en sendos apéndices se desarrollan dos aspectos teóricos relevantes para que el lector profundice si esto es necesario: el apéndice A, que explica en qué consiste el método de Denavit-Hartenberg, clásico en la Robótica y de aplicación muy interesante en la deconstrucción del C-espacio; y el apéndice B, que indica, de forma general, cómo se deben discretizar las expresiones matemáticas a las que se llega en el capítulo 3, lo que es necesario para poder implementar los algoritmos que se exponen en el capítulo 4.

Capítulo 2

Evaluación del Espacio de las Configuraciones

El señor Jones vive a 500 millas de ti. Los dos salís de casa a las 5:00 y conducís el uno hacia el otro. El señor Jones viaja a 35 mph, y tu conduces a 40 mph. ¿A qué hora te cruzarás con el señor Jones en la carretera?
Calvin: "Con el tráfico que hay por aquí a las 5:00, ¿quién sabe?". Siempre pillo las preguntas con truco.

Calvin y Hobbes
BILL WATTERSON

YA SE HA COMENTADO que la idea fundamental del espacio de las configuraciones es representar el robot como un punto en un espacio apropiado —el espacio de las configuraciones del robot—. Esta proyección simplifica de forma drástica las tareas de planificar y controlar los movimientos de un robot. Sin embargo, aparece como problema adicional el proyectar los obstáculos en este espacio.

Tanto los obstáculos como el tipo de robot considerado determinarán la proyección. 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. Sin embargo, los robots más implantados en las aplicaciones actuales son los

móviles constituidos por un único elemento y los articulados con un punto fijo. Existen aplicaciones muy específicas donde se utilizan robots móviles articulados, aunque en lo relativo a este trabajo se pueden considerar como un robot móvil más un robot articulado.

Se empezará presentando 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, para un objeto rígido que se mueve. Posteriormente, en las secciones siguientes, se van a analizar de forma detallada los métodos utilizados por diferentes autores para la generación de los C-obstáculos.

2.1 Conceptos Teóricos

Aunque los conceptos de configuración y espacio de las configuraciones ya fueron introducidos en el capítulo previo, en esta sección se presentan de forma más rigurosa, incluyendo unos casos particulares que ayudarán a comprenderlos.

2.1.1 Espacio de las Configuraciones de un Objeto Rígido Móvil

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 $n = 2$ o 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.

Definición 1 *Una configuración q de* **A** *es una especificación de la posición y la orientación de F_A con respecto a F_W.*

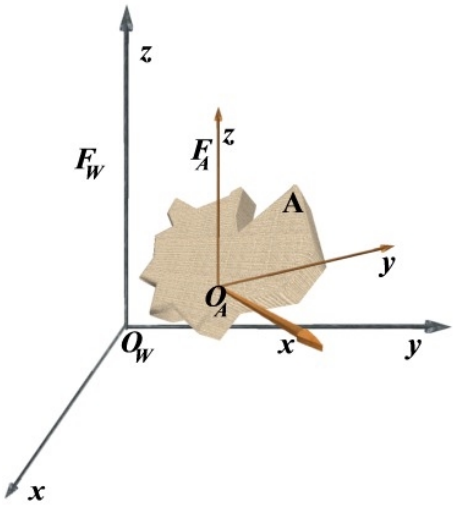


Figura 2.1: Robot móvil en $W = R^3$

La configuración de referencia de **A**, que se denota por 0, es una única configuración de C seleccionada de forma arbitraria.

Definición 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 intrínsicamente 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 **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 **A** está en la configuración q, un punto a de **A** se denota por $a(q)$ en W. Así, para dos configuraciones q y q' cualesquiera, $a(q)$ y $a(q')$ son el mismo punto de **A**, pero 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 para un Objeto Rígido Móvil

En general, W contiene obstáculos fijos que son regiones de R^n. $\mathbf{B}$ denota tanto al obstáculo físicamente 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 3 *El obstáculo* $\mathbf{B}$ *en* W *se proyecta en* C *sobre la región*

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

$\mathbf{C}$B *se denomina obstáculo en el espacio de las configuraciones o C-obstáculo.*

Ésta es la definición propuesta por Lozano-Pérez [LP83] y es admitida de forma unánime en la bibliografía. 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 4 *El conjunto*

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

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

Ejemplo

A continuación se ilustrarán con un ejemplo sencillo las nociones y definiciones establecidas previamente. En concreto, se mostrará una representación gráfica del C-obstáculo que se obtiene cuando se considera un robot móvil cuyo espacio de trabajo se encuentra ocupado por un único obstáculo.

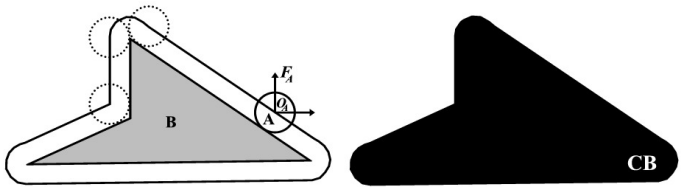


Figura 2.2: C-obstáculo para un disco robótico y un obstáculo poligonal

Sea **A** un robot con forma de disco que se mueve libremente en $W = R^2$, donde existe un obstáculo **B** poligonal como se muestra en la figura 2.2. En W se elige un sistema de referencia F_W. En **A** se establece un sistema de referencia F_A que se mueve con el robot, cuyo origen O_A se sitúa en el centro del disco. En este caso una configuración q vendría dada por la posición del punto de referencia O_A respecto de F_W. Entonces el espacio de las configuraciones sería $C = R^2$. El C-obstáculo **C**B debido al polígono **B** se obtiene aumentado **B** de forma isotrópica con el radio de **A**. El límite de **C**B es la curva seguida por el punto O_A cuando **A** da vueltas sobre el contorno de **B**. De esta forma se tienen definidas todas las posibles configuraciones del robot **A** que producen colisión con el polígono **B**.

2.1.3 Espacio de las Configuraciones para un Robot Articulado

En las definiciones presentadas previamente se había considerado un objeto rígido móvil, en un espacio de trabajo bidimensional o tridimensional. Estas nociones 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.

Un robot articulado $\mathbf{A}$ se compone de una serie de objetos rígidos móviles $\mathbf{A}_1, \ldots, \mathbf{A}_p$ (llamados eslabones) conectados por articulaciones que limitan los movimientos relativos de cada par de objetos que conectan.

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 1 en la página 24, 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 $\mathbf{A}$.

El C-espacio de las configuraciones de un robot articulado con p articulaciones prismáticas o de revolución es una variedad de dimensión p [NS83][Spi82]. Una carta de esta variedad se puede definir simplemente asociando un ángulo, entre 0 y 2π, con cada articulación de revolución y un número real con cada articulación prismática. Entonces, una configuración q se representa por una lista $(q_1, \cdots, q_p)$ de coordenadas, donde cada una describe la posición y la orientación relativa de dos elementos unidos en $\mathbf{A}$. Los robots con articulaciones de revolución tienen espacios de las configuraciones conectados de forma múltiple y son necesarias varias cartas para formar un atlas. Sin embargo, en la práctica, basta con considerar

una única carta para cada coordenada angular perteneciendo al intervalo $[0, 2\pi)$, y aplicar aritmética módulo 2π para cada coordenada.

2.1.4 C-Obstáculos para un Robot Articulado

Existen dos tipos de obstáculos en el espacio de las configuraciones 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 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*

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

Definición 6 *El C-obstáculo debido a la interacción del elemento* $\mathbf{A_i}$ *con el elemento* $\boldsymbol{A_j}$*, siendo* $i < j$*, se denota por* $\boldsymbol{CA_{ij}}$ *y se define por*

$$q = (q_1, \cdots, q_i, \cdots, q_j, \cdots, q_p) \in C /$$

$$\mathbf{A_i}(q_1, \cdots, q_i) \cap \mathbf{A_j}(q_1, \cdots, q_j) \neq 0$$

Ejemplo

Sea $\mathbf{A}$ un robot articulado formado por dos elementos, $\mathbf{A_1}$ y $\mathbf{A_2}$, que son segmentos de rectas y que están conectados por una articulación de revolución. El primer elemento está unido a la base mediante otra articulación de revolución. El robot se

mueve en $W = R^2$, donde existe un conjunto **B** de obstáculos puntuales. Se eligen como sistemas de referencia F_W en W, F_{A_1} unido a $\mathbf{A_1}$ y F_{A_2} unido a $\mathbf{A_2}$, como se muestra en la figura 2.3. Una configuración q vendría dada por la orientación relativa de dos elementos unidos, siendo en este caso $q = (\theta_1, \theta_2)$. Entonces el espacio de las configuraciones sería $C = S^1 \times S^1$, donde S^1 denota el círculo unidad en R^2 [Cur98].

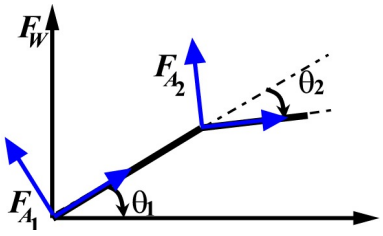


Figura 2.3: Manipulador planar de revolución

En la figura 2.5aparecen los C-obstáculos que se obtienen, debidos a los nueve obstáculos situados en el espacio de trabajo (figura 2.4). 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.

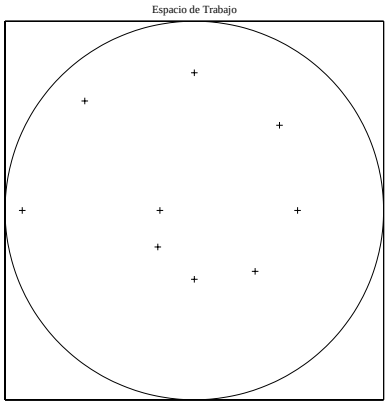


Figura 2.4: Obstáculos en el espacio de trabajo

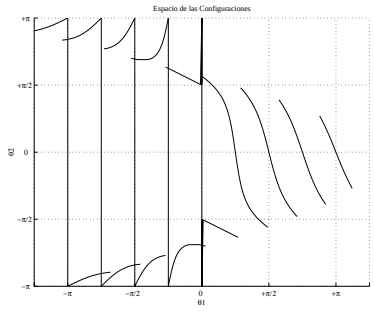


Figura 2.5: Obstáculos en el espacio de las configuraciones

El cálculo de los $\mathbf{CA_{ij}}$ se puede realizar adaptando los métodos que se presentarán para robots móviles. De todas formas, únicamente sería necesario calcularlos una sóla vez para un determinado robot articulado. Sin embargo, los C-obstáculos $\mathbf{CB_{ij}}$ es necesario calcularlos cada vez que su espacio de trabajo se modifique.

Es importante destacar que esto último conlleva una gran complejidad computacional que es necesario optimizar, llegando a la intratabilidad cuando el número de grados de libertad es muy grande —y con ello la dimensión del espacio de las configuraciones—. No es de extrañar, por tanto, que éste haya sido el objetivo fundamental de muchos trabajos de investigación, entre los que se incluye el que recoge esta memoria.

2.2 Métodos para la obtención de los C-obstáculos

Con la intención de proporcionar una visión global de la evolución de las distintas técnicas que han contribuido al avance en la representación de los C-obstáculos, en el capítulo 1 se presentaba una revisión muy somera de los principales trabajos. Ahora, una vez definidos más formalmente los conceptos de espacio de las configuraciones y de C-obstáculo, e introducidos unos sencillos ejemplos donde, intuitivamente, se ponía de manifiesto la forma de generar un C-obstáculo, se van a re-

visar, con mayor grado de detalle, los procedimientos que han proporcionado distintos autores.

Hay que destacar que, como se muestra a continuación, la mayoría de los trabajos, principalmente en sus orígenes, se centran en robots móviles. A continuación, surgieron distintos procedimientos donde se extendían estas técnicas a articulados y, posteriormente, se propusieron nuevos métodos especialmente pensados para este tipo de robot. En general, existen pocos trabajos de investigación dedicados a articulados trabajando en espacios tridimensionales.

Los detractores del espacio de las configuraciones aducen que la principal dificultad que esta idea plantea es proyectar los obstáculos en ese espacio de una forma rápida, eficiente y exacta. El análisis que se presenta a continuación servirá para mostrar cómo se ha intentado salvar estas dificultades. Con él se pretende, además, contrastar las diferentes técnicas surgidas con la propuesta en esta tesis doctoral y, de esta forma, poner de manifiesto las principales aportaciones y mejoras introducidas.

2.2.1 Cálculo de los C-obstáculos para Robots Móviles

En esta sección se van a examinar las diferentes técnicas utilizadas para construir los C-obstáculos, para un objeto rígido que se mueve libremente sobre un plano ocupado por obstáculos. Solamente se considerará que los objetos en el espacio de trabajo, tanto obstáculos como el robot, están representados por modelos algebraicos. Estos modelos presentan como principal ventaja que los objetos se pueden describir por un número 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*.

Representación de un C-obstáculo con C-restricciones

Se caracteriza por utilizar la lógica como herramienta, de forma que define un predicado CB que toma una configuración q como argumento. CB(q) toma valor verdadero si y sólo si $q \in \mathbf{CB}$, donde **CB** es el C-obstáculo representado por CB. Este es el resultado de un teorema propuesto y demostrado en [LP83] para espacios de trabajos poligonales, donde se propone una determinada expresión para el predicado CB. Posteriormente, en [Don84] se propone una ampliación de esta expresión que recoge la extensión a espacios de trabajos poliédricos. Otras expresiones del predicado CB han sido propuestas en [Can88].

Representación límite de un C-obstáculo

En este caso, se trata de una descripción explícita del límite del C-obstáculo **CB**, como una lista de caras, lados y vértices, con sus ecuaciones y su relación topológica de adyacencia. En [LP83] se proponen dos métodos para construir el C-obstáculo, uno con mayor tiempo de ejecución que el otro, donde tanto **A** como **B** pueden ser polígonos o poliedros. En [AB88] se describe otro método que directamente construye el límite de **CB** a partir de la descripción de **A** y **B**, primero para polígonos y después para poliedros.

Características Comunes

En ambas, el coste 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 finalizar, en [Cur98] se realiza una revisión detallada de estas técnicas, las más representativas para la generación de los C-obstáculos para robots móviles, y se llega a las siguientes consideraciones:

1. En todos los algoritmos, una característica importante a destacar es que su tiempo de computación depende, tanto si se trata de polígonos convexos o no convexos

como de poliedros, del número de vértices del robot y de los obstáculos.

2. Una condición necesaria en todos ellos es que los vértices y lados de los obstáculos y del robot han de estar perfectamente identificados. Esto supone que existe, bien un conocimento 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.

2.2.2 Cálculo de los C-obstáculos para Robots Articulados

Existen distintos métodos para construir los C-obstáculos para robots articulados y la mayoría de ellos se centran en manipuladores cuyo espacio de trabajo es R^2. Pocos contemplan manipuladores que trabajen en espacios tridimensionales, como el PUMA, el SCARA, etc. A continuación se comentan las características más destacadas de los principales métodos propuestos.

En general los trabajos desarrollados para la construcción de la región del C-obstáculo para robots articulados usan dos métodos: *Descomposición aproximada en celdas* y *Ecuaciones del límite de un C-obstáculo.*

Descomposición aproximada en celdas

Se trata de aplicar la idea del planteamiento de *descomposición aproximada en celdas* propuesta para robots móviles a robots articulados. Este consiste en decidir si las celdas generadas por la descomposición del C-espacio pertenecen o no a la región del C-obstáculo, utilizando las ecuaciones explícitas que definen los límites de los C-obstáculos. Para móviles estas ecuaciones se pueden establecer utilizando los métodos comentados en la sección previa. Este planteamiento podría ser directamente aplicable a articulados, pero las ecuaciones explícitas de los límites de los C-obstáculos son más difíciles de establecer y de explotar. En su lugar se utiliza un método que consiste en discretizar el intervalo de movimiento de cada articulación en intervalos más pequeños; después, se comprueba si

el robot colisiona o no con los obstáculos cuando las distintas articulaciones se encuentran en estos subintervalos.

Siguiendo este planteamiento se puede calcular, de forma aproximada, la región de los C-obstáculos y construir un grafo de conectividad, que se utiliza para buscar un canal que conecte la configuración inicial a la final en etapa de planificación de movimientos.

En [Cur98] se realiza una revisión detallada de las distintas variantes que ha propiciado este método ([Gou84], [Fav86] y [LP87]).

Ecuaciones del límite de un C-obstáculo

Engloba a aquellos métodos que calculan las ecuaciones exactas del límite del C-obstáculo. En estos trabajos se consideran las ecuaciones algebraicas que definen los obstáculos y 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 obstáculos en el espacio de las configuraciones. Este método, para manipuladores articulados concretos, se presenta en [LPP91] y [NB91] y para cualquier tipo de manipulador general en [GM90] y [Hwa90].

Así como el método de la descomposición aproximada en celdas se aplica indistintamente a un brazo planar o espacial, los procedimientos que calculan las ecuaciones algebraicas de los C-obstáculos dependen en gran medida de los grados de libertad del brazo. En los dos apartados siguientes se analizan por separado los procedimientos más representativos que siguen esta vertiente, para robots planares, en primer lugar, y para espaciales, posteriormente.

- ***Manipuladores articulados en el plano***. Uno de los primeros análisis de la geometría de los límites de los C-obstáculos para distintos brazos planares fue presentado por Lumelsky en [Lum87]. En ese trabajo el autor plantea que la transformación de los obstáculos del espacio de trabajo al espacio de las configuraciones está fuertemente relacionado con la cinemática inversa del mecanismo robótico. Además, para un manipulador planar con dos articulaciones de revolución y cuyos elementos son seg-

mentos de líneas rectas (figura 2.3), las dos configuraciones, codo arriba y codo abajo (distinto signo de θ_2), que puede que adoptar el manipulador para que el efector final esté en contacto con un obstáculo puntual situado en las coordenadas cartesianas (x, y), se obtienen a partir de la resolución de las ecuaciones de la cinemática inversa.

En [LPP91] se propone simplificar las expresiones para θ_1 y θ_2 utilizando las coordenadas polares del punto en lugar de coordenadas cartesianas. Para obtener la transformación de los obstáculos desde el espacio de trabajo al C-espacio hay que calcular todos los puntos de contacto a lo largo de toda la longitud del robot y no sólo con el efector final. Considerando las colisiones con el segundo elemento (con el primero son triviales), la solución no es única, sino que hay un conjunto infinito de soluciones.

Así, se llega a unas expresiones, en función de la distancia, s, desde la segunda articulación al objeto, que únicamente definen los choques del segundo elemento con el obstáculo puntual. Para puntos situados a una distancia s menor que la longitud del primer elemento, hay que tener en cuenta las configuraciones con las que éste contacta con el obstáculo [NB91].

Un método general para obtener las ecuaciones del límite del C-obstáculo para manipuladores planares con n elementos se propone en [GM89]. Utiliza modelos polígonales para los elementos del robot y los obstáculos y determinan los límites de los obstáculos en el espacio imagen —se usa un cuaternión para representar el desplazamiento planar de un sistema de referencia desplazado respecto de uno fijo— del robot. Después, los límites de los obstáculos son transformados de nuevo al espacio de las configuraciones del robot.

Es interesante considerar la geometría de los obstáculos en el espacio de las configuraciones en función de su posición en el espacio de trabajo ([MF93] [NB91]). En la figura 2.4 se muestran nueve obstáculos puntuales que se encuentran en el espacio de trabajo de un manipulador planar de revolución. Están igualmente espaciados, con ϕ variando desde $-\pi$ hasta π con incrementos de $\pi/4$.

Los cuatro que se encuentran situados a mayor distancia radial del origen sólo pueden chocar con el segundo elemento, mientras que el resto pueden colisionar con ambos. En la figura 2.5 se muestran los nueve obstáculos en el espacio de las configuraciones.

- Los cuatro obstáculos más alejados sólo pueden colisionar con el segundo elemento y aparecen con forma de S invertida.
- El resto (que pueden colisionar con ambos elementos) se muestran como líneas rectas, paralelas al eje θ_2, para los choques con el primer elemento; y con líneas curvadas para las colisiones con el segundo elemento.

En [LPP91] y [NB91] se propone construir mapas de primitivas del C-espacio para un manipulador dado almacenando el C-espacio generado a partir del hecho de que el único parámetro que afecta a la curva del C-obstáculo es el radio r donde se encuentra el obstáculo puntual, mientras que el ángulo ϕ solamente introduce un desplazamiento en θ_1

En [MF93] se obtienen las ecuaciones de la cinemática inversa para un manipulador con dos elementos poligonales y también en [LPP91] y [MF93] se considera la extensión a obstáculos poligonales.

- ***Manipuladores articulados en el espacio***. En [LPP91], [NB91] y [MF93] se propone la extensión de sus algoritmos a robots espaciales, mediante una técnica denominada *slicing*. Esta es una variante del planteamiento de la descomposición aproximada en celdas. Consiste en descomponer el recorrido de uno de los grados de libertad en un número finito de intervalos. En cada intervalo, se calculan los contactos con los obstáculos considerando únicamente los grados de libertad restantes. Este planteamiento es aplicable directamente a estructuras mecánicas donde dos ejes de articulación intersecten.

  Concretamente, en [LPP91], [NB91] y [MF93] aplican el método a estructuras mecánicas tipo PUMA, aunque sólo analizan las tres primeras articulaciones, obteniendo

C-espacios tridimensionales. Así, consideran que los dos últimos elementos trabajan en un plano, al igual que el manipulador planar con dos articulaciones de revolución analizado previamente. Dicho plano viene determinado para un valor concreto de θ_0, el grado de libertad asociado a la articulación de la cintura. Como en el caso del planar, los C-obstáculos forman una familia de curvas parametrizadas por un parámetro. De hecho, es exactamente la misma familia de curvas, ya que el efecto de θ_0 es seleccionar el plano donde se produce la interacción con el obstáculo. En tres dimensiones la forma del obstáculo transformado depende únicamente del radio r donde se encuentra. Dependiendo de la posición del punto, esta primitiva se traslada en el C-espacio una cantidad ϕ en θ_1 y ϕ_0 en θ_0, donde ϕ y ϕ_0 son las variables angulares del obstáculo, considerando coordenadas esféricas.

El método propuesto por [GM89] para robots planares y obstáculos poligonales se generaliza en un trabajo posterior [GM90] para robots espaciales y obstáculos poliédricos. En [GM90] se presenta una formulación algebraica de los límites de los C-obstáculos para robots espaciales con desacoplo cinemático. Esta formulación se aplica a un robot PUMA para obtener las ecuaciones parametrizadas del límite de los C-obstáculos. Sin embargo, en este método se supone una fuerte simplificación, como es la de modelar el brazo por una cadena de dos articulaciones esféricas. Además, solamente considera la parte del límite del C-obstáculo debida a choques con el efector final.

Existe otro trabajo [Hwa90] para obtener las ecuaciones del límite de los C-obstáculos para manipuladores que trabajan en entornos poliédricos. Los obstáculos se modelan como uniones de caras triangulares, y los manipuladores mediante segmentos de línea. Las ecuaciones del límite de los C-obstáculos se calculan a partir de las condiciones que definen la intersección entre las caras triangulares y los segmentos de línea. Además, se muestra que las expresiones explícitas de los límites se pueden obtener para manipuladores cuya cinemática se represente con la notación de Denavit-Hatenberg. Más

concretamente, se muestra que la ecuación del límite para la variable de una determinada articulación se puede resolver explícitamente en función de las variables de las articulaciones previas. Para validar el método, se aplica a un robot Adept cuyos elementos se aproximan por una unión de cilindros. Los obstáculos se aumentan en todas las direcciones por el radio de los cilindros y los elementos del robot se reducen a los ejes del cilindro. Se analizan las intersecciones en los ejes del cilindro y las uniones de los triángulos que aproximan a las superficies de las obstáculos aumentados.

Curto llega a las siguientes consideraciones en [Cur98], donde analiza con mayor detalle los métodos comentados:

1. En los métodos que utilizan la descomposición en celdas es necesario construir una representación, más o menos aproximada, de los elementos que constituyen el robot para un conjunto elevado de configuraciones. En alguno de ellos hay que construirla para todas las configuraciones. Esto supone una elevada carga computacional asociada al cálculo de la cinemática directa.

   Además, habría que aplicar de forma explícita un test de colisiones para todos los elementos en todas las configuraciones consideradas. Este test de colisiones sería realizado con los métodos expuestos para los robots móviles.

2. Respecto al segundo grupo, hay que destacar que no existen muchos trabajos dedicados a robots articulados, y concretamente a brazos espaciales. Los procedimientos propuestos calculan explícitamente las ecuaciones del límite de los C-obstáculos realizando una serie de simplificaciones. En primer lugar, simplifican la geometría de los elementos del robot y de los obstáculos. Concretamente, modelan los objetos mediante regiones algebraicas. Esto supone aproximar tanto a los elementos del robot como a los obstáculos por un conjunto de formas geométricas. Incluso, se llega a modelar al robot mediante segmentos de línea. Además, existen distintas técnicas que sustituyen la estructura mécanica del robot por otra más fácil de analizar. Todas estas simplificaciones, en ge-

neral, son muy restrictivas para manipuladores articulados trabajando en espacios tridimensionales, aunque no tanto para planares.

En segundo lugar, no todas las técnicas tienen en cuenta las colisiones entre todos los elementos que constituyen el brazo y los obstáculos. Normalmente, se restringen al estudio de los contactos con el efector final del robot.

Como consecuencia de estas simplificaciones, aunque con estos métodos se logre disminuir el tiempo de computación, los C-obstáculos que se obtienen son una versión aproximada de los reales.

2.2.3 Uso de la Convolución para Calcular los C-obstáculos

Para robots móviles, Kavraki [Kav95] propone calcular una matriz binaria que representa al C-espacio como la convolución de una matriz binaria de los obstáculos en el espacio de trabajo y de otra del robot. Para realizar la convolución eficientemente utiliza la Transformada Rápida de Fourier. Con el método propuesto en [Kav95] se obtiene un bitmap del espacio de las configuraciones más exacto que con los que utilizan primitivas elementales. Además es independiente del número de obstáculos, de la forma de estos y de la del robot y, únicamente, depende de la resolución con la que se discretiza. En dicho trabajo solamente se tienen en consideración objetos poligonales que se mueven sobre un plano. Para esta situación particular la idea de aplicar la convolución se puede intuir con facilidad. Sin embargo, no aporta ningún formalismo matemático que permita extender el planteamiento a otras situaciones con mayor complejidad, como los robots articulados en R^3.

Así, siguiendo la misma filosofía, Curto [Cur98] propone un nuevo método que abarca tanto a robots móviles como articulados, en dos y tres dimensiones. El método está sustentado por un formalismo matemático sólido y de vocación general, cuyo punto clave es la proposición de una nueva expresión matemática para calcular los C-obstáculos, equivalente a las propuestas por otros autores [LP83] (definición 3). En concreto, esta expresión permite calcular los C-obstáculos como la integral extendida sobre el espacio de trabajo del producto de

dos funciones: una que describe al robot y otra que describe a los obstáculos. Al igual que Kavraki, Curto utiliza como herramienta matemática la transformada de Fourier.

En el método propuesto en [Cur98] se obtiene una representación explícita de los obstáculos en el C-espacio para manipuladores, ampliamente utilizados en espacios tridimensionales, como el PUMA, el SCARA y el Stanford, lo que supone un avance importante, pues no se encuentran en la bibliografía resultados para este tipo de manipuladores.

Por otro lado, la extensión del método de Kavraki aportada por Curto no se obtiene en detrimento de una de las mayores ventajas del primero: el coste computacional de los algoritmos que se proponen en [Cur98] es independiente de la geometría tanto de los objetos que componen el robot como de los situados en su entorno, sin ninguna contraprestación en cuanto a la exactitud de la representación obtenida.

Sin embargo, se deben tener en cuenta las siguientes consideraciones:

- Los métodos para obtener los C-obstáculos mediante la convolución, al ser métodos discretos, tienen un gran consumo de memoria cuando se pretende proporcionar realismo, al ser necesaria una resolución de discretización elevada.

- Aunque el método propuesto en [Cur98] es válido tanto para robots móviles como para robots articulados, no es generalizable a cualquier tipo de estructura robótica, como es el caso de los robots redundantes.

Por tanto, dos líneas de investigación se abren si se quieren solventar los problemas mencionados.

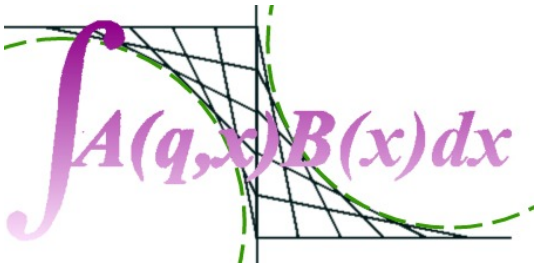
Así, una alternativa para reducir las necesidades de memoria es el uso de estructuras jerárquicas de datos tanto para las representaciones como para el cálculo. De esta manera se puede ir refinando la representación el C-espacio solamente en aquellas zonas donde sea necesario, sin necesidad de representar todo el espacio a gran resolución. Un ejemplo de este enfoque se tiene en [BTCM01].

El otro aspecto mencionado, llegar a un método general que sea aplicable a cualquier estructura robótica, incluyendo

el caso de los robots redundantes, constituye el objeto de este trabajo de investigación.

Capítulo 3

Formalismo Matemático para la Deconstrucción del C-espacio



Calvin: Hobbes, he estado pensando.
Hobbes: ¿Un fin de semana?
Calvin: Bueno, no lo hice queriendo.

El Gran Calvin y Hobbes Ilustrado
BILL WATTERSON

En el presente capítulo se va a presentar un nuevo formalismo matemático que permite la evaluación de las posibles colisiones entre cualquier elemento de una estructura robótica, redundante o no, con objetos que ocupan el espacio de trabajo; en definitiva, se propone un nuevo método formal para la representación de los C-obstáculos general para todo tipo de robots.

Este método sigue la línea iniciada por [Kav95] y continuada por [Cur98], por lo que parte del formalismo coincide con el propuesto por Curto. Sin embargo, el formalismo que se propone supera las limitaciones de métodos anteriores, pues gracias a la deconstrucción del espacio de las configuraciones, esto es gracias a la posibilidad de evaluar el C-espacio de forma separada para cada uno de los elementos que forman el robot, y además, mediante la repetición, para cada uno de ellos, de un método básico de evaluación, se puede aplicar a cualquier estructura robótica.

3.1 Planteamiento del Problema

Se considera como punto de partida un robot, formado por un único objeto rígido, que se mueve en un espacio de trabajo ocupado por objetos estáticos. Inicialmente, se pueden ignorar las propiedades dinámicas del robot, las restricciones mecánicas de los movimientos —los cuales, de este modo, están sólo

limitados por los obstáculos— y los aspectos relacionados con el tiempo [Lat91].

3.1.1 Uso del Espacio de las Configuraciones

En [Cur98] se propone una representación de los C-obstáculos (coherente con [LP83]) basada en la integral del producto de dos funciones: una que representa al robot y otra a los obstáculos en el espacio de trabajo.

A continuación se detallan estas funciones.

Sean W y C el espacio de trabajo y el espacio de las configuraciones de un robot, respectivamente. Así, la función A que se propone para representar al robot situado en un punto $x \in W$ en la configuración $q \in C$, vendría dada por la siguiente definición.

Definición 7 *Sea* $\mathbf{A}(q)$ *el subconjunto de* W *que representa al robot en la configuración* q*. Se define la función* $A : C \times W \rightarrow R$ *como*

$$A(q,x) = \begin{cases} 1 & si\ x \in \mathbf{A}(q) \\ 0 & si\ x \notin \mathbf{A}(q) \end{cases} \tag{3.1}$$

De la misma forma, la función B, que se propone para representar a los obstáculos en el espacio de trabajo, estaría definida como se muestra a continuación.

Definición 8 *Sea* $\mathbf{B}$ *el subconjunto de* W *formado por los obstáculos. Se define la función* $B : W \rightarrow R$ *como*

$$B(x) = \begin{cases} 1 & si\ x \in \mathbf{B} \\ 0 & si\ x \notin \mathbf{B} \end{cases} \tag{3.2}$$

Con estas dos funciones A y B, Curto propone una nueva definición para calcular los C-obstáculos:

Definición 9 *Sea la función* $CB : C \rightarrow R$ *definida como*

$$CB(q) = \int A(q,x)B(x)dx \quad \forall q \in C, \quad \forall x \in W \quad (3.3)$$

Se define la región $\mathbf{CB_f}$ *como el subconjunto de* C *que verifica*

$$\mathbf{CB_f} = \{q \in C / CB(q) > 0\} \quad (3.4)$$

En [Cur98] se demuestra que $\mathbf{CB_f}$ coincide con la definición de C-obstáculo propuesta por Lozano-Pérez en [LP83], que es la más comunmente aceptada en el campo de la robótica.

3.1.2 Parametrización de los espacios

En la sección anterior se presentaban las expresiones para las funciones CB (expresión 3.3), B (expresión 3.2) y A (expresión 3.1), definidas, respectivamente, sobre los espacios C, W, y $C \times W$, sin suponer ninguna parametrización específica de W y de C.

Es éste el momento de seleccionar una representación, tanto de C como de W, mediante una lista de parámetros reales, que permitirá obtener la correspondiente forma parametrizada de las funciones previamente definidas.

Una representación de W y de C viene dada seleccionando un sistema de referencia F_W para el espacio de trabajo y otro F_A para el robot. Respecto a ellos se eligen las coordenadas de trabajo para C y W, de tal forma, que un punto x de W vendrá expresado respecto al sistema de referencia F_W por las coordenadas

$$(x_1, x_2, \cdots, x_n)$$

donde n es la dimensión de W, es decir, $n \leq 3$. Una configuración q viene representada por una lista de parámetros

$$(q_1, q_2, \cdots, q_m)$$

que especifican la posición y orientación de F_A con repecto a F_W, siendo m la dimensión de C.

Con esta parametrización de C y W la función A dada previamente en la expresón 3.1

$$\begin{array}{rcl} A : C \times W & \longrightarrow & R \\ (q_1, \cdots, q_m) \times (x_1, \cdots, x_n) & \longrightarrow & A(q_1, \cdots, q_m, x_1, \cdots, x_n) \end{array}$$

estaría definida como

$$A(q_1, \cdots, q_m, x_1, \cdots, x_n) = \begin{cases} 1 & \text{si } (x_1, \cdots, x_n) \in \mathbf{A}(q_1, \cdots, q_m) \\ 0 & \text{si } (x_1, \cdots, x_n) \notin \mathbf{A}(q_1, \cdots, q_m) \end{cases} \tag{3.5}$$

donde $\mathbf{A}(q_1, \cdots, q_m)$ es el subconjunto de W que representa al robot en la configuración $(q_1, \cdots, q_m)$.

De la misma forma, la función B de la expresión 3.2,

$$\begin{array}{rcl} B : W & \longrightarrow & R \\ (x_1, \cdots, x_n) & \longrightarrow & B(x_1, \cdots, x_n) \end{array}$$

se definiría como

$$B(x_1, \cdots, x_n) = \begin{cases} 1 & \text{si } (x_1, \cdots, x_n) \in \mathbf{B} \\ 0 & \text{si } (x_1, \cdots, x_n) \notin \mathbf{B} \end{cases} \tag{3.6}$$

siendo $\mathbf{B}$ el subconjunto de W formado por los obstáculos.

La función CB (expresión 3.3), después de la parametrización,

$$\begin{array}{rcl} CB : C & \longrightarrow & R \\ (q_1, \cdots, q_m) & \longrightarrow & CB(q_1, \cdots, q_m) \end{array}$$

quedaría como $CB(q_1, \cdots, q_m)$, que se calcularía así

$$\int A(q_1, \cdots, q_m, x_1, \cdots, x_n) B(x_1, \cdots, x_n) dx_1 \cdots dx_n \tag{3.7}$$

En la sección siguiente se comprobará que tanto la colocación de los sistemas de referencia, como la elección de las funciones coordenadas —en el espacio de trabajo y en el de las configuraciones—, son cruciales para el presente trabajo. Los primeros permiten dotar de generalidad al método mientras que las segundas proporcionan una reducción de la complejidad computacional asociada.

3.2 Cadenas Cinemáticas

Una de las posibilidades que existen es tratar a los robots articulados como cadenas cinemáticas. El robot $\mathbf{A}$ puede verse como un conjunto de r objetos rígidos; la cinemática de esta cadena, o lo que es lo mismo, las limitaciones en los movimientos impuestas por las articulaciones a cada uno de los objetos $\mathbf{A}_i$ —los grados de libertad (DOFs)—, determinará cada subconjunto del espacio de las configuraciones para dichos objetos. La unión de estos subconjuntos se corresponde con el espacio de las configuraciones para $\mathbf{A}$.

Un aspecto importante a tener en cuenta es que la dimensión del espacio de las configuraciones crece con el número articulaciones del robot, lo que también significa un aumento de la complejidad de su cálculo.

3.2.1 Propiedad de Superposición para C-obstáculos

Si ahora se considera que el robot está formado por r cuerpos rígidos, para los C-obstáculos resultantes se cumple el **teorema de superposición**:

Teorema 1 *Sea* $\mathbf{A}$ *un robot articulado formado por* r *elementos* $\mathbf{A}_1, \ldots, \mathbf{A}_r$. *Si* $\mathbf{CB}_1, \ldots, \mathbf{CB}_r$ *son, respectivamente, las regiones de los C-obstáculos debidas a los elementos* $\mathbf{A}_1, \ldots, \mathbf{A}_r$ *donde se proyecta el obstáculo* $\mathbf{B}$, *entonces la región* $\mathbf{CB}$ *del C-obstáculo debida al robot* $\mathbf{A}$ *donde se proyecta* $\mathbf{B}$ *se obtiene como*

$$\mathbf{CB} = \bigcup_{i=1}^{r} \mathbf{CB}_i \tag{3.8}$$

Esta propiedad es la base del cálculo de los C-obstáculos para robots formados por varios elementos conectados mediante diferentes tipos de articulaciones y se demuestra a continuación.

—**Demostración**— $1^a parte:\ \mathbf{CB} \subseteq \bigcup_{i=1}^{r} \mathbf{CB}_i$

$\forall q \in \mathbf{CB} \Rightarrow CB(q) > 0$, como

$$CB(q) = \int A(q,x)B(x)dx > 0 \Rightarrow \exists x,\ x \in \mathbf{B} \text{ y } x \in \mathbf{A}_q$$

Además

$$\mathbf{A}_q = \bigcup_{i=1}^{r} \mathbf{A}_{i_q}$$

por tanto, $\exists k \in \{1,\ldots,r\},\ x \in \mathbf{B}$ y $x \in \mathbf{A}_q$, que por ser funciones definidas positivas, $\int A_k(q,x)B(x)dx > 0$, y entonces

$$q \in \mathbf{CB}_k$$

—**Demostración**— $2^a parte:\ \bigcup_{i=1}^{r} \mathbf{CB}_i \subseteq \mathbf{CB}$

$$\forall q \in \bigcup_{i=1}^{r} \mathbf{CB}_i \Rightarrow \exists k \in \{1,\ldots,r\},\ q \in \mathbf{CB}_k$$

Si se cumple que

$$q \in \mathbf{CB}_k \Rightarrow \mathbf{CB}_k(q) = \int A_k(q,x)B(x)dx > 0$$

donde, por ser funciones definidas positivas, $\exists x,\ x \in \mathbf{B}$ y $x \in \mathbf{A}_k$. Si $x \in \mathbf{A}_k$, entonces

$$x \in \bigcup_{i=1}^{r} \mathbf{A}_i = \mathbf{A}$$

y por tanto, $\int A(q,x)B(x)dx > 0 \Rightarrow q \in \mathbf{CB}$

La propiedad de superposición de los C-obstáculos es la clave que habilita la deconstrucciónn, principal aportación de

esta tesis doctoral, ya que tratando individualmente cada uno de los eslabones se consigue simplificar el cálculo de **C**B.

En la siguiente sección se propone un nuevo formalismo matemático que explota este concepto.

3.3 Deconstrucción del Cálculo de CB

A continuación se expone el formalismo matemático que se propone para proyectar los obstáculos del espacio de trabajo en el espacio de las configuraciones para cualquier tipo de estructura robótica, incluídas las redundantes, para las que no se conoce la existencia de ningún método formal. Se considera que un robot —redundante o no— está formado por un número determinado de elementos, unidos por articulaciones, que forman la cadena cinemática.

3.3.1 Aplicación de la Propiedad de Superposición

Teniendo en cuenta la expresión 3.8 correspondiente a la propiedad de superposición, el cálculo de **CB** para un robot **A** formado por un cadena cinemática de r elementos, vendría dado por la unión de todos los $\mathbf{CB}_k$ correspondientes a cada uno de los elementos del robot. El cálculo de cada uno de estos conjuntos vendría dado analíticamente, por la evaluación de las funciones CB_k asociadas,

$$CB_k(q_{1_k}, \cdots, q_{s_k}),\ \forall k \in \{1, \ldots, r\} \tag{3.9}$$

con $\{q_{1_k}, \cdots, q_{s_k}\} \subseteq \{q_1, \cdots, q_m\}$, donde, como se vió en el punto anterior, $\{q_1, \cdots, q_m\}$ son las variables de configuración asociadas al robot **A**. Es decir, para el $k-$ésimo elemento se tiene en cuenta únicamente el subconjunto de variables de configuración asociadas a él, y donde, de forma similar a la expresión 3.7, cada una de las funciones $CB_k(q_{1_k}, \cdots, q_{s_k})$ se calcula de la siguiente manera

$$\int A_k(q_{1_k}, \cdots, q_{s_k}, x_1, \cdots, x_n) B(x_1, \cdots, x_n) dx_1 \cdots dx_n \tag{3.10}$$

3.3.2 Elección de los Sistemas de Referencia

Para resolver la integral 3.10 surge el problema de que es necesaria la evaluación de la posición precisa de la función $A_k(q_{1_k}, \cdots, q_{s_k}, x_1, \cdots, x_n)$, tarea complicada por su dependencia de todos los grados de libertad relativos a ella y a los elementos anteriores en la cadena. Sin embargo, este escollo es superable aprovechando la posibilidad de elección de un sistema de referencia más adecuado para evitar la necesidad de dicha evaluación.

Para ello, se considera el robot formado por la cadena cinemática de la figura 3.1 (los tres puntos negros simbolizan un número indeterminado de elementos en la cadena). Como puede observarse, siguiendo el procedimiento de Denavit-Hartenberg[DH55] (ver apéndice A), se asocia un sistema de referencia a cada elemento, situando el origen al final del mismo y con una orientación de los ejes dependiente de la posición y orientación del elemento.

Desde que los introdujeran Denavit y Hartenberg, se asocian cuatro parámetros con cada eslabón de la cadena : a_i, la longitud normal entre los ejes de las articulaciones asociadas; α_i, el ángulo entre los dos ejes de las articulaciones; d_i, la posición relativa de los dos eslabones (distancia entre a_i y a_{i-1}); y θ_i, el ángulo entre a_i y a_{i-1}.

Se propone utilizar para el elemento $k-$ésimo el sistema de referencia determinado por los elementos anteriores. Así, para el elemento 1, se utiliza el sistema de referencia F_{A_0}, que coincide con el sistema de referencia asociado al espacio de trabajo, F_W; para el elemento $k-$ésimo se trabajará con el sistema de referencia $F_{A_{k-1}}$.

De este modo, $A_k(q_{1_k}, \cdots, q_{s_k}, x_1, \cdots, x_n)$, la expresión a evaluar, puede leerse de esta otra forma

$$A_k(\underbrace{q_{1_k}, \cdots, q_{u_k}}_{DOF_{(1,\ldots,k-1)}}, \underbrace{q_{(u+1)_k}, \cdots, q_{s_k}}_{DOF_k}, x_1, \cdots, x_n) \qquad (3.11)$$

donde $(q_{1_k}, \cdots, q_{u_k})$ son los grados de libertad asociados a los elementos anteriores en la cadena al que se esta considerando, el elemento k, cuyos grados de libertad son $(q_{(u+1)_k}, \cdots, q_{s_k})$.

En este punto, la posición y orientación del elemento $\mathbf{A}_k$ se expresa en relación con el sistema de referencia FA_0. La

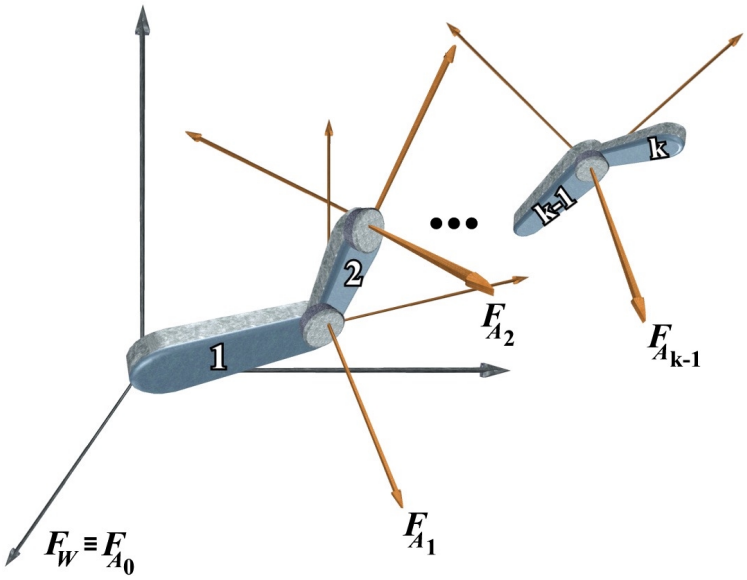


Figura 3.1: Sistemas de referencia en la cadena cinemática de un robot

posición, al igual que el sistema de referencia $F_{A_{k-1}}$, viene determinada por los grados de libertad asociados a los eslabones anteriores en la cadena, es decir, algunos de los parámetros relativos a cada $\mathbf{A}_i$ —elementos anteriores— en esa subcadena, (a_i, α_i, d_i y θ_i).

Así, si se expresa la posición y orientación del elemento $\mathbf{A}_k$ respecto al sistema de referencia $F_{A_{k-1}}$, su evaluación será mucho más sencilla. Para ello se necesita introducir una transformación homogénea $\mathbf{T}$.

Definición 10 *Sea* ${}_0^{k-1}\mathbf{T}$ *la función que permite llevar el sistema de referencia* F_{A_0} *hasta coincidir con* $F_{A_{k-1}}$.

Utilizar esta tranformación homogénea, tiene una consecuencia: será necesario expresar el espacio de trabajo, que es-

taba dado respecto al sistema de referencia F_W, $(x_1, \cdots, x_n)$, en función del nuevo sistema de referencia, $F_{A_{k-1}}$. Es importante resaltar que esta transformación homogénea es función de la posición y orientación de los elementos anteriores en la cadena, es decir, ${}^{k-1}_{0}\mathbf{T} = f(q_{1_k}, \cdots, q_{u_k})$.

En este momento, la posición y orientación del eslabón $\mathbf{A}_k$ expresada en relación con el sistema de referencia $F_{A_{k-1}}$, sólo dependerá de los grados de libertad asociados a él, es decir, $(q_{(u+1)_k}, \cdots, q_{s_k})$. Con lo que la evaluación de la expresión 3.11 es equivalente a la de esta otra

$$A'_k(q_{(u+1)_k}, \cdots, q_{s_k}, x'_1, \cdots, x'_n) \tag{3.12}$$

con

$$(x'_1, \cdots, x'_n) = {}^{k-1}_{0}\mathbf{T}(x_1, \cdots, x_n) \tag{3.13}$$

Finalmente, la expresión 3.10, que permite calcular la porción de C-obstáculo correspondiente al elemento $\mathbf{A}_k$, quedaría como sigue

$$\int A'_k(q_{(u+1)_k}, \cdots, q_{s_k}, x'_1, \cdots, x'_n) B'(x'_1, \cdots, x'_n) dx'_1 \cdots dx'_n \tag{3.14}$$

3.3.3 Elección de Funciones Coordenadas

Kavraki, en [Kav95], propone la simplificación del cálculo del espacio de las configuraciones aprovechando la existencia de un producto de convolución —y realizando éste usando la transformada de Fourier— entre las funciones A y B. Sin embargo, esta ventaja sólo era aprovechable en el caso de robots móviles. Curto amplía este concepto en [Cur98], habilitando el caso de los robots articulados mediante la introducción de un cambio de funciones coordenadas, si bien, este método no era aplicable al caso de robots redundantes.

A continuación se muestra cómo este cambio de funciones coordenadas es aplicable dentro del nuevo formalismo propuesto, salvando así la barrera que tenían los métodos anteriores. Los beneficios que proporciona se muestran en dos fases: aparición del producto de convolución y uso de la transformada de Fourier.

Producto de Convolución en el Cálculo de CB_k

Como se demuestra en [Cur98], basta utilizar las funciones coordenadas adecuadas, $(\xi_1, \cdots, \xi_n)$, que permitan dos cosas:

- Encontrar una dependencia funcional más sencilla en la función A'_k, de forma que el elemento $\mathbf{A}_k$ sea independiente de un subconjunto de $(q_{(u+1)_k}, \cdots, q_{s_k})$, es decir, sólo dependa de $(q_{(v+1)_k}, \cdots, q_{s_k})$.
- Encontrar alguna relación entre alguna de las variables de configuración de las que depende y algunas de las funciones coordenadas, lo que permitirá encontrar convolución.

Así, habrá que definir una nueva función $\bar{A}'_k$, tal que la integral 3.14 venga dada por

$$\int \bar{A}'_k(0,\cdots,0,q_{(v+1)_k},\cdots,q_{s_k},\xi_1-q_{(u+1)_k},\cdots,\xi_v-q_{v_k},\xi_{(v+1)_k},\cdots,\xi_n)B'(\xi_1,\cdots,\xi_n)d\xi_1\cdots d\xi_n \tag{3.15}$$

con lo que se consigue que la función $\bar{A}'_k$ sólo dependa de $(q_{(v+1)_k}, \cdots, q_{s_k})$. Así, para las variables $(q_{(u+1)_k}, \cdots, q_{v_k})$ se produce convolución, apareciendo el producto de convolución que se expresa a continuación.

$$\int (\bar{A}'_{k\,(0,\cdots,0,q_{(v+1)_k},\cdots,q_{s_k})} * B)_{(\xi_1,\cdots,\xi_{v_k})}(\xi_{(v+1)_k},\cdots,\xi_n)\, d\xi_{(v+1)_k}\cdots d\xi_n \tag{3.16}$$

donde los subíndices $(\xi_1, \cdots, \xi_{v_k})$ indican que el producto de convolución se realiza para todos los valores de estas variables.

La Transformada de Fourier en el Cálculo de CB_k

Al igual que en los métodos comentados ([Cur98] y [Kav95]), se puede usar la transformada de Fourier, reduciendo la complejidad del cálculo de CB_k.

El teorema de convolución establece que dadas dos funciones $f_1(x)$ y $f_2(x)$ definidas en R, cuyas transformadas de Fourier son respectivamente $\mathcal{F}[f_1(x)]$ y $\mathcal{F}[f_2(x)]$, entonces:

$$\mathcal{F}[(f_1 * f_2)(x)] = \mathcal{F}[f_1(x)] \cdot \mathcal{F}[f_1(x)]$$

De esta forma, se puede calcular el producto de convolución de dos funciones como la transformada de Fourier inversa del producto de las transformadas de ambas funciones.

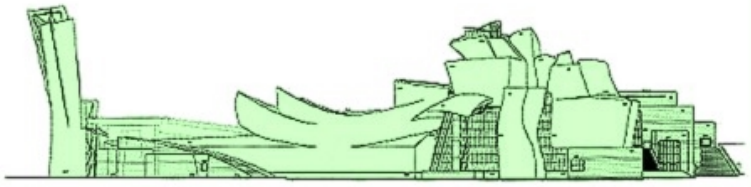
Y así, aplicando lo expuesto en la expresión 3.16, el cálculo de CB_k se obtendría realizando la transformada de Fourier inversa de la siguiente integral

$$\int \mathcal{F}\Big[\bar{A}'_{(0,\cdots,0,q_{(v+1)_k},\cdots,q_{s_k})}(q_{(u+1)_k},\cdots,q_{v_k},\xi_{(v+1)_k},\cdots,\xi_n)\Big]_{(x_1,\cdots,x_{v_k})} \cdot\mathcal{F}\big[B'(\xi_{(v+1)_k},\cdots,\xi_n)\big]_{(\xi_1,\cdots,\xi_{v_k})} d\xi_{(v+1)_k}\cdots d\xi_n \tag{3.17}$$

Los subíndices $(\xi_1,\cdots,\xi_{v_k})$ indican que se trata de transformadas de dimensión v_k de funciones definidas en R^n. Sobre el resto de las coordenadas, $(\xi_{(v+1)_k},\cdots,\xi_n)$, donde no se produce convolución, se efectúa la integración.

Capítulo 4

Hacia la Deconstrucción del C-Espacio



Calvin: Querría ser
Neo-deconstructivista,
pero mamá no me dejaría.

Un Mundo Mágico
BILL WATTERSON

La Real Academia Española define el verbo **deconstruir** de la siguiente manera: *Deshacer analíticamente los elementos que constituyen una estructura conceptual*; si se hace una observación adecuada del formalismo propuesto en el capítulo anterior, es evidente que el proceso que permite finalmente representar aquellas configuraciones de un robot, que resultarán en una colisión con algún objeto del espacio de trabajo, es precisamente la *acción y efecto de deconstruir* el espacio de las configuraciones, esto es, hacer su **deconstrucción**[1]. La idea de analizar y explotar los componentes de una estructura en un espacio ya se utiliza en la enseñanza de arquitectura [BCP01], donde se entiende por deconstrucción la individualización de subsistemas de partes homogéneas y su representación separada.

[1] En este texto se utiliza el término **deconstrucción** de forma literal, no en el sentido filosófico introducido por Jacques Derrida, si bien, la conexión entre el proceso propuesto y las ideas de Derrida es innegable: *El conjunto* —el C-espacio en nuestro caso— *es una especie de palimpsesto, donde las capas se superponen sin que haya una que sea más fundamental o más fundadora que la otra* [Der88]; más aún: *Y la deconstrucción no es más que la subversión de la lógica arquitectónica de adición [...] como no hay nada encima o debajo de los pliegues convolucionados de la superficie, es cuestión de seguir alguna línea circular de investigación, pasando una y otra vez por el mismo conjunto pequeño de temas, circulando obsesivamente* [Wig95]; o en palabras del propio Derrida en [Der99]: *Hay algo que ha sido construido [...] y entonces llega un deconstructor y destruye la construcción piedra a piedra, analiza la estructura y la deshace*; sin embargo, más tarde añade que *la deconstrucción no es sólo —como su nombre parecería indicar— la técnica de una «construcción trastocada»*. Queda así aplazado el debate filosófico a otro momento y lugar.

4.1 El Camino de la Deconstrucción

A continuación se va a seguir un planteamiento incremental que permita explicar el concepto de deconstrucción presente en el formalismo propuesto en el capítulo anterior. Se comenzará analizando el espacio de las configuraciones para el caso más sencillo, un robot planar de una sola articulación que gira en un plano. En este primer ejemplo ya se asentarán algunos conceptos generalizables a estructuras cinemáticas más complejas y se presentará el algoritmo correspondiente.

Dado que el procedimiento de frconstrucción tiene sentido para cadenas cinemáticas, se analizarán los casos correspondientes a un robot planar de dos articulaciones, un robot planar redundante de tres articulaciones, un robot PUMA de tres articulaciones, y para finalizar, también en un espacio tridimensional, un robot redundante de cuatro articulaciones.

Todos los algoritmos propuestos producen como resultado una representación de las configuraciones que producirían colisiones con los obstáculos del espacio de trabajo. Los resultados para los casos estudiados coinciden con los que se encuentran en la bibliografía, salvo para los casos de robots redundantes, para los que existen pocos trabajos previos. Esta situación última se debe a la complejidad que se añade con cada nuevo grado de libertad, al tener que evaluar el robot para todas sus variables de configuración. Destacan así dos de las ventajas más importantes de la deconstrucción: cada elemento de la cadena cinemática se va a tratar individualmente y sin necesidad de considerar al elemento en todas las configuraciones posibles (gracias a que se posibilitan convoluciones para cada elemento de la cadena).

Se puede añadir finalmente que esta revisión es lo suficientemente explicativa de la aplicabilidad del método a todo tipo de estructuras robóticas redundantes.

4.1.1 Robot Planar de Una Articulación

Sea un robot $\mathbf{A}$ formado por un único elemento, $\mathbf{A}_1$, con una articulación de revolución (ver figura 4.1); las posibles posiciones del robot vendrán definidas por el ángulo de giro en un plano de su articulación, θ_1 —único grado de libertad—, por lo que se producirán colisiones en aquellos ángulos para los que,

a lo largo de la longitud (l_1) del elemento, se encuentre algún objeto.

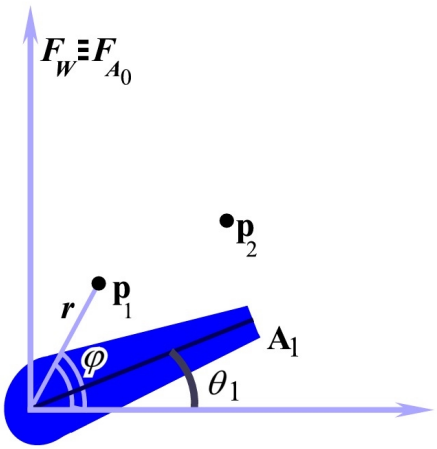


Figura 4.1: Un robot planar con un sólo grado de libertad y dos obstáculos puntuales (p_1 y p_2)

Como ya se ha visto en el formalismo presentado anteriormente, hay que elegir adecuados sistemas de referencia; y unas funciones coordenadas, que son cruciales en el método.

Los sistemas de referencia para el espacio de trabajo, F_W, y para el robot, F_{A_0}, elegidos son los siguientes (figura 4.1):

- F_W situado en la base del robot.
- F_{A_0} coincidente con F_W.

Una configuración $q \in C$ viene parametrizada por θ_1. Para representar un punto $x \in W$, se eligen las coordenadas polares, de forma que x estaría parametrizado por $x = (r, \varphi)$.

La función $B : W \rightarrow R$ estaría definida de la siguiente forma

$$B(r,\varphi) = \begin{cases} 1 & \text{si } (r,\varphi) \in \mathbf{B} \\ 0 & \text{si } (r,\varphi) \notin \mathbf{B} \end{cases} \tag{4.1}$$

Mientras que $A : C \times W \rightarrow R$ vendría dada por

$$A(\theta_1, r, \varphi) = A_1(\theta_1, r, \varphi) = \begin{cases} 1 & \text{si } (r, \varphi) \in \mathbf{A}_1(\theta_1) \\ 0 & \text{si } (r, \varphi) \notin \mathbf{A}_1(\theta_1) \end{cases} \quad (4.2)$$

Al considerar las coordenadas polares para representar al robot y a los obstáculos en el espacio de trabajo, se encuentra una relación entre el recorrido angular del DOF θ_1, y φ, ángulo de las coordenadas polares, lo que permitirá simplificar la evaluación de la función A y, por tanto, de los C-obstáculos. Para ello, se parte de las funciones A_1 y B definidas anteriormente y se sigue el formalismo.

Si se tiene en cuenta que se verifica lo siguiente

$$A_1(\theta_1, r, \varphi) = A_1(0, r, \varphi - \theta_1) \quad (4.3)$$

donde como se muestra en la figura 4.2 un punto del robot en la configuración θ_1, cuyas coordenadas son (r, φ), pasa a tener coordenadas $(r, \varphi - \theta_1)$ en la configuración 0.

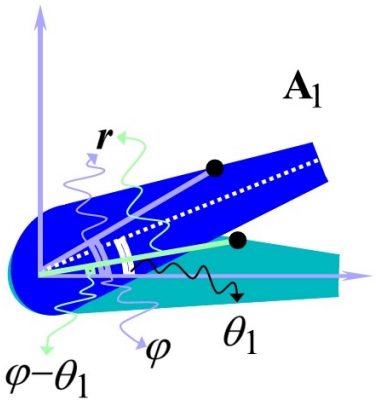


Figura 4.2: Coordenadas de un punto del robot en las configuraciones θ_1 y 0

Por tanto, se observa que se simplifica de forma considerable la evaluación de la función A_1, pues sólo es necesario determinar el valor de esta función (que representa al robot) en la configuración $\theta_1 = 0$, y no para todos los valores de $\theta_1 \in [-\pi, \pi)$.

De esta forma, el robot en una configuración dada se denota

$$A_1(0, r, \varphi - \theta_1) = A_{1_{(0)}}(r, \varphi - \theta_1) \tag{4.4}$$

Y así, se consigue reducir la evaluación del robot a calcular una sola vez el elemento $\mathbf{A}_1$ en la configuración 0, $A_{1_{(0)}}$.

Procediendo del modo en que se ha hecho, la expresión 3.9 para $CB(\theta_1)$, que se corresponde con la expresión 3.10, es decir,

$$CB(\theta_1) = \int A_1(\theta_1, r, \varphi) B(r, \varphi) dr d\varphi \tag{4.5}$$

pasaría a calcularse de esta otra forma

$$CB(\theta_1) = \int A_{1_{(0)}}(r, \varphi - \theta_1) B(r, \varphi) dr d\varphi \tag{4.6}$$

Atendiendo a la definición del producto de convolución, $f(t) * g(t) = \int f(t')g(t - t')dt'$, es necesario introducir una nueva definición

$$\bar{A}_{1_{(0)}}(r, \varphi) = A_{1_{(0)}}(r, -\varphi) \tag{4.7}$$

Con lo que la evaluación del C-espacio vendría caracterizada por el siguiente producto de convolución

$$CB(\theta_1) = \int (\bar{A}_{1_{(0)}} * B)_{\varphi}(r, \theta_1) dr \tag{4.8}$$

donde el subíndice φ indica que el producto de convolución de las funciones $\bar{A}_1$ y B se realiza para todos los valores de la variable $\varphi \in [-\pi, \pi)$.

En este punto, gracias al teorema de convolución, este proceso se simplifica enormemente, convirtiéndose en una multiplicación de las transformadas de Fourier en una dimensión de las funciones A_1 y B. Por tanto,

$$\mathcal{F}[CB(\theta_1)] = \int \mathcal{F}\left[\bar{A}_{1_{(0)}}(r, \theta_1)\right]_{\varphi} \mathcal{F}\left[B(r, \theta_1)\right]_{\varphi} dr \tag{4.9}$$

Este cálculo permite encontrar en qué configuraciones hay colisión, para lo que es necesario multiplicar las transformadas de ambas funciones para todos los ángulos en cada radio, y sumar los resultados a lo largo de toda la longitud de la articulación.

Configuración y obstáculos

Para un mismo ángulo —una configuración en un robot planar con un único DOF— pueden darse varias colisiones simultáneas con diferentes objetos o partes de un objeto, sin embargo, se trata de un solo punto en el C-espacio.

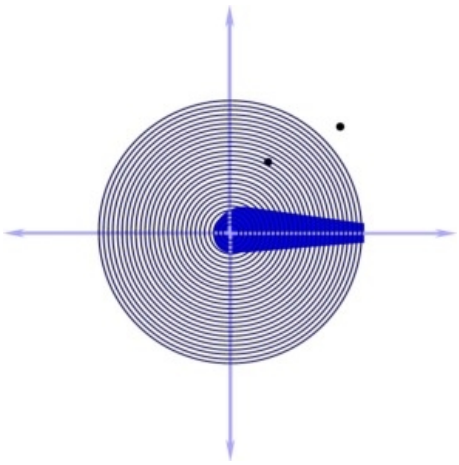


Figura 4.3: Convoluciones acumuladas por radios para un robot planar con un DOF.

Algoritmo

El procedimiento anteriormente expuesto se concreta en el algoritmo de la tabla 4.1, donde, como se puede observar, se hace la evaluación de las posibles colisiones para todos los puntos del único elemento del robot. Así, se considera que éste está en la configuración 0, es decir, $\theta_1 = 0$, y se acumulan los resultados para los puntos que caen en cada radio que se evalúa (ver figura 4.3); esto hará posible la posterior representación de todos los C-obstáculos.

En este punto, para poder computar la expresión 4.9, se hace necesaria una discretización del espacio de trabajo y del espacio de las configuraciones. Esto implica dar nuevas definiciones para las funciones A, B y CB para espacios discretos, que se denotan por A^*, B^* y CB^*. Esto se hace de forma genérica en el apéndice B.

Se debe construir una matriz binaria de pares de coordenadas (r, φ) para el espacio de trabajo, B^*, y, de forma similar, otra para el robot solamente en la configuración $\theta_1 = 0$, $\bar{A}^*_{1_{(0)}}$. Se calcula el producto de dos vectores, la transformada de los puntos del espacio de trabajo para el radio *rad*, y la trans-

formada de $\bar{A}^*_{1_{(0)}}(rad)$ —vector de ángulos—, acumulando el resultado para los N —la resolución de discretización elegida— valores del radio, $[0, l_1]$, donde l_1 es la longitud del único eslabón del robot. Finalmente, sólo queda calcular la transformada inversa del producto acumulado y construir el vector CB^* colocando unos para aquellos índices del vector producto acumulado cuyos valores son mayores que cero.

```
Construir la matriz binaria B*
Calcular F[B*]
Construir la matriz binaria Ā*_1(0)
Hacer ProdAcum = 0
Para cada radio rad
   Calcular F[Ā*_1(0)(rad)]
   Obtener P = F[B*(rad)] · F[Ā*_1(0)(rad)]
   Acumular ProdAcum = ProdAcum + P
Calcular IP = F^-1[ProdAcum]
Para cada θ1 tal que IP(θ1) > 0
   Asignar CB*(θ1) = 1
```

Tabla 4.1: Algoritmo para el cálculo del C-espacio de un robot planar de una articulación

4.1.2 Robot Planar de Dos Articulaciones

Una vez presentado el caso más sencillo, en esta sección se va a aplicar el método matemático a un robot articulado (figura 4.4). Sea $\mathbf{A}$ un robot formado por dos objetos rígidos, $\mathbf{A}_1$ y $\mathbf{A}_2$, que se mueven en R^2 mediante dos articulaciones mecánicas de revolución, es decir, se tienen dos grados de libertad $(\theta_1, \theta_2) \in [-\pi, \pi)$. Al igual que en el caso anterior, como coordenadas de trabajo se eligiran (r, φ) que pueden variar en el intervalo $[0, l_1 + l_2] \times [-\pi, \pi)$, siendo l_1 y l_2 las longitudes de los elementos.

Para este tipo de estructura robótica se debe realizar un paso clave que habilita la deconstrucción del espacio de las

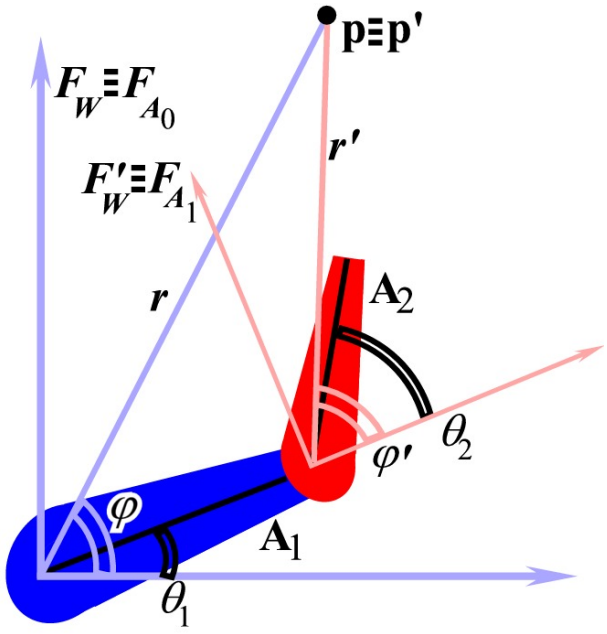


Figura 4.4: Robot planar de dos articulaciónes

configuraciones: un cambio de sistema de referencia. Con ello, es posible tratar cada elemento de la cadena cinemática de forma individual, aplicándole una evaluación del espacio de las configuraciones a cada uno que, en esencia, es similar al caso presentado en la sección anterior.

Este cambio de sistema de referencia se lleva a cabo mediante una transformación homogénea sobre las coordenadas de los puntos del espacio de trabajo.

Elección de los Sistemas de Referencia

Para la elección de los sistemas de referencia se sigue el método clásico de Denavit-Hartenberg [DH55]. Los sistemas de referencia asociados al espacio de trabajo y a los elementos del robot se deben colocar como sigue (figura 4.4):

- F_W situado en la base del robot.

- F_{A_0} coincidente con F_W.
- El origen de F_{A_1} coincide con extremo final del elemento A_1, con un eje situado en la misma dirección que la línea que recorre al primer elemento, mientras que el otro es perpendicular a ésta.

Elección de Funciones Coordenadas y Cálculo de CB

De la elección de las coordenadas apropiadas depende la posibilidad de utilizar la convolución en cada uno de los elementos de la cadena del robot, lo cual constituye uno de los objetivos de la deconstrucción.

De esta manera, lo dicho para el caso de un robot planar de una sola articulación vale para el primer elemento del presente caso, por lo que se puede encontrar una relación entre θ_1 y φ. Por tanto, la mejor opción es tomar un sistema de coordenadas polares.

Como se desarrollará a continuación, la idea de la deconstrucción es encontrar una relación del mismo tipo para el segundo grado de libertad, es decir, θ_2, asociado a la segunda articulación. Esto nos permitirá resolver el conjunto de colisiones, debidas a uno y otro elemento, por separado —gracias a la transformación homogénea— y de forma similar —un producto de transformadas de Fourier.

Entonces, para este caso, aplicando la propiedad de superposición, hay dos conjuntos que calcular

$$\mathbf{CB} = \mathbf{CB}_1 \cup \mathbf{CB}_2 \tag{4.10}$$

lo que significa dos funciones a evaluar (expresión 3.9), que según la expresión 3.10 se haría como sigue

$$CB_1(\theta_1) = \int A_1(\theta_1, r, \varphi) B(r, \varphi) dr d\varphi \tag{4.11}$$

$$CB_2(\theta_1, \theta_2) = \int A_2(\theta_1, \theta_2, r, \varphi) B(r, \varphi) dr d\varphi \tag{4.12}$$

donde, habrá que definir las funciones A_k y B del formalismo particularizadas a este nuevo caso.

La función $B : W \to R$ sería la misma que para el robot de una sola articulación, esto es

$$B(r, \varphi) = \begin{cases} 1 & \text{si } (r, \varphi) \in \mathbf{B} \\ 0 & \text{si } (r, \varphi) \notin \mathbf{B} \end{cases} \tag{4.13}$$

Mientras que las funciones $A_k : C \times W \rightarrow R$, dada la estructura del manipulador, serían:

$$A_1(\theta_1, r, \varphi) = \begin{cases} 1 & \text{si } (r,\varphi) \in \mathbf{A}_1(\theta_1) \\ 0 & \text{si } (r,\varphi) \notin \mathbf{A}_1(\theta_1) \end{cases} \tag{4.14}$$

$$A_2(\theta_1, \theta_2, r, \varphi) = \begin{cases} 1 & \text{si } (r,\varphi) \in \mathbf{A}_2(\theta_1, \theta_2) \\ 0 & \text{si } (r,\varphi) \notin \mathbf{A}_2(\theta_1, \theta_2) \end{cases} \tag{4.15}$$

Uso de la Convolución para el Cálculo de CB_1

Como ya se hizo en el caso anterior, siguiendo el formalismo matemático, el cálculo de $CB_1(\theta_1)$ se simplifica hasta llegar a la expresión 4.9, que se computa mediante el algoritmo de la tabla 4.1.

En este momento se podrían encontrar los puntos de los C-obstáculos debidos al primer elemento, ya que dada una configuración del primer elemento que produzca colisión, $\theta_1 = \gamma$, los puntos $(\gamma, \lambda), \forall \lambda \in \theta_2$, forman parte de un C-obstáculo.

Uso de la Transformación Homogénea para el Cálculo de CB_2. Método de Denavit-Hartenberg

Queda ahora el problema de evaluar la expresión 4.12 que depende de las dos variables de configuración involucradas: θ_1, que define el sistema de referencia F_{A_1}, y θ_2, que define al elemento $\mathbf{A}_2$ sobre él.

Ya se ha dicho que los objetivos de la deconstrucción son dos:

- Poder evaluar cada elemento por separado;
- Y, además, hacerlo para cada elemento de forma similar.

En definitiva, se trata de encontrar una relación entre θ_2 y alguna de las coordenadas el espacio de trabajo, que permita actuar como se ha hecho con el primer elemento. Para esto, y trabajando en coordenadas polares, es necesario elegir unos nuevos sistemas de referencia que permitan encontrar tal relación. Una elección adecuada sería la que aparece en la figura 4.4, donde se establece un nuevo sistema de referencia, F'_W, para W, que coincide con el sistema F_{A_1}.

El efecto buscado al utilizar este nuevo sistema de referencia, según la expresión 3.12, es que el elemento $\mathbf{A}_2$ sólo dependa

del grado de libertad asociado a él, θ_2, y no dependa de θ_1. Sin embargo, esto conlleva una transformación de los puntos del espacio de trabajo desde el sistema de referencia F_W al nuevo sistema F'_W.

El problema que nos ocupa (figura 4.4) es determinar las coordenadas (r', φ') de un punto (p') en el espacio de trabajo respecto al sistema de referencia F'_W —o, lo que es lo mismo, F_{A_1}— a partir de las coordenadas (r, φ) del mismo punto (p), pero respecto al sistema de referencia original, F_W —o el de la base del robot, F_{A_0}—. Para este cambio de coordenadas, se recurre al método de D-H, tal y como se explicó en del capítulo anterior. Esto permite incorporar un método completamente conocido y aceptado por la comunidad científica al proceso de deconstrucción del espacio de las configuraciones.

Este método clásico, recogido en el apéndice A, indica cómo escoger adecuadamente los sistemas de referencia asociados a cada eslabón, para poder así pasar de uno al siguiente mediante cuatro transformaciones básicas, que dependen exclusivamente de las características geométricas y cinemáticas del eslabón. Este método permite reducir el problema cinemático directo a encontrar una matriz de transformación homogénea 4×4 que relaciona la ubicación de cada elemento del robot con respecto al sistema de referencia de su base.

Aplicando el método de D-H, la colocación de los ejes es la de la figura 4.5, y los parámetros los de la tabla 4.2. Así, la relación entre el par de sistemas de referencia viene dada por la siguiente matriz de transformación homogénea[2].

$$ {}^{0}_{1}\mathbf{T} = \begin{bmatrix} C\theta_1 & -S\theta_1 & 0 & l_1 C\theta_1 \\ S\theta_1 & C\theta_1 & 0 & l_1 S\theta_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Por tanto, si se quiere calcular los valores (r', φ') —el punto p'—, a partir de (r, φ) —p—, es necesario calcular la inversa de ${}^{0}_{1}\mathbf{T}$, ya que la matriz original relaciona puntos definidos en el sistema de referencia 1 con el 0. Es decir,

$$ p = {}^{0}_{1}\mathbf{T} \cdot p' \Rightarrow p' = {}^{0}_{1}\mathbf{T}^{-1} \cdot p $$

[2] La notación habitual para esta matriz se denota por $\mathbf{A}$, en el presente trabajo se ha optado por usar la letra $\mathbf{T}$, para evitar confusiones con el robot o sus elementos.

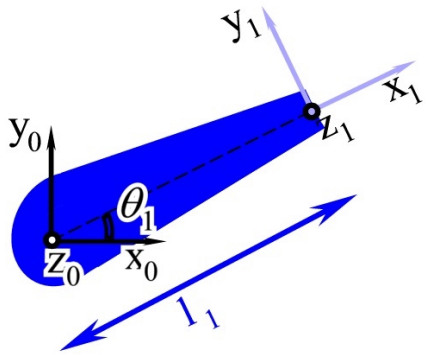


Figura 4.5: Ejes de Denavit-Hartenberg para un robot planar

Eslabón	θ_i	a_i	d_i	α_i
1	θ_1	l_1	0	0

Tabla 4.2: Parámetros de Denavit-Hartenberg para un manipulador planar

Haciendo la inversa, las nuevas coordenadas se calcularían como sigue

$$p' = {}^0_1\mathbf{T}^{-1} \cdot p = \begin{bmatrix} C\theta_1 & S\theta_1 & -l_1 \\ -S\theta_1 & C\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Hay que tener en cuenta que el método de D-H[3] trabaja con sistemas de coordenadas cartesianas. Como ya se ha comentado las coordenadas más adecuadas para expresar un punto en W son las coordenadas polares, por lo que habrá que hacer las conversiones pertinentes.

[3]Por otro lado, el lector puede pensar que utilizar el procedimiento de D-H, para calcular esta transformación de coordenadas es excesivo, pues es un cálculo trigonométrico sencillo. Es cierto, pero sólo para esta estructura robótica; para casos más complejos, como los que se estudiarán en las secciones posteriores, aplicar este método es muy aconsejable, además de dotar a la deconstrucción de la generalidad perseguida.

Realizando el cambio de funciones coordenadas, de cartesaianas a polares, el resultado sería el siguiente.

$$\begin{bmatrix} r'C\varphi' \\ r'S\varphi' \\ 1 \end{bmatrix} = \begin{bmatrix} rC\varphi C\theta_1 + rS\varphi S\theta - l_1 \\ -rC\varphi S\theta_1 + rS\varphi C\theta_1 \\ 1 \end{bmatrix}$$

por lo que, simplificando, quedarían como se expresa a continuación.

$$r' = \sqrt{r^2 + l_1^2 - 2rl_1C(\varphi - \theta_1)}$$

$$\varphi' = artg\left(\frac{-rS(\theta_1 - \varphi)}{rC(\theta_1 - \varphi) - l_1}\right)$$

Finalmente, se puede observar que, hay que realizar una transformación de los obstáculos y el robot sobre este nuevo sistema de referencia; así, se tienen las siguientes consecuencias:

- Se tiene una nueva función para describir los obstáculos en el espacio de trabajo utilizando el nuevo sistema de referencia, que se denominará B'. Los valores de r' y φ' dependen de los parámetros de D-H asociados al primer eslabón: l_1 y θ_1.
- Hay que definir una nueva función para describir el segundo elemento del planar, A'_2 (que tendrá una definición análoga a A'_1). Ésta no depende de ningún parámetro de D-H asociado al eslabón anterior, sólo depende de θ_2.

Con el cambio de sistemas de referencia introducido, ahora la expresión 4.12 se puede evaluar como

$$CB_2(\theta_1, \theta_2) = \int A'_2(\theta_2, r', \varphi')B'(r', \varphi')dr'd\varphi' \qquad (4.16)$$

Uso de la Convolución para el cálculo de CB_2

Como se puede observar, de nuevo la expresión 4.16 es similar a 4.11, con lo que se puede evaluar de la misma forma (expresión 4.9) y computar a través del mismo algoritmo (tabla 4.1). Es decir, gracias a la relación que se encuentra entre θ_2 y φ', la expresión 4.16 puede ponerse como

$$CB_2(\theta_1, \theta_2) = \int A'_{2_{(0)}}(r', \varphi' - \theta_2)B'(r', \varphi')dr'd\varphi' \qquad (4.17)$$

donde, introduciendo de nuevo la definición,

$$\bar{A}'_{2_{(0)}}(r', \varphi') = A'_{2_{(0)}}(r', -\varphi') \tag{4.18}$$

se encuentra el siguiente producto de convolución

$$CB_2(\theta_1, \theta_2) = \int (\bar{A}'_{2_{(0)}} * B')_{\varphi'}(r', \theta_2) dr' \tag{4.19}$$

que se simplifica, aplicando el teorema de convolución, para llegar a la expresión final:

$$\mathcal{F}\left[CB_2(\theta_1, \theta_2)\right] = \int \mathcal{F}\left[\bar{A}'_{2_{(0)}}(r', \theta_2)\right]_{\varphi'} \mathcal{F}\left[B'(r', \theta_2)\right]_{\varphi'} dr' \tag{4.20}$$

Algoritmo

Al igual que en el caso anterior, para poder computar los obstáculos en el espacio de las configuraciones asociados a cada uno de los elementos, es necesaria una discretización (ver apéndice B). Una vez hecho esto —y como la deconstrucción pretende—, el algoritmo consiste en repetir, un número indeterminado de veces, la acumulación de los productos de convolución que ilustra la figura 4.3.

En la figura 4.6 se representa el proceso completo de deconstrucción para un robot planar de dos elementos. Los discos de color gris representan una acumulación de productos de convolución —similar al concepto ilustrado en 4.3—, es decir, la evaluación de la expresión 4.9 (o alguna equivalente, como la expresión 4.20), mientras que los discos de puntos simbolizan que estos son cálculos potenciales. El C-espacio estará ocupado por los C-obstáculos debidos al primer elemento, $\mathbf{CB}_1$, evaluados como muuestra la figura 4.6.a, y los C-obstáculos debidos al segundo elemento, $\mathbf{CB}_2$, evaluados como muestra la figura 4.6.b.

Una vez calculada la parte correspondiente a $\mathbf{CB}_1$, se conocen cuáles son las configuraciones del primer elemento que producen colisión, y se puede construir parte de la matriz del C-espacio, haciendo $CB^*(\gamma, \lambda) = 1$, para cada valor $\theta_1 = \gamma$ que produzca colisión, siendo λ todos los valores de θ_2.

El siguiente paso es calcular la parte correspondiente a $\mathbf{CB}_2$. Esto se hace transformando el espacio de trabajo respecto del sistema de referencia del primer elemento para cada

Construir la matriz binaria B^*
Construir la matriz binaria $\bar{A}^*_{1_{(0)}}$
Construir la matriz binaria $\bar{A}'^*_{2_{(0)}}$
Para cada radio rad
  Calcular $\mathcal{F}\left[\bar{A}^*_{1(0)}(rad)\right]$
  Calcular $\mathcal{F}\left[\bar{A}^*_{2(0)}(rad)\right]$
Calcular $\mathcal{F}[B^*]$
Hacer $ProdAcum = 0$
Para cada radio rad
  Obtener $P = \mathcal{F}[B^*(rad)] \cdot \mathcal{F}\left[\bar{A}^*_{1(0)}(rad)\right]$
  Acumular $ProdAcum = ProdAcum + P$
Calcular $IP = \mathcal{F}^{-1}[ProdAcum]$
Para cada θ_1 tal que $IP(\theta_1) > 0$
  Para cada θ_2
    Asignar $CB^*(\theta_1, \theta_2) = 1$
Para cada θ_1 tal que $IP(\theta_1) = 0$
  Transformar B^* en B'^*
  Calcular $\mathcal{F}\left[B'^*\right]$
  Hacer $ProdAcum = 0$
  Para cada radio rad
    Obtener $P = \mathcal{F}\left[B'^*(rad)\right] \cdot \mathcal{F}\left[\bar{A}'^*_{2(0)}(rad)\right]$
    Acumular $ProdAcum = ProdAcum + P$
  Calcular $IP2 = \mathcal{F}^{-1}[ProdAcum]$
  Para cada θ_2 tal que $IP2(\theta_2) > 0$
    Asignar $CB^*(\theta_1, \theta_2) = 1$

Tabla 4.3: Algoritmo para el cálculo del C-espacio de un robot planar de dos articulaciones

configuración libre. Por lo tanto, si se ha utilizado una resolución N en la discretización, en el peor de los casos, habría que realizar N cálculos similares al que se hizo para el primer elemento; esto corresponde al caso en que no exista ningún obstáculo para el primer eslabón. Cuando hay obstáculos para el primer elemento, el número de veces que se debe transfor-

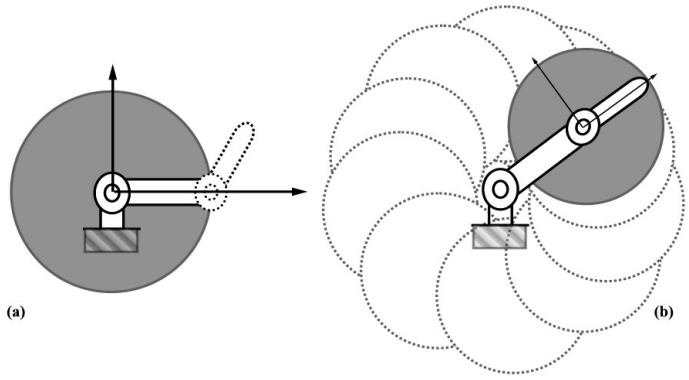


Figura 4.6: Deconstrucción del C-espacio para un robot planar de dos articulaciones: a) cálculo asociado a $\mathbf{CB}_1$ y b) cálculo asociado a $\mathbf{CB}_2$

mar el espacio de trabajo y acumular productos de convolución, será $N - t$, siendo t el número de valores de θ_1 que producen colisión.

Por otro lado, conviene resaltar que la matriz binaria $A'^{*}_{2_{(0)}}$ sólo es necesaria construirla una vez, ya que no depende de θ_1. Para cada iteración del bucle en θ_1, se trata siempre de la discretización del segundo elemento del robot para un valor de la variable de configuración $\theta_2 = 0$ respecto a un sistema de referencia colocado en su base. Asimismo, obsérvese que las transformadas del vector de ángulos $A'^{*}_{2_{(0)}}$ para cada radio, sólo es necesario calcularlas una vez.

Finalmente, se han recuadrado en la tabla 4.3 las dos partes del algoritmo que consisten en realizar exactamente las mismas operaciones, pero sobre diferentes datos.

La Deconstrucción Gráficamente

Como última observación, cabe destacar que el proceso de deconstrucción que se ha realizado se desprende del análisis y explotación de la estructura del robot, es decir, de deshacer analíticamente los elementos que lo constituyen. En la figura 4.7 se muestra cómo el robot planar se descompone en dos

elementos iguales para los que se aplica el mismo proceso de evaluación del C-espacio.

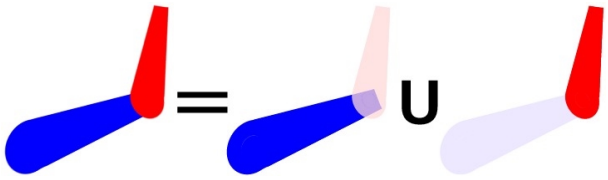


Figura 4.7: Deconstrucción de la estructura de un robot planar

4.1.3 Robot Planar de Tres Articulaciones. Redundancia

Se llega en este punto a una de las aportaciones más interesantes de este trabajo: la aplicación del formalismo matemático para el caso de un robot redundante.

Considérese el robot articulado, $\mathbf{A}$, de la figura 4.8, formado por tres objetos rígidos, $\mathbf{A}_1$, $\mathbf{A}_2$ y $\mathbf{A}_3$, que se mueven en R^2 mediante tres articulaciones mecánicas de revolución. Por tanto —aparece en este caso la redundancia—, se tienen tres grados de libertad $(\theta_1, \theta_2, \theta_3) \in [-\pi, \pi)$. Como se verá, la elección de las funciones coordenadas sigue siendo (r, φ), que pueden variar en el intervalo $[0, l_1 + l_2 + l_3] \times [-\pi, \pi)$, donde l_1, l_2 y l_3 son las longitudes del primer, segundo y tercer elemento, respectivamente.

Se muestra a continuación la potencia de la deconstrucción aplicada a robots redundantes. En el desarrollo se va a tener en cuenta que el robot se puede descomponer como muestra la figura 4.9, que se corresponde con los dos casos estudiados previamente.

Elección de los Sistemas de Referencia

De forma análoga al caso del robot planar, se sigue el procedimiento de Denavit-Hartenberg para elegir los sistemas de referencia para el espacio de trabajo y para el robot y sus elementos (figura 4.8).

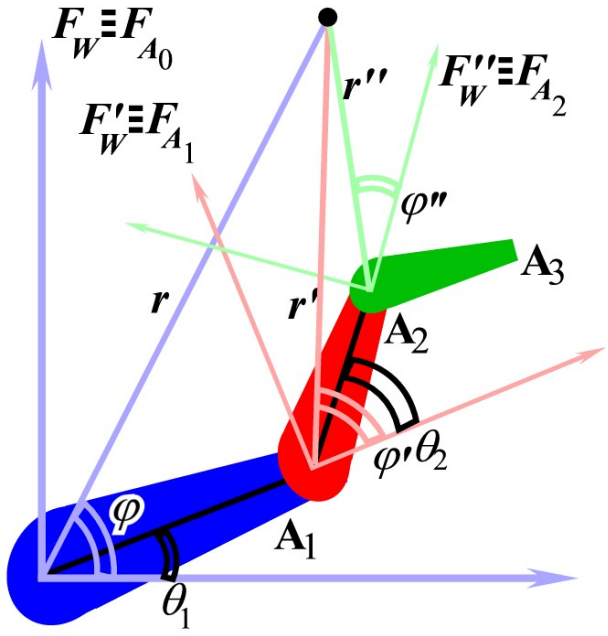


Figura 4.8: Robot planar de tres articulaciónes

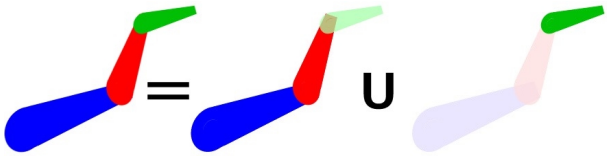


Figura 4.9: Deconstrucción de la estructura de un robot planar redundante

- F_W situado en la base del robot.
- F_{A_0} coincidente F_W.
- El origen de F_{A_1} coincide con extremo final del elemento A_1, con un eje situado en la misma dirección que la línea

que recorre al primer elemento, mientras que el otro es perpendicular a ésta.

- El origen de F_{A_2} coincide con extremo final del elemento A_2, con un eje situado en la misma dirección que la línea que recorre al segundo elemento, mientras que el otro es perpendicular a ésta.

Elección de Funciones Coordenadas y Cálculo de CB

De nuevo, el análisis realizado para el caso de un robot planar de una sola articulación vale para la primera articulación del presente caso, por lo que se puede encontrar una relación entre θ_1 y φ. Además, al igual que para el robot planar de dos articulaciones, se encuentra para el segundo elemento del robot una relación similar, entre θ_2 y φ'.

Siguiendo la idea de la deconstrucción, para poder resolver por separado el conjunto de colisiones asociado a cada uno de los tres eslabones de la cadena, se procede como se ha visto para el robot planar con los dos primeros elementos del robot, es decir, utilizando las coordenadas polares se permite introducir el teorema de convolución en la evaluación de cada función CB_k. Por tanto, para poder aplicar las mismas ideas al tercer eslabón de la cadena, hay que encontrar una relación del mismo tipo para el tercer grado de libertad, es decir, θ_3, asociado a la tercera articulación.

En el presente caso, gracias a la deconstrucción, tres serán los conjuntos que se deben calcular

$$\mathbf{CB} = \mathbf{CB}_1 \cup \mathbf{CB}_2 \cup \mathbf{CB}_3 \tag{4.21}$$

Y así, se tienen tres funciones a evaluar (expresión 3.9), lo que según la expresión 3.10 se haría como ya se ha comentado anteriormente para CB_1 y CB_2, y para la tercera como sigue

$$CB_3(\theta_1, \theta_2, \theta_3) = \int A_3(\theta_1, \theta_2, \theta_3, r, \varphi) B(r, \varphi) dr d\varphi \tag{4.22}$$

donde, de nuevo, se definen las funciones A_k y B del formalismo, particularizadas a este caso. La función B sería la misma que los dos casos anteriores (expresiones 4.1 y 4.13) y $A_3 : C \times W \to R$

$$A_3(\theta_1,\theta_2,\theta_3,r,\varphi) = \begin{cases} 1 & \text{si } (r,\varphi) \in \mathbf{A}_3(\theta_1,\theta_2,\theta_3) \\ 0 & \text{si } (r,\varphi) \notin \mathbf{A}_3(\theta_1,\theta_2,\theta_3) \end{cases} \quad (4.23)$$

Uso de la Transformación Homogénea (Método de D-H) para el Cálculo de CB_3

En esta sección es donde se realizan los cambios de sistemas de referencia que habilitan la deconstrucción. Se trata de realizar una transformación similar a como se hace con CB_2 —para la que se procede exactamente igual que en el caso de un robot planar de dos articulaciones—. Sin embargo, para la expresión 4.22 el problema que se presenta es más complicado.

En este caso los DOFs θ_1 y θ_2 definen el sistema de referencia F_{A_2}, mientras que lo que define al elemento $\mathbf{A}_3$ respecto a él es θ_3.

Se trata de utilizar un sistema de referencia que permita hacer que la función A_3 no dependa de ninguno de los parámetros asociados a los dos eslabones anteriores. Es decir, se trata de encontrar una relación entre θ_3 y una de las coordenadas del espacio de trabajo. El problema es similar al visto para el segundo elemento de un robot planar. Se trabaja en coordenadas polares y es necesario elegir unos sistemas de referencia que permitan encontrar esta relación. La figura 4.8 muestra esta elección, donde se establece un nuevo sistema de referencia para W, F''_W, que coincide con F_{A_2}.

Entonces, en este caso se tendrán que transformar los puntos del espacio de trabajo, respecto al sistema de referencia original, F_W, respecto al nuevo sistema, F''_W, esto es, pasar de las coordenadas (r,φ) a las coordenadas (r'',φ'').

La figura 4.10 muestra los parámetros de D-H (ver apéndice A, donde se explica cómo elegir los ejes), que además se recogen en la tabla 4.4. Así, la relación entre los sistemas de referencia viene dada por las siguientes matrices de transformación homogénea.

$${}^0_1\mathbf{T} = \begin{bmatrix} C\theta_1 & -S\theta_1 & l_1C\theta_1 \\ S\theta_1 & C\theta_1 & l_1S\theta_1 \\ 0 & 0 & 1 \end{bmatrix} \quad {}^1_2\mathbf{T} = \begin{bmatrix} C\theta_2 & -S\theta_2 & l_2C\theta_2 \\ S\theta_2 & C\theta_2 & l_2S\theta_2 \\ 0 & 0 & 1 \end{bmatrix}$$

Teniendo en cuenta que ${}^0_2\mathbf{T} = {}^0_1\mathbf{T}\,{}^1_2\mathbf{T}$, y que la nuevas coordendas del punto serán (r'',φ''), dado que $p'' = {}^0_2\mathbf{T}^{-1}\cdot p$, enton-

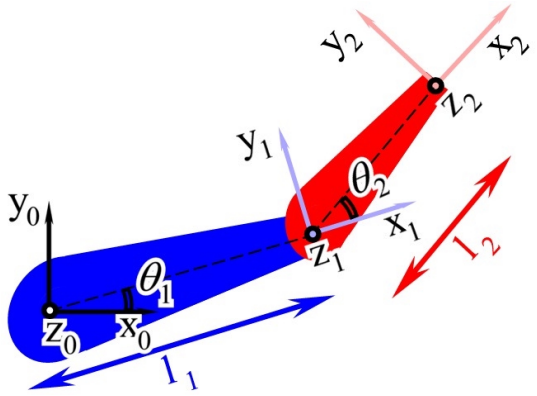


Figura 4.10: Ejes de Denavit-Hartenberg para un manipulador planar de tres articulaciones

Eslabón	θ_i	a_i	d_i	α_i
1	θ_1	l_1	0	0
2	θ_2	l_2	0	0

Tabla 4.4: Parámetros de Denavit-Hartenberg para un manipulador planar de tres articulaciones

ces, a través de los cálculos apropiados[4], se llega a las siguientes expresiones:

$$r''=\sqrt{r^2-2rl_2C(\theta_1+\theta_2-\varphi)-2rl_1C(\theta_1-\varphi)+2l_1l_2C\theta_2+l_1^2+l_2^2}$$

$$\varphi''=artg\left(\frac{l_1S\theta_2-rS(\theta_1+\theta_2-\varphi)}{rC(\theta_1+\theta_2-\varphi)-l_1C\theta_2-l_2}\right)$$

No debe olvidarse que el método de D-H trabaja en coordendas cartesianas, por lo que son necesarias una serie de cambios de coordenadas para, partiendo de un punto p dado en coordenadas polares, llegar a sus coordenadas transformadas, también polares.

[4] En el trabajo de investigación que se documenta, todos estos cálculos, y los de las demás secciones, han sido llevados a cabo mediante la *toolbox* de cálculo simbólico de MATLAB©, de The Mathworks.

Uso de la Convolución para el Cálculo de CB_3

De forma análoga al caso anterior, con el cambio de sistema de referencia introducido, a partir de 4.22, se llega a la siguiente expresión para calcular las colisiones asociadas al tercer elemento del brazo planar redundante.

$$\mathcal{F}\left[CB_3(\theta_1,\theta_2,\theta_3)\right] = \int \mathcal{F}\left[\bar{A}'_{3_{(0)}}(r'',\theta_3)\right]_{\varphi''} \mathcal{F}\left[B''(r'',\theta_3)\right]_{\varphi''} dr'' \tag{4.24}$$

Recordando, el cálculo de $CB_1(\theta_1)$ se simplifica hasta llegar a la expresión 4.9, y el de $CB_2(\theta_1,\theta_2)$ hasta la expresión 4.20. Todas estas expresiones (4.9, 4.20 y 4.24) se computan individualmente mediante el algoritmo de la tabla 4.1.

Algoritmo

Al igual que en el caso anterior, para poder computar los obstáculos en el espacio de las configuraciones asociados a cada uno de los elementos, es necesaria una discretización (ver apéndice B). El algoritmo se detalla en las tablas 4.5 y 4.6, donde, como se explica a continuación, se logra una deconstrucción óptima pues se repite, en tres ocasiones —las zonas recuadradas del algoritmo—, la acumulación de los productos de convolución que ilustra la figura 4.3.

En la figura 4.11 se representa el proceso completo de deconstrucción del espacio de las configuraciones para un robot planar de tres articulaciones. Como hasta ahora, los discos de color gris representan una acumulación de productos de convolución, es decir, la evaluación de las expresiones 4.9, 4.20 o 4.24, mientras que los discos de puntos y la zona gris claro simbolizan cálculos que será necesario hacer o no en función de las configuraciones libres para los eslabones anteriores de la cadena.

El C-espacio estará ocupado por los C-obstáculos debidos al primer elemento, $\mathbf{CB}_1$, los debidos al segundo elemento, $\mathbf{CB}_2$ y los debidos al tercer elemento, $\mathbf{CB}_3$, que se evalúan tal y como se muestran en las figuras 4.11.a, 4.11.b y 4.11.c, respectivamente.

Al igual que para el planar de dos articulaciones, una vez calculada la parte correspondiente a $\mathbf{CB}_1$, se conocen cuáles

```
Construir la matriz binaria B*
Construir la matriz binaria Ā*_{1(0)}
Construir la matriz binaria Ā'*_{2(0)}
Construir la matriz binaria Ā''*_{3(0)}
Para cada radio rad
   Calcular F[Ā*_{1(0)}(rad)]
   Calcular F[Ā'*_{2(0)}(rad)]
   Calcular F[Ā''*_{3(0)}(rad)]
Calcular F[B*]
Hacer ProdAcum = 0
Para cada radio rad
   Obtener P = F[B*(rad)] · F[Ā*_{1(0)}(rad)]
   Acumular ProdAcum = ProdAcum + P
 Calcular IP = F^{-1}[ProdAcum]
Para cada θ1 tal que IP(θ1) > 0
   Para cada θ2
      Para cada θ3
         Asignar CB*(θ1, θ2, θ3) = 1
```

Tabla 4.5: Algoritmo para el cálculo del C-espacio de un robot planar de tres articulaciones (1^a parte)

son las configuraciones del primer elemento que producen colisión, y se puede construir parte de la matriz del C-espacio, pero en este caso se debe hacer $CB^*(\gamma, \lambda, \sigma) = 1$, para cada valor $\theta_1 = \gamma$ que produzca colisión, siendo λ todos los valores de θ_2 y σ todos los valores de θ_3. Similarmente, cuando se conocen las colisiones asociadas al segundo eslabón, se debe hacer $CB^*(\gamma, \lambda, \sigma) = 1$, para cada combinación ($\theta_1 = \gamma$, $\theta_2 = \lambda$) que produzca colisión, siendo σ todos los valores de θ_3.

Para la estructura robótica en estudio, utilizando una resolución N en la discretización, en el peor de los casos, habría que realizar 1 cálculo similar al que se hizo para el primer elemento, para cada par de valores de configuración (θ_1, θ_2); esto

```
Para cada θ1  tal que IP1(θ1) = 0
    Transformar B*  en B'*
    Calcular F[B'*]
    Hacer ProdAcum = 0
    Para cada radio rad
        Obtener P = F[B'*(rad)] · F[Ā'*2(0)(rad)]
        Acumular ProdAcum = ProdAcum + P
    Calcular IP2 = F^-1[ProdAcum]
    Para cada θ2  tal que IP2(θ2) > 0
        Para cada θ3
            Asignar CB*(θ1, θ2, θ3) = 1
    Para cada θ2  tal que IP2(θ2) = 0
        Transformar B*  en B''*
        Calcular F[B''*]
        Hacer ProdAcum = 0
        Para cada radio rad
            Obtener P = F[B''*(rad)] · F[Ā''*3(0)(rad)]
            AcumularProdAcum = ProdAcum + P
        Calcular IP3 = F^-1[ProdAcum]
        Para cada θ3  tal que IP3(θ3) > 0
            Asignar CB*(θ1, θ2, θ3) = 1
```

Tabla 4.6: Algoritmo para el cálculo del C-espacio de un robot planar de tres articulaciones (2^a parte)

corresponde al caso en que no exista ningún obstáculo ni para el primer eslabón, ni para el segundo.

Por otro lado, si se llama t al número de valores de θ_1 que producen colisión con el primer elemento, y p al número de pares (θ_1, θ_2) que producen colisión sólo con el segundo eslabón, entonces el número total de acumulaciones de productos de convolución que hay que realizar es $1 + (N - t) + (N^2 - p)$; además, el número de veces que se debe transformar el espacio de trabajo de B^* a $B^{'*}$ es $(N - t)$, y, de B^* a $B^{''*}$, $(N^2 - p)$

veces.

Falta añadir que las matrices binarias $A'^{*}_{2_{(0)}}$ y $A'^{*}_{3_{(0)}}$ sólo se construyen una vez, ya que $A'^{*}_{2_{(0)}}$ no depende de θ_1, y $A'^{*}_{3_{(0)}}$ no depende de θ_1 ni de θ_2. Asimismo, las transformadas de los vectores de ángulos para cada radio de estas dos matrices, también se calculan una sola vez.

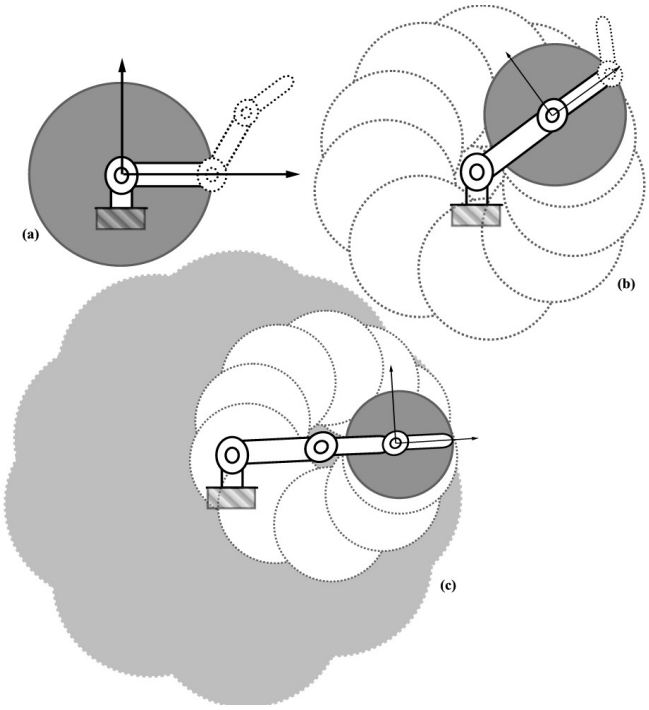


Figura 4.11: Deconstrucción del C-espacio para un robot planar de tres articulaciónes: a) cálculo asociado a $\mathbf{CB}_1$; b) cálculo asociado a $\mathbf{CB}_2$; y c) cálculo asociado a $\mathbf{CB}_3$

4.1.4 Robot PUMA de Tres Articulaciones

Hasta ahora sólo se han considerado robots en el plano. A continuación se expone cómo se aplicaría la deconstrucción a

una estructura robótica que trabaja en un espacio 3D: un robot PUMA (figura 4.12).

Considérese el robot articulado, $\mathbf{A}$, formado por tres objetos rígidos, $\mathbf{A}_1$, $\mathbf{A}_2$ y $\mathbf{A}_3$, que se mueven en R^3 mediante tres articulaciones mecánicas de revolución. Nuevamente, se tienen tres grados de libertad $(\theta_1, \theta_2, \theta_3) \in [-\pi, \pi)$. Este tipo de robot articulado se utiliza ampliamante en la industria, y sus articulaciones reciben los siguientes nombres: cintura (θ_1), hombro (θ_2) y codo (θ_3).

Dado que ahora se trabaja en un entorno de 3 dimensiones, la mejor opción, como se explicará a continuación, es elegir las coordenadas esféricas, $(r, \varphi_1, \varphi_2)$, que pueden variar en el intervalo $[0, l_2 + l_3] \times [-\pi, \pi) \times \left[\frac{-\pi}{2}, \frac{\pi}{2}\right)$, con l_2 y l_3, las longitudes del segundo y tercer elemento, respectivamente.

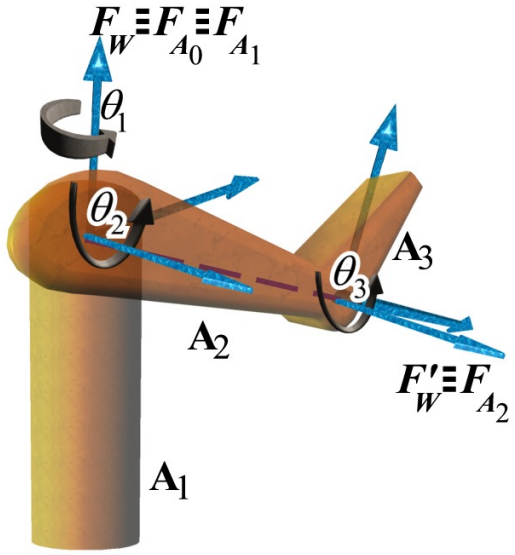


Figura 4.12: Un robot PUMA en un entorno de trabajo 3D

Elección de los Sistemas de Referencia

En este caso es importante resaltar que, con el objetivo de obtener ciertas simetrías que simplificarán el cálculo de los

C-obstáculos, los sistemas de referencia F_W y F_{A_0} se eligen —siempre siguiendo el método de D-H— de manera que sus orígenes estén situados en el punto de intersección de los dos elementos[5] $\mathbf{A}_1$ y $\mathbf{A}_2$. En concreto, se han situado los sistemas de referencia de la siguiente forma (figura 4.12):

Obviar la cintura

El hecho de situar los sistemas de referencia del robot y del espacio de trabajo en la articulación del hombro tiene dos consecuencias, se ignoran las colisiones con la cintura y se considera que el hombro absorve el movimiento de ésta. Es decir, se asocian al segundo elemento dos grados de libertad.

- Los tres sistemas de referencia F_W, F_{A_0} y F_{A_1}, asociados respectivamente con el espacio de trabajo W, al robot, y al primer elemento $\mathbf{A}_1$, son equivalentes. El origen está en el punto en que intersectan los ejes de giro de los dos primeros elementos. Un eje se corresponde con el eje de giro de la cintura y el otro con el eje de giro del hombro. El tercer eje se elige para que forme un triedo con estos.
- El origen de F_{A_2} se coloca al final del segundo elemento, en el punto de articulación del codo. Un eje se coloca siguiendo la línea que une las articulaciones de los elementos $\mathbf{A}_2$ y $\mathbf{A}_3$; otro se corresponde con el eje de giro del codo; y el tercero es perpendicular a los otros dos.

Se puede observar en la figura 4.13 que la estructura del robot cuando se explota, en un proceso deconstructivo, el todo en sus partes, consiste en dos elementos: $\mathbf{A}_2$, que se mueve en un espacio tridimensional, y $\mathbf{A}_3$ que se mueve en un plano —determinado por los valores θ_1 y θ_2—, y que constituye un caso ya estudiado en secciones anteriores.

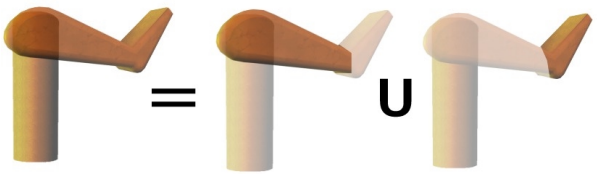


Figura 4.13: Deconstrucción de la estructura de un robot PUMA

[5] Esta elección peculiar está justificada: por un lado, las colisiones entre elementos no tienen excesivo interés, pues siempre son las mismas y pueden ser precalculadas; y, por otro, se ignoran las colisiones del primer elemento con el entorno, ya que los robots se deben ubicar de forma que éstas nunca se produzcan, sino no se podrían mover.

Elección de Funciones Coordenadas y cálculo de CB

Es éste un punto interesante, pues, gracias a la elección de la coordenadas —y a la especial elección de los sistemas de referencia—, se introduce una gran simplificación. Hasta ahora sólo era posible encontrar una relación entre un grado de libertad y una variable espacial cada vez. Como se verá, gracias a esta nueva elección, es posible encontrar dos relaciones entre variables de configuración y variables espaciales, con las ventajas que esto conlleva.

Efectivamente, los grados de libertad asociados al segundo elemento son los dos ángulos de giro en el espacio tridimensional, por lo que la elección de las coordenadas esféricas $(r, \varphi_1, \varphi_2)$ se presenta como una opción ideal ya que θ_1 se relaciona con φ_1 y θ_2 con φ_2.

Siguiendo la idea de la deconstrucción, para poder resolver por separado el conjunto de colisiones asociado a cada uno de los tres eslabones de la cadena, esta vez habrá que econtrar una relación del mismo tipo para el tercer grado de libertad, es decir, θ_3, asociado a la tercera articulación. Por tanto, tres conjuntos conforman el espacio de las configuraciones para un robot PUMA.

$$\mathbf{CB} = \mathbf{CB}_1 \cup \mathbf{CB}_2 \cup \mathbf{CB}_3 \tag{4.25}$$

y así, para este caso, se tienen tres funciones (expresión 3.9) a evaluar, lo que según la expresión 3.10 se haría como sigue

$$CB_1(\theta_1) = \int A_1(\theta_1, r, \varphi_1, \varphi_2) B(r, \varphi_1, \varphi_2) dr d\varphi_1 d\varphi_2 \tag{4.26}$$

$$CB_2(\theta_1, \theta_2) = \int A_2(\theta_1, \theta_2, r, \varphi_1, \varphi_2) B(r, \varphi_1, \varphi_2) dr d\varphi_1 d\varphi_2 \tag{4.27}$$

$$CB_3(\theta_1, \theta_2, \theta_3) = \int A_3(\theta_1, \theta_2, \theta_3, r, \varphi_1, \varphi_2) B(r, \varphi_1, \varphi_2) dr d\varphi_1 d\varphi_2 \tag{4.28}$$

donde las funciones A_k y B del formalismo se particularizan a un robot PUMA con una parametrización de $q = (\theta_1, \theta_2, \theta_3)$ y $x = (r, \varphi_1, \varphi_2)$.

De esta manera, la función $B : W \rightarrow R$ vendría definida por

$$B(r, \varphi_1, \varphi_2) = \begin{cases} 1 & \text{si } (r, \varphi_1, \varphi_2) \in \mathbf{B} \\ 0 & \text{si } (r, \varphi_1, \varphi_2) \notin \mathbf{B} \end{cases} \tag{4.29}$$

Mientras que las $A_k : C \times W \rightarrow R$ serían:

$$A_1(\theta_1, r, \varphi_1, \varphi_2) = \begin{cases} 1 & \text{si } (r, \varphi_1, \varphi_2) \in \mathbf{A}_1(\theta_1) \\ 0 & \text{si } (r, \varphi_1, \varphi_2) \notin \mathbf{A}_1(\theta_1) \end{cases} \tag{4.30}$$

$$A_2(\theta_1, \theta_2, r, \varphi_1, \varphi_2) = \begin{cases} 1 & \text{si } (r, \varphi_1, \varphi_2) \in \mathbf{A}_2(\theta_1, \theta_2) \\ 0 & \text{si } (r, \varphi_1, \varphi_2) \notin \mathbf{A}_2(\theta_1, \theta_2) \end{cases} \tag{4.31}$$

$$A_3(\theta_1, \theta_2, \theta_3, r, \varphi_1, \varphi_2) = \begin{cases} 1 & \text{si } (r, \varphi_1, \varphi_2) \in \mathbf{A}_3(\theta_1, \theta_2, \theta_3) \\ 0 & \text{si } (r, \varphi_1, \varphi_2) \notin \mathbf{A}_3(\theta_1, \theta_2, \theta_3) \end{cases} \tag{4.32}$$

Es importante el hecho de que no es posible una colisión entre el elemento $\mathbf{A_1}$ y algún obstáculo en W, ya que en otro caso el robot no podría moverse. Entonces, debido a esta consideración, la expresión 4.26 es nula. En los siguientes pasos se explica cómo se evalúan las CB restantes.

Uso de la Convolución para el Cálculo de CB_2

Teniendo en cuenta la relación, comentada previamante, de φ_1 con θ_1 y de φ_2 con θ_2, se puede llegar a una expresión análoga a la 4.3.

$$A_2(\theta_1, \theta_2, r, \varphi_1, \varphi_2) = A_2(0, 0, r, \varphi_1 - \theta_1, \varphi_2 - \theta_2) \tag{4.33}$$

Y de forma coherente con la notación elegida en aquel caso, el elemento en una configuración dada se denota

$$A_2(0, 0, r, \varphi_1 - \theta_1, \varphi_2 - \theta_2) = A_{2_{(0,0)}}(r, \varphi_1 - \theta_1, \varphi_2 - \theta_2) \tag{4.34}$$

Con este sencillo cambio se obtiene una enorme ventaja, ya que la evaluación de la función A_2 se reduce a considerar al elemento en la configuración $(\theta_1 = 0, \theta_2 = 0)$, en lugar de para todos los valores de $\theta_1, \theta_2 \in [-\pi, \pi)$.

De forma que el cálculo de $CB_2(\theta_1, \theta_2)$ se hace mediante la siguiente integral

$$\int A_{2_{(0,0)}}(r, \varphi_1 - \theta_1, \varphi_2 - \theta_2) B(r, \varphi_1, \varphi_2) dr d\varphi_1 d\varphi_2 \tag{4.35}$$

Y considerando la convolución de dos funciones definidas en R^3 sobre las variables θ_1 y θ_2 se llega a

$$CB_2(\theta_1,\theta_2) = \int (\bar{A}_{2_{(0,0)}} * B)_{(\varphi_1,\varphi_2)}(r,\theta_1,\theta_2)dr \qquad (4.36)$$

donde el subíndice $((\varphi_1,\varphi_2)$ indica que el producto de convolución de las funciones $\bar{A}_2$ y B se realiza para todos los valores de las varibles $(\varphi_1,\varphi_2) \in [-\pi,\pi)$, y la función $\bar{A}_{2_{(0,0)}}$ se define por

$$\bar{A}_{2_{(0,0)}}(r,\varphi_1,\varphi_2) = A_{2_{(0,0)}}(r,-\varphi_1,-\varphi_2) \qquad (4.37)$$

Finalmente, como en casos anteriores, pero en dos dimensiones en esta ocasión, se puede aplicar el teorema de convolución, con lo que la expresión 4.36 ahora se calcularía como la transformada de Fourier inversa de

$$\int \mathcal{F}\left[\bar{A}_{1_{(0,0)}}(r,\theta_1,\theta_2)\right]_{(\varphi_1,\varphi_2)} \mathcal{F}\left[B(r,\theta_1,\theta_2)\right]_{(\varphi_1,\varphi_2)} dr \qquad (4.38)$$

Uso de la Transformación Homogénea (Método de D-H) para CB_3

Como se verá, la aplicación de la deconstrucción para un robot PUMA implica utilizar un nuevo sistema de referencia que permita hacer que la función A_3 no dependa de ninguno de los parámetros asociados a los dos eslabones anteriores. Es decir, se debe encontrar una relación entre θ_3 y alguna de las coordenadas del espacio de trabajo. Para ello, trabajando en coordenadas esféricas, la elección de los sistemas de referencia de la figura 4.12, donde se establece un nuevo sistema F'_W, que coincide con F_{A_2}, es la adecuada.

Siguiendo el procedimiento propuesto en esta tesis, lo que se pretende es realizar la transformación de los puntos del espacio de trabajo dados utilizando F_W como referencia, a las coordenadas resultantes de utilizar F_{A_2} (F'_W) como referencia (ver figura 4.12); esto es, pasar de las coordenadas (r,φ_1,φ_2) a las coordenadas $(r',\varphi'_1,\varphi'_2)$.

Aplicando el método de D-H (ver apéndice A), la colocación de los ejes es la de la figura 4.14. Y los parámetros los de la tabla 4.7. Así, la relación entre los sistemas de referencia viene dada por las siguientes matrices de transformación homogénea.

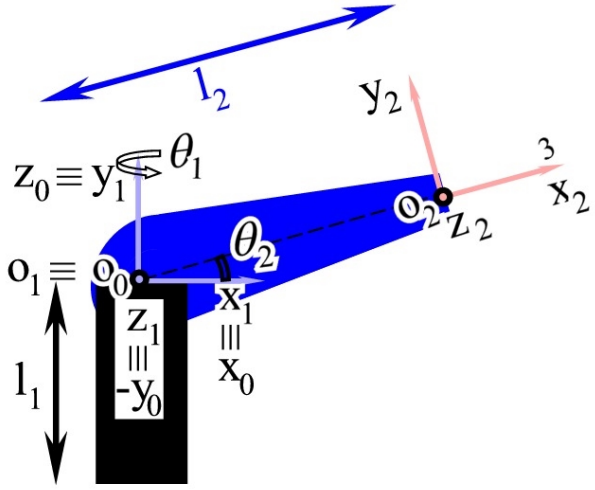


Figura 4.14: Elección de ejes según D-H para un robot PUMA de tres articulaciónes

Eslabón	θ_i	a_i	d_i	α_i
1	θ_1	0	0	$\frac{\pi}{2}$
2	θ_2	l_2	0	0

Tabla 4.7: Parámetros de Denavit-Hartenberg para un manipulador PUMA de tres articulaciones

$$ {}^{0}_{1}\mathbf{T} = \begin{bmatrix} C\theta_1 & 0 & S\theta_1 & 0 \\ S\theta_1 & 0 & -C\theta_1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{1}_{2}\mathbf{T} = \begin{bmatrix} C\theta_2 & -S\theta_2 & 0 & l_2C\theta_2 \\ S\theta_2 & C\theta_2 & 0 & l_2S\theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Una vez más, ${}^{0}_{2}\mathbf{T} = {}^{0}_{1}\mathbf{T}\,{}^{1}_{2}\mathbf{T}$, y recordando que, para cualquier

punto, $p' = {}^0_2\mathbf{T}^{-1} \cdot p$, tras realizar los cálculos pertinentes se llega a las siguientes expresiones:

$$r'=\sqrt{l_2^2+r^2-2rl_2(C\theta_1C\theta_2C\varphi_1C\varphi_2+S\theta_1C\theta_2S\varphi_1C\varphi_2+S\theta_2S\varphi_2)}$$
$$\varphi_1'=artg\left(\frac{-r(C\theta_1S\theta_2C\varphi_1C\varphi_2+S\theta_1S\theta_2S\varphi_1C\varphi_2-C\theta_2S\varphi_2)}{r(C\theta_1C\theta_2C\varphi_1C\varphi_2+S\theta_1C\theta_2S\varphi_1C\varphi_2+S\theta_2S\varphi_2)-l_2}\right)$$
$$\varphi_2'=artg\left(\frac{z'}{\sqrt{x'^2+y'^2}}\right)$$

con

$$x'=r(C\theta_1C\theta_2C\varphi_1C\varphi_2+S\theta_1C\theta_2S\varphi_1C\varphi_2+S\theta_2S\varphi_2)-l_2$$
$$y'=-r(C\theta_1S\theta_2C\varphi_1C\varphi_2+S\theta_1S\theta_2S\varphi_1C\varphi_2-C\theta_2S\varphi_2)$$
$$z'=r(S\theta_1C\varphi_1C\varphi_2-C\theta_1S\varphi_1C\varphi_2)$$

Cabe mencionar que, en este caso, hay que pasar los puntos de coordenadas esféricas a coordenas cartesianas —para aplicar el método de D-H— y el resultado de nuevo a coordendas esféricas.

Finalmente, se puede comentar que, de forma análoga a casos anteriores, la aplicación de la deconstrucción tiene las siguientes consecuencias:

- Se tiene una nueva función para describir los obstáculos en el espacio de trabajo, B'. Los valores de r', φ_1' y φ_2' dependen de algunos parámetros de D-H asociados al primer eslabón, θ_1, y al segundo, θ_2 y l_2.

- Se tiene una nueva función para describir el tercer elemento del PUMA, A_3'. Ésta no depende de ningún parámetro de D-H asociado a eslabones anteriores, sólo depende de θ_3.

Por otro lado, dado que el codo es una articulación de revolución con el eje de giro paralelo al de la articulación del hombro (figura 4.12), de toda la esfera del espacio 3D cubierta por $(r, \varphi_1, \varphi_2)$, sólo pueden ser obstáculos para $\mathbf{A}_3$ aquéllos que se encuentren en el disco de radio l_3, la longitud del tercer elemento, y con ángulo φ_1' variando entre $-\pi$ y π. Es decir, dado que se está calculando el cambio de referencia del codo según sea la configuración (θ_1, θ_2), se podría optar por trasladar todos los puntos del espacio de trabajo y considerar sólo aquellos que están en el plano de acción del elemento, esto es,

aquéllos con $\varphi_2' = 0$. Sin embargo, es más efiente sólo trasladar aquellos puntos que esten en el disco de interés, es decir, con $\varphi_1 = \theta_1$ en el sistema de referencia original y con $r' < l_3$ en el trasladado. Este concepto se ilustra en la figura 4.15.

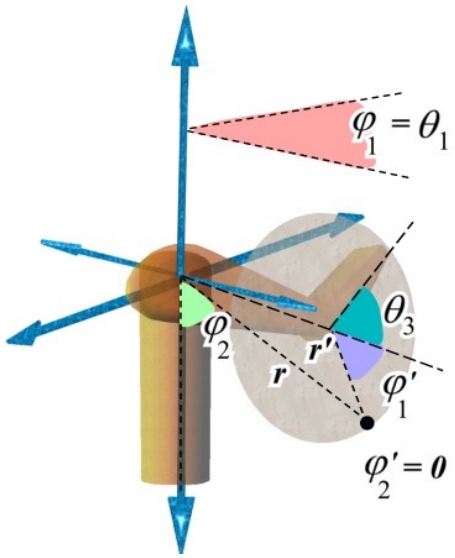


Figura 4.15: Cualquier punto que tenga $\varphi_1 = \theta_1$ pertenece al plano del disco de interés

Con el cambio de sistemas de referencia introducido, se pasa de trabajar en el espacio, a trabajar en el plano, por lo que ahora la expresión 4.28 se puede evaluar como

$$CB_3(\theta_1, \theta_2, \theta_3) = \int A_3'(\theta_3, r', \varphi_1')B'(r', \varphi_1')dr'd\varphi_1' \qquad (4.39)$$

Uso de la Convolución para el Cálculo de CB_3

Como se puede observar, de nuevo la expresión 4.39 es similar al caso de una articulación para un planar (expresiones 4.5, 4.12, 4.22), con lo que se puede evaluar de la misma forma (expresión 4.9) y computar a través del mismo algoritmo (tabla 4.1). Es decir, gracias a la relación que se encuentra entre θ_3 y

φ_1', la expresión 4.39 puede ponerse como

$$CB_3(\theta_1, \theta_2, \theta_3) = \int A'_{3_{(0)}}(r', \varphi_1' - \theta_3) B'(r', \varphi_1') dr' d\varphi_1' \quad (4.40)$$

donde, introduciendo una nueva definición,

$$\bar{A}'_{3_{(0)}}(r', \varphi_1') = A'_{3_{(0)}}(r', -\varphi_1') \quad (4.41)$$

se encuentra el siguiente producto de convolución

$$CB_3(\theta_1, \theta_2, \theta_3) = \int (\bar{A}'_{3_{(0)}} * B')_{\varphi_1'}(r', \theta_3) dr' \quad (4.42)$$

que se simplifica, aplicando el teorema de convolución, para llegar a la expresión final:

$$\mathcal{F}[CB_3(\theta_1, \theta_2, \theta_3)] = \int \mathcal{F}\left[\bar{A}'_{3_{(0)}}(r', \theta_3)\right]_{\varphi_1'} \mathcal{F}[B'(r', \theta_3)]_{\varphi_1'} dr' \quad (4.43)$$

Hay que destacar que, a diferencia de la expresión previa, donde era necesario hacer transformadas bidimensionales, en este caso las transformadas son en una dimensión, gracias a que sólo es necesario barrer discos.

Algoritmo

Como en todos los casos, para poder computar los obstáculos en el espacio de las configuraciones asociados a cada uno de los elementos, es necesaria una discretización (ver apéndice B). Una vez hecho esto, la deconstrucción entra en acción, ya que el algoritmo (tablas 4.8 y 4.9) consiste en una acumulación por esferas y en repetir, un número indeterminado de veces, la acumulación de los productos de convolución que ilustra la figura 4.3.

En esencia, el algoritmo sólo difiere en relación con las estructuras planares en que en esta ocasión, al trabajar para el hombro en un espacio tridimensional, se acumulan productos de convolución en 2D, es decir se acumula por radios de una esfera en vez de por radios de un círculo, como se hacia hasta ahora. Después, para el codo se pasa de nuevo a un espacio bidimensional, con lo que se acumulan productos de convolución en una dimensión.

```
Construir la matriz binaria B* en esféricas
Construir la matriz binaria Ā*_{2(0,0)} en esféricas
Construir la matriz binaria Ā'*_{3(0)} en polares
Para cada radio rad
   Calcular F[Ā'*_{3(0)}(rad)]
Para cada radio rad
  Calcular F[B*_{(rad)}] en 2D
Hacer ProdAcum2D = 0
Para cada radio rad
   Calcular F[Ā*_{2(0,0)}(rad)]
   Obtener P = F[B*(rad)] · F[Ā*_{2(0,0)}(rad)]
   Acumular ProdAcum2D = ProdAcum2D + P
Calcular IP2D = F^{-1}[ProdAcum2D] en 2D
Para cada θ1
   Para cada θ2  tal que  IP2D(θ1, θ2) > 0
      Para cada θ3
         Asignar  CB*(θ1, θ2, θ3) = 1
```

Tabla 4.8: Algoritmo para el cálculo del C-espacio de un robot PUMA de tres articulaciones (1^a parte)

En la figura 4.16 se muestra una ilustración de la secuencia de pasos que se deben realizar en el proceso de deconstrucción de un manipulador PUMA:

- Paso 1: considerar el sistema de referencia inicial, es decir, el asociado al segundo elemento, cuyo origen está en la intersección de los ejes de giro de cintura y hombro.
- Paso 2: considerar el segundo elemento del robot en la configuración $(\theta_1, \theta_2) = (0, 0)$. La configuración de los elementos restantes no tiene importancia en este paso.
- Paso 3: hacer la convolución en dos dimensiones (empezando en radio = 0 y avanzando, acumulando por esferas, hasta radio = longitud del segundo elemento). Los

```
Para cada θ1
   Para cada θ2  tal que IP2D(θ1, θ2) = 0
      Transformar B*  en B'*(θ1,θ2)
      Calcular F[B'*(θ1,θ2)]
      Hacer ProdAcum = 0
      Para cada radio rad
         Obtener P = F[B'*(θ1,θ2)(rad)] · F[Ā'*3(0)(rad)]
         Acumular ProdAcum = ProdAcum + P
      Calcular IP = F^-1[ProdAcum]
      Para cada θ3  tal que IP(θ3) > 0
         Asignar CB*(θ1, θ2, θ3) = 1
```

Tabla 4.9: Algoritmo para el cálculo del C-espacio de un robot PUMA de tres articulaciones (2^a parte)

siguientes tres pasos (4 a 6) se deben repetir para cada una de las configuraciones libres de (θ_1, θ_2), las cuales se conocen tras la ejcución de los tres primeros pasos (1 a 3).

- Paso 4: colocarse en el origen del nuevo sistema de referencia, cuya posición viende determinada por los valores de (θ_1, θ_2) de la configuración libre que se está considerando.
- Paso 5: considerar el tercer elemento en la configuración inicial ($\theta_3 = 0$).
- Paso 6: realizar la convolución en una dimensión (comenzando en radio = 0, acumulando por circunferencias, hasta radio = longitud del tercer elemento).

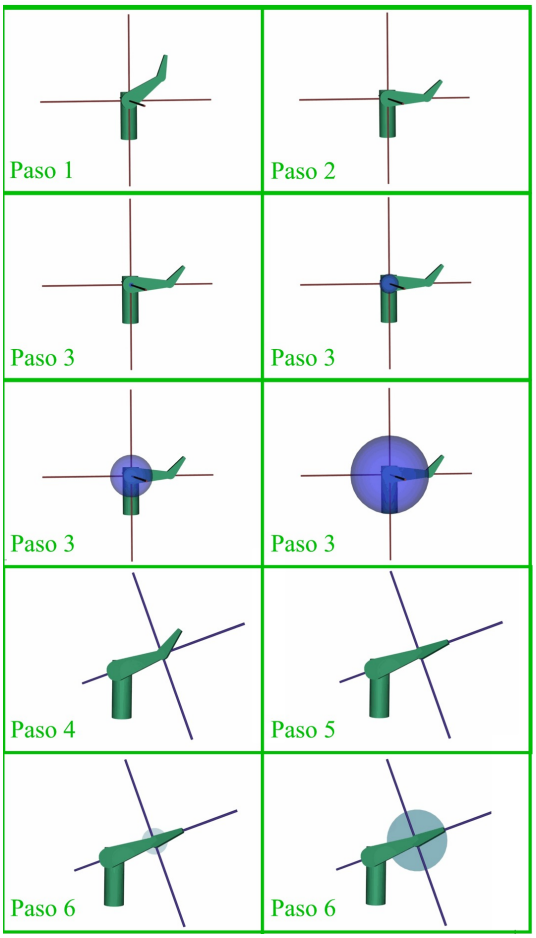


Figura 4.16: Deconstrucción de un robot PUMA

4.1.5 Robot PUMA Redundante de Cuatro Articulaciones

Siguiendo con ejemplos de la deconstrucción en el espacio, llega el turno de considerar otro caso interesante: un robot PUMA redundante de cuatro articulaciones (figura 4.17).

Considérese el robot articulado, $\mathbf{A}$, formado por cuatro objetos rígidos, $\mathbf{A}_1$, $\mathbf{A}_2$, $\mathbf{A}_3$ y $\mathbf{A}_4$, que se mueven en R^3 mediante cuatro articulaciones mecánicas de revolución. En este caso se debe tener en cuenta la redundancia, pues se tienen cuatro grados de libertad $(\theta_1, \theta_2, \theta_3, \theta_4) \in [-\pi, \pi)$. Por analogía con el robot PUMA de tres articulaciones, las de este manipulador se llamarán como sigue: cintura (θ_1), hombro (θ_2), codo (θ_3) y cuarta (θ_4).

Como en el caso precedente, se eligen las coordenadas esféricas, $(r, \varphi_1, \varphi_2)$, que pueden variar en el intervalo $[0, l_2 + l_3 + l_4] \times [-\pi, \pi) \times \left[\frac{-\pi}{2}, \frac{\pi}{2}\right)$, donde l_2, l_3 y l_4 son las longitudes del segundo, tercer y cuarto elementos, respectivamente.

Como se verá en los siguientes apartados, este es el de caso de estudio más complejo de los comentados hasta ahora. Sin embargo, después de un análisis deconstructivo (figura 4.18) se puede comprobar que este manipulador se pude descomponer en dos estrucuturas ya estudiadas —que a su vez han pasado por el proceso de deconstrucción.

Elección de los Sistemas de Referencia

Se siguen manteniendo los sistemas de referencia elegidos para el PUMA, pero ahora hay que añadir uno más. Es decir, se sitúan los sistemas de referencia de la siguiente forma (figura 4.17):

- Los tres sistemas de referencia F_W, F_{A_0} y F_{A_1}, asociados respectivamente con el espacio de trabajo W, al robot, y al primer elemento $\mathbf{A}_1$, son equivalentes. El origen está en el punto en que intersectan los ejes de giro de los dos primeros elementos. Un eje se corresponde con el eje de giro de la cintura y el otro con el eje de giro del hombro. El tercer eje se elige para que forme un triedo con estos.

- El origen de F_{A_2} se coloca al final del segundo elemento, en el punto de articulación del codo. Un eje se coloca

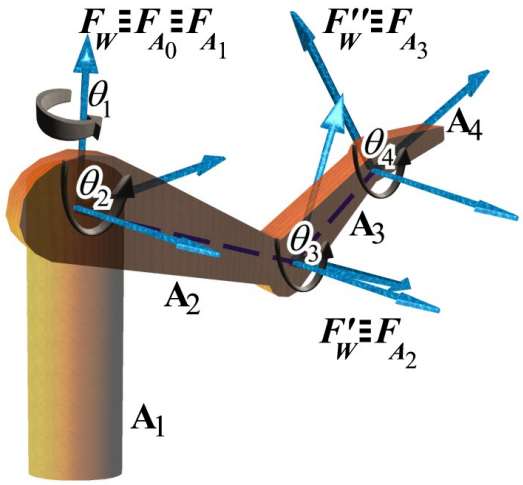


Figura 4.17: Un robot PUMA redundante en un entorno de trabajo 3D

Figura 4.18: Análisis de la estructura de un robot PUMA redundante

siguiendo la línea que une las articulaciones de los elementos $\mathbf{A}_2$ y $\mathbf{A}_3$; otro se corresponde con el eje de giro del codo; y el tercero es perpendicular a los otros dos.

- El origen de F_{A_3} se coloca al final del tercer elemento, en el punto de articulación del siguiente elemento. Un eje se coloca siguiendo la línea que une las articulaciones de los elementos $\mathbf{A}_3$ y $\mathbf{A}_4$; otro se corresponde con el eje de giro de la cuarta articulación; y el tercer eje es perpendicular a los otros dos.

Elección de Funciones Coordenadas y Cálculo de CB

Como en el caso del PUMA se siguen eligiendo las coordenadas esféricas, de esta forma se encuentran las relaciones de θ_1 con φ_1 y de θ_2 con φ_2, con las ventajas que esto conlleva.

Por otro lado, el proceso de deconstrucción coincide exactamente, hasta llegar al codo, con la de un PUMA de tres articulaciones. Es decir, en ese punto de la deconstrucción se pasa de considerar un espacio 3D a considerar un disco de interés. En el siguiente paso, donde se encuentra una relación del mismo tipo para el cuarto grado de libertad, es decir, θ_4 (asociado a la cuarta articulación), se mantendrá el mismo plano de acción, pero cambiando a otro disco de interés.

Así, para este caso, el conjunto **CB** es la unión de cuatro conjuntos

$$\mathbf{CB} = \mathbf{CB}_1 \cup \mathbf{CB}_2 \cup \mathbf{CB}_3 \cup \mathbf{CB}_4 \tag{4.44}$$

Y las funciones asociadas (expresión 3.9) se evalúan según la expresión 3.10. Las expresiones para los tres primeros elementos son iguales que para el caso del PUMA de tres articulaciones (expresión 4.26 para CB_1, expresión 4.27 para CB_2 y expresión 4.28 para CB_3), mientras que la expresión para $CB_4(\theta_1, \theta_2, \theta_3, \theta_4)$ es

$$\int A_4(\theta_1, \theta_2, \theta_3, \theta_4, r, \varphi_1, \varphi_2) B(r, \varphi_1, \varphi_2) dr d\varphi_1 d\varphi_2 \tag{4.45}$$

Así, habrá que definir las funciones A_k y B del formalismo correspondientes a un robot PUMA de cuatro articulaciones. La función B es la misma que para el caso del manipulador PUMA (expresión 4.29). Las funciones $A_k : C \times W \rightarrow R$ se definen de forma que A_1, A_2 y A_3 son la mismas que en el caso del robot PUMA, expresiones 4.30, 4.31 y 4.32, respectivamente. Mientras que asociada con el cuarto elemento vendría la función $A_4(\theta_1, \theta_2, \theta_3, \theta_4, r, \varphi_1, \varphi_2)$ definida como sigue

$$\begin{cases} 1 & \text{si } (r, \varphi_1, \varphi_2) \in \mathbf{A}_4(\theta_1, \theta_2, \theta_3, \theta_4) \\ 0 & \text{si } (r, \varphi_1, \varphi_2) \notin \mathbf{A}_4(\theta_1, \theta_2, \theta_3, \theta_4) \end{cases} \tag{4.46}$$

La evaluación del espacio de las configuraciones para las tres primeras articulaciones se corresponde exactamente al caso anterior, un robot PUMA de tres articulaciones. Es decir,

se hace en primer lugar una acumulación de productos de convolución en dos dimensiones (acumulando por radios de una esfera), y através de una transformación de los puntos del espacio de trabajo, se pasa a considerar un plano, y, en concreto un disco de interés. Se verá que a continuación, para poder hacer la deconstrucción del cuarto elemento, se sigue trabajando en el mismo plano, pero con otro sistema de referencia y otro disco de interés.

Uso de la Trasnformación Homogénea (Método de D-H) para el Cálculo de CB_4

En este momento, interesa trabajar con un sistema de referencia que permita utilizar una función de $\mathbf{A}_4$ que no dependa de ningún parámetro de los elementos anteriores en la cadena cinemática.

En este caso, para poder realizar el cambio aludido hay que usar $F''_W \equiv F_{A_3}$ como nuevo sistema de refrencia, lo que implica (figura 4.17) hacer, para cada punto del espacio de trabajo, una transformación de las coordenadas[6] $(r, \varphi_1, \varphi_2)$, dadas respecto a F_W, a las coordenadas $(r'', \varphi''_1, \varphi''_2)$, dadas respecto a F''_W.

Esto se puede hacer aplicando el método de D-H como hasta ahora, colocando los ejes adecuadamente, y construyendo las matrices de transformación según los parámetros. Sin embargo, ésta, aunque aún asequible, es una transformación complicada; imaginando una cadena de n eslabones, puede llegar a ser intratable.

Así, se opta por una solución que permite extender la deconstrucción a cualquier número de eslabones en una cadena cinemática: guardar las coordenadas de los puntos transformados para el eslabón anterior y hacer una nueva transformación respecto a estos.

En efecto, si se examina la figura 4.17, se puede ver que considerar aisladamente el codo y el cuarto elemento, con las coordenadas (r', φ'_1) de los puntos del espacio de trabajo, dadas respecto al sistema de referencia $F'_W \equiv F_{A_2}$, es enfrentarse de nuevo al problema de deconstruir un robot planar de dos articulaciones (ver figura 4.4). Por lo que aplicando el método

[6] Como se sigue trabajando en un plano, no es necesario el valor de φ''_2, ya que siempre es 0.

como se explicó en aquella sección (página 68), se llega a las siguientes ecuaciones para calcular las nuevas coordenadas.

$$r'' = \sqrt{r'^2 + l_3^2 - 2r'l_3C(\varphi_1' - \theta_3)}$$

$$\varphi_1'' = artg\left(\frac{-r'S(\theta_3 - \varphi_1')}{rC(\theta_3 - \varphi_1') - l_3}\right)$$

Con el cambio de sistemas de referencia introducido, trabajando en el mismo plano que para el codo, la expresión 4.45 se puede evaluar como

$$CB_4(\theta_1, \theta_2, \theta_3, \theta_4) = \int A_4'(\theta_4, r'', \varphi_1'')B''(r'', \varphi_1'')dr''d\varphi_1'' \tag{4.47}$$

Uso de la Convolución para el Cálculo de CB_4

Una vez más, la expresión 4.47 es similar al caso de una articulación para un planar (expresiones 4.5, 4.12, 4.22), con lo que se puede evaluar de la misma forma (expresión 4.9) y computar a través del mismo algoritmo (tabla 4.1). Es decir, gracias a la relación que se encuentra entre θ_4 y φ_1'', la expresión 4.47 puede ponerse como

$$CB_4(\theta_1, \theta_2, \theta_3, \theta_4) = \int A'_{4_{(0)}}(r'', \varphi_1'' - \theta_4)B''(r'', \varphi_1'')dr''d\varphi_1'' \tag{4.48}$$

donde, introduciendo esta nueva definición,

$$\bar{A}'_{4_{(0)}}(r'', \varphi_1'') = A'_{4_{(0)}}(r'', -\varphi_1'') \tag{4.49}$$

se encuentra el siguiente producto de convolución

$$CB_4(\theta_1, \theta_2, \theta_3, \theta_4) = \int (\bar{A}'_{4_{(0)}} * B'')_{\varphi_1''}(r'', \theta_4)dr'' \tag{4.50}$$

que se simplifica, aplicando el teorema de convolución, para llegar a la expresión final, en la que $CB_4(\theta_1, \theta_2, \theta_3, \theta_4)$ se obtendría a través de la transformada de Fourier inversa de:

$$\int \mathcal{F}\left[\bar{A}'_{4_{(0)}}(r'', \theta_4)\right]_{\varphi_1''} \mathcal{F}\left[B''(r'', \theta_4)\right]_{\varphi_1''} dr'' \tag{4.51}$$

Algoritmo

Como en todos los casos, para poder computar los obstáculos en el espacio de las configuraciones asociados a cada uno de los elementos, es necesaria una discretización (ver apéndice B). Una vez hecho esto, se desconstruye cada elemento, con lo que el algoritmo (tablas 4.10 y 4.11) consiste en una acumulación por esferas de productos de convolución en dos dimensiones, y luego, en diferentes planos, repetir, como se indica en dos recuadros, la operación que ilustra la figura 4.3.

```
Construir la matriz binaria B* en esféricas
Construir la matriz binaria Ā*_{2(0,0)} en esféricas
Construir la matriz binaria Ā'*_{3(0)} en polares
Construir la matriz binaria Ā'*_{4(0)} en polares
Para cada radio rad
   Calcular F[B*_{(rad)}] en 2D
   Calcular F[Ā'*_{3(0)}(rad)]
   Calcular F[Ā'*_{4(0)}(rad)]
Hacer ProdAcum2D = 0
Para cada radio rad
   Calcular F[Ā*_{2(0,0)}(rad)]
   Obtener P = F[B*(rad)] · F[Ā*_{2(0,0)}(rad)]
   Acumular ProdAcum2D = ProdAcum2D + P
Calcular IP2D = F^{-1}[ProdAcum2D] en 2D
Para cada θ1
   Para cada θ2 tal que IP2D(θ1, θ2) > 0
      Para cada θ3
         Para cada θ4
            Asignar CB*(θ1, θ2, θ3, θ4) = 1
```

Tabla 4.10: Algoritmo para el cálculo del C-espacio de un robot PUMA de cuatro articulaciones ($1^a parte$)

En esencia, el algoritmo sólo difiere del caso visto para el

```
Para cada θ1
   Para cada θ2  tal que IP2D(θ1, θ2) = 0
      Transformar B*  en B'*(θ1,θ2)
      Calcular F[B'*(θ1,θ2)]
      Hacer ProdAcum = 0
      Para cada radio rad
         Obtener P = F[B'*(θ1,θ2)(rad)] · F[Ā'*3(0)(rad)]
         Acumular ProdAcum = ProdAcum + P
      Calcular IP = F^-1[ProdAcum]
      Para cada θ3  tal que IP(θ3) > 0
         Para cada θ4
         Asignar CB*(θ1, θ2, θ3, θ4) = 1
      Para cada θ3  tal que IP(θ3) = 0
         Transformar B'*(θ1,θ2)  en B''*(θ1,θ2,θ3)
         Calcular F[B''*(θ1,θ2,θ3)]
         Hacer ProdAcum = 0
         Para cada radio rad
            Obtener P = F[B''*(θ1,θ2,θ3)(rad)] · F[Ā'*4(0)(rad)]
            Acumular ProdAcum = ProdAcum + P
         Calcular IP = F^-1[ProdAcum]
         Para cada θ4  tal que IP(θ4) > 0
            Asignar CB*(θ1, θ2, θ3, θ4) = 1
```

Tabla 4.11: Algoritmo para el cálculo del C-espacio de un robot PUMA de cuatro articulaciones ($2^a parte$)

robot PUMA de tres articulaciones en que hay que guardar las coordenadas del espacio de trabajo para cada configuración (θ_1, θ_2) libre, pues luego se utilizan para hacer una transformación sencilla para cada $(\theta_2, \theta_2, \theta_3)$ libre.

En la figura 4.19 se muestra en secuencia el proceso:

- Paso 1: colocarse en el origen del primer sistema de referencia.
- Paso 2: considerar el robot en la configuración $(\theta_1, \theta_2) = (0, 0)$.

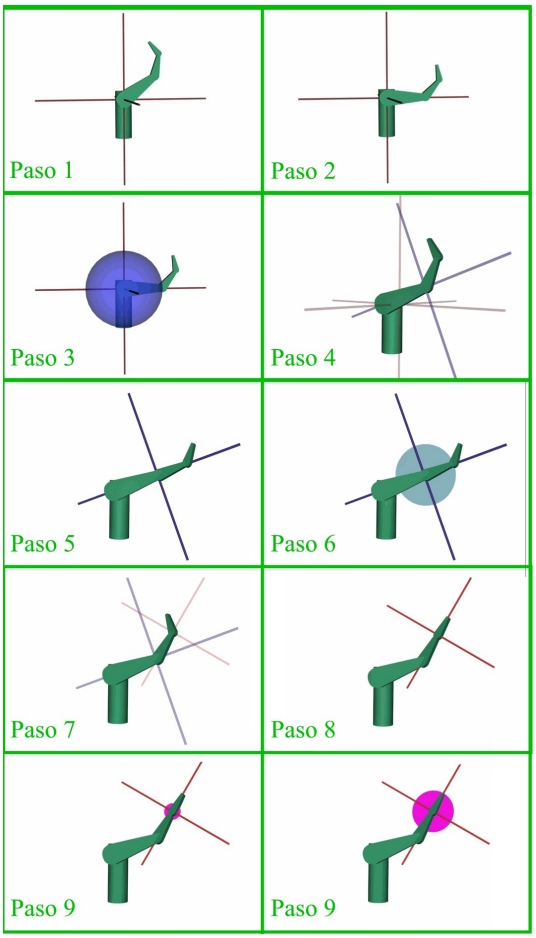


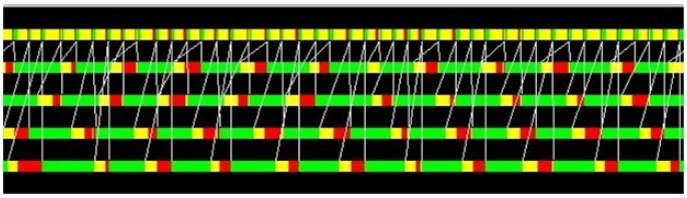
Figura 4.19: Deconstrucción de un robot PUMA redundante

- Paso 3: hacer la convolución en dos dimensiones (empezando en radio = 0 y avanzando, acumulando esferas, hasta radio = longitud del hombro).
- Paso 4: para cada configuración libre de (θ_1, θ_2) colocarse en el origen del nuevo sistema de referencia.
- Paso 5: considerar la configuración inicial $(\theta_3 = 0)$.

- Paso 6: realizar la convolución en una dimensión (comenzando en radio = 0, acumulando por circunferencias, hasta radio = longitud del codo).
- Paso 7: para cada configuración libre de $(\theta_1, \theta_2, \theta_3)$ colocarse en el origen del nuevo sistema de referencia.
- Paso 8: considerar la configuración inicial $(\theta_4 = 0)$.
- Paso 9: realizar la convolución en una dimensión (comenzando en radio = 0, acumulando por circunferencias, hasta radio = longitud del cuarto elemento).

Capítulo 5

Técnicas y Herramientas



Hobbes: Entramos en una nueva década.
Calvin: ¿Y qué? Buf. ¿Dónde están los cohes volantes? ¿Y las colonias lunares? ¿Y las botas antigravedad, eh? ¿A esto llaman una década nueva? ¿A esto llaman futuro? ¡Ja! ¿Y las mochilas impulsoras? ¿Y los rayos desintegradores? ¿Y las ciudades flotantes?
Hobbes: Francamente, no estoy seguro que la gente esté preparada para manejar la tecnología que tienen.
Calvin: ¡Mira esto, ¿Aún no controlamos el clima?! ¡Pues sí que...!

El Gran Calvin y Hobbes
Ilustrado
BILL WATTERSON

ESTE CAPÍTULO se dedica a una breve exposición de las principales técnicas y herramientas que han permitido implementar el método de la deconstrucción del espacio de las configuraciones, bien como instrumentos para producir el resultado final, bien permitiendo explorar las posibilidades para llegar a una mejor solución.

5.1 Técnicas

A continuación se exponen las características destacadas de las dos técnicas principales usadas en este trabajo: el Cálculo

Paralelo, como una forma de conseguir mejores resultados y más rápido; y las Redes de Petri, como primera aproximación al problema, ya que permiten modelar y validar las soluciones propuestas. En ambos casos, tras una somera introducción general, se ubican los casos particulares elegidos, justificando estas elecciones.

5.1.1 Programación Paralela

Desde la aparición de los ordenadores, la necesidad de mayores velocidades de cálculo ha sido continua. Cada nueva aplicación exige trabajar al límite de su velocidad. Como hasta ahora la velocidad de los ordenadores parece siempre crecer, es tentador pensar que estos van a ser lo suficiente rápidos como para satisfacer el creciente apetito computacional. Sin embargo, la historia dice que mientras una tecnología particular es suficiente para aplicaciones conocidas, siempre aparecen nuevas aplicaciones propiciadas por dichas tecnologías y que demandarán el desarrollo de nuevas tecnologías[1].

Tendencias en Paralelismo

Tradicionalmente, el desarrollo de computadores más rápidos estaba impulsado por la simulación de sistemas muy complejos como el clima, dispositivos mecánicos, circuitos electrónicos, reacciones químicas, etc. Sin embargo, este impulso lo ejercen actualmente las aplicaciones comerciales, que requieren una capacidad de procesamiento de grandes cantidades de datos de formas muy sofisticadas. Aunque la supremacía de las aplicaciones comerciales pueda definir la arquitectura de la mayoría de los futuros computadores paralelos, las aplicaciones científicas tradicionales seguirán siendo usuarias muy importantes de las tecnologías de computación paralela.

[1] Un ejemplo curioso de este fenómeno ocurrió a finales de los años cuarenta, cuando el gobierno británico encargó un estudio en el que se concluía que las necesidades computacionales de Gran Bretaña se podrían satisfacer con dos o tres computadoras. En aquellos días los ordenadores se utilizaban únicamente para calcular las tablas de balística. Los autores del informe no tomaron en consideración su aplicación en la ciencia y la ingeniería, dejando aparte las aplicaciones comerciales que pronto dominarían la Informática. Similarmente, el estudio de mercado inicial para *Cray Research* preveía una demanda mundial de diez supercomputadores; desde entonces se han vendido cientos.

En general, el rendimiento de un ordenador depende directamente del tiempo requerido para realizar una operación básica y del número de operaciones básicas que se pueden realizar concurrentemente. El tiempo de ejecución de una operación básica está finalmente limitado por el *ciclo de reloj* del procesador, es decir, el tiempo que se requiere para la operación más primitiva. Sin embargo, los tiempos de ciclo de reloj están decreciendo lentamente y parece que se están acercando a límites físicos, como es la velocidad de la luz. Por tanto, no se puede depender de procesadores más rápidos para proporcionar un incremento en el rendimiento de los computadores. Los diseñadores de ordenadores utilizan gran variedad de técnicas para superar estas limitaciones en el rendimiento de un único ordenador, incluyendo la segmentación (*pipelining*, diferentes etapas de varias instrucciones que se ejecutan concurrentemente) y las unidades funcionales múltiples (varios multiplicadores, sumadores, etc., controlados por un único flujo de ejecución). Este enfoque se ve facilitado por los avances en la tecnología VLSI (*Very Large Scale Integration*) que continúa decrementando el número de componentes necesarios para construir un ordenador.

La Red como supercomputador

Aprovechando el auge de Internet ya se han desarrollado mecanismos para aprovechar la capacidad de procesamiento —que de otro modo se desperdiciaría— de los ordenadores de nuestros vecinos conectados a la red mientras éstos leen el correo electrónico o imprimen una fotografía (SETI@Home o DCTI), en ocasiones sin que éstos lo sepan (como el caso de Altnet de Brilliant Digital Enterprise Inc.).

Otra tendencia que está cambiando la Informática es la creciente capacidad de los ordenadores conectados en red, que hacen posible el diseño de aplicaciones que utilizan recursos físicamente distribuidos como si fueran parte del mismo ordenador. El procesamiento distribuido no es simplemente un subconjunto de la computación paralela, ya que en el procesamiento distribuido hay que tener muy en cuenta problemas como la fiabilidad, la seguridad y la heterogeneidad, que generalmente son considerados tangencialmente en la computación paralela. No obstante, la tarea básica de desarrollar programas que puedan ejecutarse en varios ordenadores a la vez, sigue siendo un problema de computación paralela.

De lo anteriormente expuesto se deduce que las tendencias en las aplicaciones, el diseño de computadores y las redes de ordenadores, sugieren un futuro en el que el paralelismo predominará no sólo en los supercomputadores sino también en estaciones de trabajo, ordenadores personales y redes de ordenadores.

Modelos de Máquina Paralela

La vertiginosa penetración de los computadores en el comercio, la ciencia, y la educación se debe fundamentalmente a su temprana estandarización en un único modelo, *La Máquina de Von Neumann*. Un ordenador de Von Neumann consta de una unidad central de procesamiento (UCP) conectada a una unidad de almacenamiento (memoria). La UCP ejecuta un programa almacenado que especifica una secuencia de operaciones de lectura y escritura en la memoria. Este modelo se ha comportado como un modelo muy robusto. Y su permanencia durante más de medio siglo ha permitido el perfeccionamiento de algoritmos y de lenguajes de programación de forma independiente de los desarrollos en la arquitectura de computadores.

De esta manera es deseable la existencia de un modelo de máquina paralela que sea simple, para facilitar su comprensión y la programación, y realista, para asegurar que los programas desarrollados para este modelo se ejecuten con una eficiencia razonable en ordenadores reales.

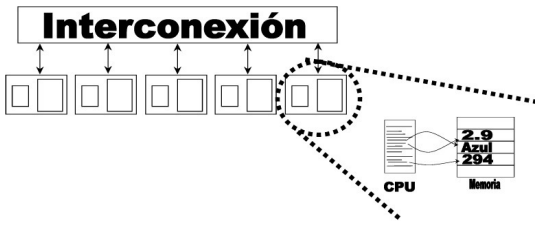


Figura 5.1: Un multicomputador

Así, un modelo de máquina paralela llamado *Multicomputador* [Fos95] reúne estos requisitos. Un multicomputador consiste en una serie de ordenadores de Von Neumann, o nodos, unidos mediante una red de interconexión (figura 5.1). Cada computador ejecuta su propio programa. Este programa puede acceder a la memoria local y puede enviar y recibir mensajes a través de la red. Los mensajes se utilizan para comunicar/sincronizar con otros computadores o, lo que es lo mismo, para leer y escribir en memorias remotas. En la red ideal, el coste de enviar un mensaje entre dos nodos es inde-

pendiente tanto de la localización del nodo como del tráfico en la red, dependiendo, sin embargo, del tamaño del mensaje.

El multicomputador es muy similar a lo que comúnmente se conoce como computador *MIMD de memoria distribuida* (*Múltiples Instrucciones Múltiples Datos*). MIMD significa que cada procesador puede ejecutar un flujo de instrucciones distinto sobre sus propios datos locales; así, memoria distribuida quiere decir que la memoria está repartida entre los distintos elementos de procesamiento, en vez de estar localizada en un lugar centralizado. La diferencia principal entre el multicomputador y un computador MIMD es que, en el último, el coste de enviar un mensaje entre dos nodos puede no ser independiente de la localización del nodo o del tráfico de la red. Ejemplos de este tipo de computadores son: *IBM SP*, *Intel Paragon*, *Thinking Machines CM5*, *Cray T3D* y *nCUBE*.

Otra clase importante de computador paralelo es el *Multiprocesador*, o computador *MIMD de memoria compartida*. En los multiprocesadores, todos los procesadores comparten el acceso a una memoria común a través de un bus o de una jerarquía de buses. En el modelo ideal de *Máquina Paralela de Acceso Aleatorio* (*Parallel Ramdom Access Machine*, PRAM), el cual se utiliza a menudo para realizar estudios teóricos de algoritmos paralelos, cualquier procesador puede acceder a cualquier elemento de memoria en el mismo espacio de tiempo. En la práctica, el escalado de esta arquitectura normalmente introduce alguna forma de jerarquía de memoria; en particular, se puede reducir la frecuencia con la que se accede a la memoria compartida mediante la introducción de copias de datos utilizados frecuentemente en una caché asociada a cada procesador. El acceso a esta caché es mucho más rápido que el de la memoria compartida, por tanto, la localidad es importante y las diferencias entre los multicomputadores y los multiprocesadores en realidad son simples cuestiones de grado. De esta forma los programas desarrollados para multicomputadores pueden ejecutarse en los multiprocesadores eficientemente, ya que la memoria compartida permite una implementación eficiente del paso de mensajes. Son ejemplos de esta clase de computadores paralelos: *Silicon Graphics Challenge*[2], *Sequent Simetry* y las

[2]Silicon Graphics actualmente comercializa la serie de servidores *Origin*; en una de estas máquinas, concretamente una *Origin 200*, se han

múltiples estaciones de trabajo multiprocesador.

Existe una clase más especializada de computadores paralelos, los computadores *SIMD (Simple Instrucción Múltiples Datos)*, en los cuales todos los procesadores ejecutan el mismo flujo de instrucciones sobre diferentes datos. Este planteamiento puede reducir tanto la complejidad del software como la del hardware, pero tiene el inconveniente de que sólo es apropiado para problemas específicos que están caracterizados por un elevado grado de regularidad, como es el caso del procesamiento de imágenes y algunas simulaciones numéricas. Los algoritmos diseñados para un multicomputador en general no pueden ejecutarse en un computador SIMD. Un ejemplo de esta clase de máquinas es *MasPar MP*.

Finalmente, dos clases de sistemas de computación que en ocasiones se utilizan como computadores paralelos son las estaciones de trabajo conectadas a través de redes de área local (*Local Area Network*, LAN), en las que los ordenadores están físicamente cerca y están conectados mediante una red rápida, y las estaciones de trabajo conectadas a través de redes de área amplia (*Wide Area Network*, WAN), en las que se conectan máquinas geográficamente distribuidas. Aunque los sistemas de este tipo introducen otros elementos como la seguridad y la fiabilidad, pueden verse para muchos propósitos como multicomputadores, aunque con grandes costes de intercomunicación.

Modelos de Programación Paralela

El modelo de máquina de Von Neumann supone un procesador capaz de ejecutar secuencias de instrucciones. Una instrucción puede especificar, además de varias operaciones aritméticas, la dirección del dato que debe ser leído o escrito en memoria y/o la dirección de la siguiente instrucción que debe ejecutarse. Aunque es posible programar un computador en términos de este modelo básico mediante lenguaje máquina, este método es para la mayoría de los propósitos prohibitivamente complejo, ya que se deben conocer millones de posiciones de memoria y organizar la ejecución de miles de instrucciones máquina. Así, se utilizan técnicas de diseño modular donde complejos pro-

probado los códigos desarrollados para realizar esta tesis doctoral.

gramas se construyen a partir de componentes simples, y estos componentes se estructuran en términos de abstracciones de alto nivel como estructuras de datos, lazos iterativos y procedimientos. Las abstracciones como los procedimientos hacen la explotación de la modularidad más fácil ya que permiten que se pueda manejar los objetos sin un conocimiento de su estructura interna. Así, lenguajes como *Pascal*, *C* o *Ada*, logran que los diseños se expresen en términos de dichas abstracciones y sean traducidos automáticamente a código ejecutable.

La programación paralela introduce nuevas fuentes de complejidad: si se tuviera que programar al nivel más bajo, no sólo se incrementaría el número de instrucciones ejecutadas, si no que se tendría que controlar explícitamente la ejecución de miles de procesadores y coordinar millones de interacciones entre procesadores. Así, *abstracción* y *modularidad* son, como mínimo, tan importantes como en la programación secuencial. De hecho, se puede destacar la modularidad como el cuarto requisito fundamental para el software paralelo, junto con la *concurrencia*, la *escalabilidad* y la *localidad*; conceptos que se introducirán en breve.

De entre todos los modelos que se van a glosar a continuación, habrá que hacer una cuidadosa elección a la hora de realizar el diseño y la implementación de cada una de las diferentes aplicaciones. Para poder elegir el modelo adecuado a un caso en particular es necesario realizar, previamente a las fases de diseño e implementación, un concienzudo estudio del problema para identificar de forma clara cuáles son las ventajas e inconvenientes que aportarían cada uno de los modelos. Además, no se debe olvidar que aspectos de tipo práctico, como es, por ejemplo, el tipo de computador sobre el que se va a utilizar la aplicación paralela, condicionan esta elección.

- **Tareas y Canales.** Inicialmente se considera qué abstracciones son apropiadas y útiles en un modelo de programación paralela. Claramente, se necesitan algunos mecanismos que permitan un explícito análisis sobre la concurrencia y la localidad y que faciliten el desarrollo de programas modulares y escalables. Además, se necesitan abstracciones que sean fáciles de usar y que encajen en la arquitectura del modelo multicomputador. Aunque se podrían elegir múltiples abstracciones, dos de ellas se

ajustan especialmente a las características planteadas: *tareas* y *canales*. A continuación se enumeran los elementos que caracterizan a estas abstracciones [Fos95]:

Tareas

Dada su estructura, cada tarea puede verse como una máquina de Von Neumann virtual.

1. Un cálculo paralelo consiste en una o varias tareas. Las tareas se ejecutan concurrentemente. El número de tareas puede variar durante la ejecución.
2. Una tarea encapsula un programa secuencial y memoria local. Además hay una serie de puertos de entrada y de salida que definen su interfaz con el entorno.
3. Una tarea puede realizar cuatro acciones básicas además de leer y escribir en su memoria local: enviar mensajes a sus puertos de salida, recibir mensajes en sus puertos de entrada, crear nuevas tareas y finalizar.
4. La operación de envío es asíncrona: se completa inmediatamente. Una operación de recepción es síncrona: provoca el bloqueo de la ejecución de la tarea hasta que el mensaje esté disponible.
5. Los pares de puertos entrada/salida se pueden conectar mediante colas de mensajes llamadas canales. Los canales se pueden crear y eliminar, y se pueden incluir en los mensajes referencias a los canales (puertos), así la conectividad puede variar dinámicamente.
6. Las tareas se pueden asignar a los procesadores físicos disponibles de varias formas; el método empleado no afecta a la semántica de un programa. En particular, múltiples tareas pueden asignarse a un único procesador, y también, una única tarea se podría asignar a múltiples procesadores.

La abstracción tarea proporciona un mecanismo para hablar de *localidad*: los datos contenidos en la memoria local de una tarea están cerca; el resto son remotos. La abstracción canal proporciona un mecanismo para indicar que para que el cálculo en una tarea se pueda realizar se requieren datos de otra tarea, lo que se conoce como *dependencia de datos*.

Dado que las tareas interactúan utilizando el mismo mecanismo (canales) independientemente de la localización de las tareas, el resultado del cálculo de un programa no depende del lugar en que se ejecute una tarea. Los algoritmos se pueden diseñar e implementar sin tener en cuenta el número de procesadores en los que se ejecutarán. Ésta es una forma directa de conseguir *escalabilidad*: al incrementar el número de procesadores, el número de tareas por procesador se reduce, pero el algoritmo no necesita ser modificado.

Una tarea es un bloque de construcción natural en el diseño modular. Ésta encapsula tanto los datos como el código que opera sobre estos datos; los puertos sobre los que envía y recibe constituyen su interfaz. Por tanto, las ventajas de la *modularidad* son accesibles para el modelo tarea/canal.

- **Paso de mensajes**. El modelo de paso de mensajes es, probablemente, el modelo de programación paralela más extendido. Los programas que utilizan paso de mensajes, como los de tarea/canal, crean múltiples tareas, cada una de las cuales encapsula datos locales. Cada tarea se identifica con un nombre único y las tareas interaccionan, enviando y recibiendo mensajes a y desde tareas a través de sus identificativos. A este respecto, el paso de mensajes no es más que una pequeña variación del modelo tarea/canal, diferenciándose únicamente en el mecanismo utilizado para la transferencia de los datos.

  Aunque el modelo de paso de mensajes no prohíbe la creación dinámica de tareas, la ejecución de múltiples tareas por procesador o la ejecución de diferentes programas por tareas diferentes, generalmente, los sistemas de paso de mensajes crean un número fijo de tareas al inicio del programa y no permiten crear o destruir tareas durante la ejecución. Se dice que estos sistemas implementan un modelo de programación *Simple Programa Múltiples Datos, SPMD*(*Single Program Multiple Data*), ya que cada tarea ejecuta el mismo programa pero opera sobre diferentes datos. El modelo SPMD es suficiente para un amplio abanico de problemas paralelos pero no facilita el desarrollo de algunos algoritmos paralelos.

- **Paralelismo de datos**. Se trata de un modelo de programación paralela muy utilizado. Busca explotar la concurrencia que se deriva de aplicar la misma operación a múltiples elementos de una estructura de datos, como por ejemplo, sumar 2 a todos los elementos de una matriz o incrementar el salario a todos los empleados con más de 5 años de servicio. Un programa de paralelismo de datos consta de una secuencia de operaciones de este tipo. Dado que cada operación en cada elemento de datos puede verse como una tarea independiente, la granularidad natural del cálculo de paralelismo de datos es pequeña, y el concepto de localidad no surge de forma clara. Por tanto, los compiladores de paralelismo de datos requieren que el programador proporcione información sobre cómo se tienen que distribuir los datos sobre los procesadores, o lo que es lo mismo, cómo se reparten los datos en tareas. Así, el compilador (*High Performance Fortran*, por ejemplo) puede traducir el paralelismo de datos en un programa SPMD, generando automáticamente el código paralelo. El modelo tarea/canal permite diseñar algoritmos directamente aplicables a los programas de paralelismo de datos.

- **Memoria Compartida**. En este modelo de programación, las tareas comparten un espacio común de direcciones de memoria, el cual se lee y escribe asincrónicamente. Se deben utilizar varios mecanismos para controlar el acceso a la memoria compartida como los semáforos y cierres. Una ventaja de este modelo desde el punto de vista del programador es que la noción de *propiedad* de los datos no existe, y por consiguiente no hay necesidad de especificar explícitamente la comunicación de datos entre productores y consumidores. Este modelo puede simplificar el desarrollo de programas. Sin embargo, la comprensión y el manejo de la localidad son más difíciles, así como la escritura de programas deterministas[3].

En la sección dedicada a las herramientas (subsección 5.2.1 en la página 124) se profundiza en el modelo de paso de men-

[3]Un algoritmo o programa es determinista si la ejecución con una determinada entrada produce siempre el mismo resultado.

sajes, puesto que es el elegido para diseñar los algoritmos paralelos que permiten la evaluación del espacio de las configuraciones para estructuras robóticas.

5.1.2 Redes de Petri

Las redes de Petri (*Petri Nets*) fueron introducidas por Carl Adam Petri, que con el trabajo presentado en su tesis doctoral, *Kommunikation mit Automaten* (Comunicación con autómatas) de 1962 [Pet62][Pet66], inició un área de investigación completamente nueva. Desde entonces las redes de Petri se han utilizado para modelar, visualizar y analizar procesos, y diseñar abstracciones en un amplísimo abanico de disciplinas técnicas y de organización. Se pueden encontrar aplicaciones en Automática, Administración de Empresas, Química, Informática, Diseño y Control de Sistemas de Fabricación, Ingeniería de Procesos y muchas más.

Las redes de Petri constituyen una herramienta excelente para el estudio de cualquier sistema, ya que permiten modelar el comportamiento y la estructura de éste, llevándolo a condiciones límite, que en un sistema real pueden ser inabordables por difíciles de producir o por muy costosas.

Especialmente relacionado con el trabajo que se presenta, aunque las redes de Petri no constituyen el único modelo de descripción de sistemas discretos capaz de modelar y analizar las evoluciones paralelas, se ha que encontrado que las redes de Petri representan explícitamente aspectos fundamentales de los sistemas distribuídos, como son la atomicidad, la sincronización, la independencia mutua de acciones, los mensajes y la memoria compartida [Rei98]. Es más, los aspectos básicos de los sistemas distribuídos se identifican tanto conceptualmente como matemáticamente con las redes de Petri [BDJ+00].

Por otro lado, la teoría de Redes de Petri ha llegado a ser reconocida como una metodología establecida en la literatura de la robótica para modelar sistemas de fabricación flexibles. De esta manera, el uso de las redes de Petri surge de manera natural como una buena elección para modelar algoritmos de cálculo paralelo del C-espacio para estructuras robóticas.

A continuación, se procede a dar la definición de las redes de Petri como modelos de descripción del funcionamiento de sistemas discretos concurrentes, según aparece en [Sil85]. Para

aquellos lectores que estén interesados en su descripción formal se les remite al capítulo segundo de [Sil85].

Definición de Red de Petri

Una *red de Petri* (PN) es un grafo orientado en el que intervienen dos clases de nodos, los *lugares* (representados por circunferencias) y las *transiciones* (representadas por segmentos rectilíneos), unidos alternativamente por *arcos*. Un arco une un lugar con una transición, o viceversa, pero nunca dos transiciones o dos lugares.

Un lugar puede contener un número positivo o nulo de *marcas*. Una marca se representa por un punto en el interior del círculo correspondiente al lugar. El conjunto de marcas asociadas en un instante dado a cada uno de los lugares constituye un marcado de la PN.

Para la descripcción funcional de los sistemas concurrentes, a los lugares se les asocian *acciones* o *salidas* del sistema que se desea modelar. A las transiciones se les asocian los *eventos* (funciones lógicas de las variables de entrada al sistema) y acciones o salidas.

De esta forma, el marcado puede evolucionar como sigue:

- Un lugar p es *lugar de entrada* de una transición t si existe un arco orientado de p hacia t;
- Un lugar p es *lugar de salida* de una transición t si existe un arco orientado de t hacia p;
- Una *transición está sensibilizada* si todos los lugares de entrada están marcados.
- Una transición sensibilizada es *disparada o franqueada* si el evento que le está asociado se verifica. El disparo de una transición consiste en quitar una marca a cada uno de los lugares de entrada y en añadir una marca a cada uno de los lugares de salida.

La figura 5.2 se muestra la representación gráfica de una PN con con cinco lugares y cuatro transiciones. El lugar p_1 contiene una marca, mientras que el lugar p_3 contiene dos marcas. Los

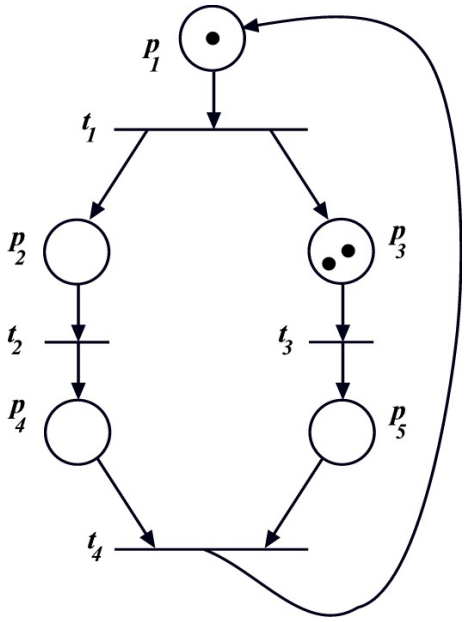


Figura 5.2: Ejemplo de Red de Petri

lugares p_4 y p_5 son de entrada para t_4, mientras que p_2 y p_3 son lugares de salida de t_1[4].

Justificación del Uso de las Redes de Petri para el Modelado de Sistemas Concurrentes

Manuel Silva, en [Sil85], realiza los siguientes comentarios sobre la aplicación de las PN para modelar sistemas concurrentes:

- Se trata de una herramienta de modelado clara, fácil de utilizar y no ambigua.
- Facilita la representación de evoluciones simultáneas. Como consecuencia, son flexibles, facilitando las modifica-

[4] Los lugares están representados con la letra p, porque es el modo habitual de hacerlo; tanto en inglés como en francés, a los lugares se les denomina *place*.

ciones locales, y permiten refinamientos sucesivos (permite descripciones de forma descendente).

- Permite una primera aproximación al problema de la validación del correcto funcionamiento del sistema (comprobando que la descripcción verifica una serie de propiedades de buen funccionamiento como, por ejemplo, ausencia de bloqueos y conflictos).
- La PN constituye una herramienta de modelado independiente de cualquier tecnología

Redes de Petri Coloreadas

A pesar de la potencia de las redes de Petri, el intento de usar las redes de Petri en la práctica descubrió dos problemas: 1) no existía el concepto de datos y por tanto los modelos se convertían en excesivamente grandes, ya que toda la manipulación de los datos tenía que representarse directamente en la estructura de la red —mediante lugares y transiciones—; y 2) no existía el concepto de jerarquía, y así, era imposible construir un modelo grande mediante un conjunto de submodelos separados con interfaces bien definidas.

El desarollo de las redes de Petri de alto nivel a finales de los 70, y la aparición de las redes de Petri jerárquicas a finales de los 80, eliminaron estos problemas.

Las redes de Petri Coloreadas (*Coloured Petri Nets*, CPN o *CP-nets*) pertenecen a la familia de las redes de alto nivel[5]; el paso a este tipo de redes desde las redes de bajo nivel se podría comparar al que se dió desde los lenguajes ensamblador hasta los lenguajes de programación modernos, con un concepto de tipo elaborado [Jen97b]. En las redes de Petri sólo hay un tipo de marca, lo que significa que el estado de un lugar está descrito por un entero —en muchos casos incluso por un valor booleano—. En las redes de alto nivel, cada marca puede llevar datos o información compleja (colores).

La razón de la aparición de las CPN fue el deseo de desarrollar un lenguaje de modelado, que estuviera bien fundamentado teóricamente, pero que fuera lo suficientemente versátil para

[5]Las CPN, introducidas por Kurt Jensen en 1979, constituyen uno de los dos dialectos más conocidos de las redes de Petri de alto nivel e incorporan estructuración de datos y descomposición jerárquica.

usarlo en la práctica para sistemas de gran complejidad y tamaño. Para conseguir esto, se combina la fuerza de las redes de Petri con la de los lenguajes de programación: las primeras aportan las primitivas para describir la sincronización de procesos concurrentes, mientras que los últimos proporcionan las primitivas para la definición de tipos de datos y la manipulación de los valores de los mismos [Jen98].

Por otro lado, las CPN se pueden extender facilmente para contemplar el concepto tiempo. Así, éstas se convierten en una excelente herramienta para la evaluación del rendimiento, es decir, evaluar la velocidad a la que un determinado sistema opera.

En [Jen97a] se da la siguiente descripción de las CPN (para una definición formal considérese [Jen97b]):

- Los círculos se denominan *lugares* y describen estados del sistema;
- Los rectángulos se llaman *transiciones* y describen las acciones;
- Las flechas se denominan *arcos*. Y las *expresiones de los arcos* describen cómo cambia el estado de la CPN cuando ocurren las transiciones;
- Cada lugar contiene un conjunto de *marcas*, cada una de las cuales lleva un valor de un dato perteneciente a un *tipo* determinado;
- Los colores de las CPN consisten en la siguiente analogía: un conjunto de colores es un tipo y el color de una marca su valor;
- Para que una transición *ocurra o se dispare*, ésta debe tener suficiente número de marcas en sus lugares de entrada, y los valores de las marcas deben emparejar con las expresiones de los arcos. Cuando la expresión de un arco de entrada se evalúa tomando el valor de una marca presente en el lugar de entrada correspondiente se dice que se *atan* las variables a valores de su tipo;
- Cuando existe alguna *atadura* se dice que la transición está *habilitada*;

- La elección entre varias ataduras es no determinista;
- Cuando se elige una atadura y la transición se dispara, la(s) marca(s) se eliminan de los lugares de entrada y se colocan en los lugares de salida.
- Se pueden utilizar *guardas*. Una transición sólo se disparará si todas sus guardas se evalúan a verdadero.

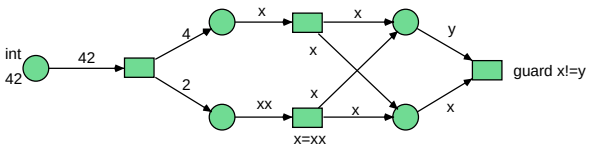


Figura 5.3: Ejemplo de Red de Petri Coloreada

En la figura 5.3 se puede ver un ejemplo sencillo de red de Petri coloreada[6], que simplemente divide el número 42 en sus dos dígitos y propaga ambos por dos caminos cada uno (usando una copia de cada dígito), hasta una transición que espera a que le lleguen, por cualquier camino, dos dígitos difrentes. A la izquierda hay un lugar con el tipo entero (`int`), lo que significa que sólo puede tomar marcas de enteros. Inicalmente el marcado es la marca entera `42`. Los otros lugares no tienen tipo asociado e inicialmente están sin marcado. La transición más a la izquierda quitará `42` del lugar incial y colocará un `4` y un `2` en los lugares correspondientes. La transición superior del centro recibe una `x`, que resulta ser un `4` en este caso, lo elimina de su lugar de entrada y lo coloca en sus dos lugares de salida. La transición inferior del centro es similar, pero en este caso, la igualdad de los arcos de entrada y salida se establece mediante la inscripción `x=xx`. La última transición tienen una guarda, `guard x!=y`, que asegura $x \neq y$. De esta forma, sólo

[6]La Universidad de Aarhus (Dinamarca), bajo la dirección de Kurt Jensen, ha desarrollado *Design/CPN* —y su sucesora, *CPN Tools*—, una de las herramientas más completas para redes de Petri. No obstante, la red de la figura se construyó con *Renew*, la herramienta que se ha utilizado para evaluar el diseño de los algoritmos paralelos de este trabajo, y que se presenta en la siguiente sección.

puede tomar un 2 del lugar de arriba y un 4 del lugar de abajo o viceversa.

Análisis de Redes de Petri Coloreadas

Durante la construcción de una CPN se utilizan simulaciones para validar el modelo, es decir, para comprobar que se comporta como cabría esparar. Es habitual trabajar de forma iterativa. En las primeras fases, el modelo es simple y cubre sólo algunas partes del sistema, ignorando muchos aspectos del sistema final. Finalmente, el alcance del modelo se extiende y se añaden más detalles. Mediante sucesivas simulaciones durante todo el proceso de diseño —no solamente al final— el modelador aprende sobre el sistema, permitiendo eliminar errores de diseño e incorporar nuevos conocimientos sobre el sistema en etapas posteriores del proceso de diseño.

5.2 Herramientas

La presente 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. Se trata de dos bibliotecas de funciones que persiguen objetivos bien diferenciados: una, MPI, es una implementación del paradigma de paso de mensajes que, como se ha explicado previamente, es un modelo de programación paralela muy extendido; la otra, FFTW, es un biblioteca que permite, a través de múltiples funciones, la ejecución de la Transformada Rápida de Fourier en varias dimensiones. Como ya se ha visto en el capítulo anterior, la transformada de Fourier se debe utilizar intensivamente para la evaluación del espacio de las configuraciones de estructuras robóticas.

Además, se hace una breve presentación de Renew, una herramienta gráfica que permite construir redes de Petri coloreadas y realizar simulaciones con ellas.

Paro los tres casos, se van a comentar sus principales características, así como las ventajas que aportan y que han determinado la elección de estas herramientas para la implementación de los algoritmos desarrollados.

Por otro lado, a lo largo del trabajo también se han utilizado otras herramientas, como MATLAB, para realizar representación de resultados o algunos cálculos de apoyo —como es el caso del cálculo simbólico— que no se comentarán, por no considerarlas fundamentales en el desarrollo del trabajo que se presenta.

5.2.1 MPI

En el enfoque de la biblioteca de paso de mensajes MPI (*Message Passing Interface*) para la programación paralela, un conjunto de procesos ejecutan programas en un lenguaje secuencial estándar mejorado con una serie de llamadas a funciones de una biblioteca para el envío y recepción de mensajes.

MPI Forum

Formado por más de 80 personas de 40 organizaciones, que representan a vendedores de sistemas paralelos, usuarios industriales, laboratorios de investigación nacionales y universidades, es el responsable del éxito y la aceptación mundial de este estándar de paso de mensajes.

Actualmente, MPI —desarrollada por el *Message Passing Interface Forum* [Tea98]— se ha convertido en el estándar *de facto* para el paso de mensajes. Se puede decir que es un sistema complejo, que contiene cerca de 200 funciones, la mayoría de las cuales tiene numerosos parámetros y variantes.

En el modelo de programación de MPI, un cálculo comprende uno o más procesos que se comunican haciendo llamadas a rutinas de biblioteca para enviar y recibir mensajes de otros procesadores. En la mayoría de las implementaciones de MPI, se crea un número fijo de procesos en la inicialización del programa, y se crea un proceso por procesador. Sin embargo, estos procesos pueden ejecutar programas diferentes. Por tanto, el modelo de programación de MPI se engloba dentro de los Múltiples Programas Múltiples Datos (MPMD) para distinguirlo del modelo SPMD, en donde cada procesador ejecuta el mismo programa.

Dado que el número de procesos en un cálculo de MPI normalmente está prefijado, la característica principal a estudiar es el mecanismo a través del cual se comunican datos entre los procesos. Los procesos pueden utilizar operaciones de comunicación punto a punto para enviar un mensaje de un proceso dado a otro.

Como se ha expuesto en secciones anteriores, el paradigma de programación de paso de mensajes está muy extendido en los computadores paralelos, especialmente los Computadores Paralelos Escalables (*Scalable Parallel Computers*, SPCs) con memoria distribuida y en las Redes de Estaciones de Trabajo

(*Networks of Workstations*, NOWs). Aunque existen múltiples variaciones, el concepto básico de comunicación de procesos mediante mensajes es bien conocido. Durante los última década se han obtenido progresos importantes mediante el paso de aplicaciones significativas a este paradigma. Cada desarrollador ha implementado su propia versión. Sin embargo, recientemente, varios sistemas de dominio público han demostrado que se puede implementar un sistema portable de paso de mensajes de forma eficiente. Es la época en que se puede definir tanto la semántica como la sintaxis de un núcleo estándar de rutinas de biblioteca que pueden ser útiles a un abanico grande de usuarios y que puede ser llevado de forma eficiente a un rango amplio de computadores[7].

Desde la aparición del estándar MPI en junio de 1994, MPI se ha aceptado y utilizado mundialmente. Y de este modo existen implementaciones que van desde aquéllas para los grandes computadores paralelos escalables, hasta las existentes para las redes de estaciones de trabajo. Incluso varios vendedores de SPCs proporcionan y respaldan sus propias implementaciones de MPI, lo que incorpora credibilidad al procesamiento paralelo.

El principal objetivo de MPI, como ocurre con la mayoría de los estándares, es alcanzar el grado suficiente de portabilidad en las diferentes máquinas. Y este grado de portabilidad deseable se corresponde con el alcanzado por lenguajes de programación como Fortran o C. Esto permitiría que el mismo código fuente de paso de mensajes se podría ejecutar en cualquier máquina con la única condición de que la biblioteca MPI esté presente, aunque para obtener el mayor rendimiento sean necesarios algunos ajustes que permitan favorecerse de las características particulares de cada sistema.

Aunque el paso de mensajes se suele enmarcar en el contexto de los computadores paralelos de memoria distribuida, el mismo código puede funcionar bien en computadores de me-

[7] Los diseñadores de MPI trataron de incorporar las características más atractivas de una serie de sistemas de paso de mensajes existentes, en lugar de elegir uno de ellos como estándar. De este modo, MPI se ha beneficiado de la experiencia del centro de investigación T.J. Watson de IBM, NX/2 de Intel, Express, Vertex de nCUBE, p4, y PARMACs, así como de las importantes contribuciones de Zipcode, Chimp, PVM, Chamaleon, y PICL.

moria compartida, una red de estaciones de trabajo o, incluso, en una estación de trabajo con varios procesos ejecutándose.

Otro tipo de compatibilidad que ofrece MPI es la capacidad de ejecución transparente en sistemas heterogéneos, es decir, un conjunto de procesadores con diferentes arquitecturas. Gracias a MPI se tiene un modelo de cálculo virtual que oculta muchas diferencias arquitectónicas. De este modo los usuarios no tienen que preocuparse de si la aplicación está enviando mensajes entre procesadores similares o diferentes ya que la implementación de MPI se encargará de hacer automáticamente cualquier conversión de datos y de utilizar el protocolo de comunicaciones adecuado.

A pesar de que la portabilidad era un objetivo primordial, MPI no se convertiría en un buen estándar si ésta se consiguiera en detrimento del rendimiento. Así, por ejemplo, el lenguaje Fortran se utiliza frente a lenguajes ensambladores porque la mayoría de los compiladores disponibles alcanzan un rendimiento aceptable frente a la alternativa no portable de los lenguajes ensambladores. El diseño de MPI ha demostrado ser el adecuado dado el alto rendimiento alcanzado con las implementaciones en un amplio rango de plataformas comparado con el que alcanzan los sistemas específicos de vendedor, que además son menos portables.

Para alcanzar implementaciones eficientes, MPI sólo especifica qué es lo que una operación hace lógicamente, evitando detallar cómo tienen lugar estas operaciones. Como resultado de esta decisión, MPI se puede implementar fácilmente en sistemas que almacenan mensajes en el emisor, en el receptor o que no los almacenan. Las implementaciones se aprovechan de las distintas propiedades de los subsistemas de comunicaciones de las diferentes máquinas. Así, en máquinas con coprocesadores de comunicaciones inteligentes, la mayor parte del protocolo de paso de mensajes se puede descargar a estos coprocesadores. En otros sistemas, la mayor parte del código de comunicaciones se ejecuta por el procesador principal.

Además, MPI se ha diseñado para evitar el trabajo innecesario que podría degradar el rendimiento. De este modo, se evita la necesidad de grandes cantidades de información extra en cada mensaje, o de una compleja codificación y decodificación de las cabeceras de los mensajes.

Una característica muy interesante en el diseño de MPI

es la capacidad de solapamiento de las comunicaciones y los cálculos, para aprovecharse de los agentes de comunicaciones inteligentes, y para esconder las latencias de las comunicaciones. Esto se consigue mediante el uso de llamadas de comunicaciones no bloqueantes, lo que separa la iniciación de las comunicaciones de la finalización.

También, la escalabilidad es un objetivo importante en la programación paralela; este objetivo se consigue a través de varias características del diseño de MPI. Por ejemplo, una aplicación puede crear subgrupos de procesos que, de uno en uno, se van comunicando mediante operaciones colectivas que limitan el alcance a aquellos procesos involucrados. Otra técnica permite proporcionar funcionalidad sin un cálculo que escale en función del número de procesos. Por ejemplo, una topología Cartesiana de dos dimensiones se puede subdividir en filas y columnas sin tener que enumerar explícitamente los procesos.

Finalmente, como todos los buenos estándares, está definido un comportamiento mínimo y conocido en todas las implementaciones. Esto libera al programador de tener que preocuparse de determinados problemas. Un ejemplo es que MPI garantiza que la transmisión de mensajes subyacente es fiable. Así, el usuario no tiene que comprobar si un mensaje se ha recibido correctamente.

Resumiendo, las principales características [Tea96] por las que se ha elegido MPI —como la herramienta más adecuada para implementar la versión paralela de los algoritmos para la deconstrucción del espacio de las configuraciones que se presentan en el siguiente capítulo— son:

- Está diseñada como una interfaz de programación de aplicaciones. Aunque MPI se utiliza como una biblioteca run-time, está diseñada principalmente para satisfacer las necesidades de un programador de aplicaciones.

- Permite comunicaciones eficientes. Evita copias memoria-a-memoria, permite solapar cálculo y comunicaciones y descarga a un coprocesador de comunicaciones, cuando es posible.

- Permite implementaciones que se puedan utilizar en entornos heterogéneos.

- Permite enlazar con el lenguaje C.
- Proporciona comunicaciones fiables. El usuario no tiene que tratar fallos de comunicaciones.
- Define una interfaz muy similar a otras bien conocidas, como PVM, NX, Express, p4, etc., proporcionando extensiones que añaden flexibilidad.
- MPI cuanta con una herramienta muy útil para la ejecución y el depurado de programas paralelos denominada XMPI[8].

Diagramas de Gantt

XMPI utiliza el siguiente código de colores: verde, el proceso se encuentra trabajando en algún cálculo ajeno a MPI; rojo, el proceso está bloqueado, es decir, no puede seguir realizando ningún cálculo porque se encuentra a la espera de algún dato que tiene que recibir de otro proceso; y amarillo, el proceso se encuentra trabajando en una función de MPI.

De todo lo dicho hay que resaltar que una de los principales atractivos de MPI, teniendo en mente su aplicación al caso que nos ocupa, es que se puede desarrollar un algoritmo paralelo, y con el mismo código fuente, sin necesidad de incorporar modificaciones, se puede generar un programa ejecutable en multicomputadores de memoria distribuida, multiprocesadores de memoria compartida, redes de estaciones de trabajo y combinaciones de todos ellos.

Finalmente, es importante el hecho de que existen múltiples implementaciones de MPI, de entre las cuales se optó por una de las más utilizadas, LAM MPI; además, cabe señalar que se utilizó la versión 6.5.4, desarrollada por la Universidad de Notre Dame[9].

5.2.2 Renew

Renew [KWD01] —abreviatura de *Reference Net Workshop* (Taller de Redes de Referencia)— es una herramienta, desarrollada en la Universidad de Hamburgo y escrita en Java,

[8] XMPI es una interfaz gráfica de usuario basada en X/Motif que se encarga de recopilar una gran cantidad de datos procedentes de la ejecución de una aplicación, o de unos registros acumulativos de las comunicaciones. En este diagrama se puede observar la evolución temporal de la ejecución de una aplicación así como el modo en que han tenido lugar las comunicaciones entre los distintos procesos.

[9] Originalmente desarrollado en el Ohio Supercomputer Center, pasó después a mantenerse en la Universidad de Notre Dame hasta el otoño de 2001, que pasó a albergarse en la Universidad de Indiana; actualmente el equipo de desarrollo de LAM sólo ofrece soporte de la versión 6.5.6.

que permite modelar y analizar sistemas con redes de Petri coloreadas (ver figura 5.3). Se trata de una herramienta abierta y muy versátil.

Las principales caracteristicas de Renew se pueden resumir en los siguientes puntos:

- Renew puede utilizarse en los principales sistemas operativos modernos sin ningún cambio.
- Se distribuye libremente incluyendo el código fuente, con lo que sus algoritmos se pueden extender y mejorar en caso de necesidad.
- Puede utilizar cualquier clase de Java, ventaja muy interesante, ya que hoy en día existen clases de Java que cubren casi cualquier aspecto de programación.
- Las propias redes de referencia son objetos de Java. Esto permite hacer llamadas desde código Java a las redes y viceversa.

Por otro lado el formalismo de red de Petri que usa Renew presenta algunos aspectos interesantes

- Soporta canales síncronos. Los canales son un mecanismo de comunicaciones muy potente (el emisor y el receptor deben de estar de acuerdo en participar en una comunicación en un determinado momento[10]) que se pueden utilizar de un modo fiable.
- Las instancias de red permiten modelado orientado a objetos con redes de Petri.
- Proporciona varios tipos de arcos que cubren la mayoría de los formalismos de red existentes.
- Una capacidad clave de las redes de referencia es que incorpora en su formalismo mecanismos para considerar

[10] En el formalismo de las redes de Petri se contempla otro mecanismo, que normalmente recibe el nombre de *lugares de fusión*; se trata de el *paso de mensajes*, en el que el emisor puede enviar un mensaje en cualquier momento, mientras que el receptor puede estar o no preparado para procesarlo; si es necesario, se puede reproducir el paso de mensajes con los canales síncronos.

> seriamente el tiempo. Aunque las redes de Petri puras capturan adecuadamente la causalidad y las situaciones de conflicto, para sistemas físicos, como es el caso del modelado de un conjunto de procesadores trabajando cooperativa y concurrentemente, es esencial añadir una noción de tiempo al formalismo.

Se puede añadir que Renew no incorpora mecanismos para el análisis de las redes, por lo que estos se deben hacer mediante la exportación de las redes diseñadas a formatos de otras aplicaciones que sí dispongan de ellos. Sin embargo, se trata de una herramienta muy útil en simulación, ya que permite explorar dinámica e interactivamente el estado de la simulación.

Todas estas características hacen a esta herramienta una opción muy interesante para validar los algoritmos presentados en el capítulo anterior, asi como para ayudar al diseño y la validación de sus versiones paralelas. Además, para el presente trabajo, el análisis de las redes no es una parte primordial, si no que se elige utilizar redes de Petri, además de por su capacidad de simulación, por su representación gráfica intuitiva, su expresividad y su semántica precisa.

5.2.3 FFTW

Como es bien conocido, la Transformada Discreta de Fourier (*Discrete Fourier Transform*, DFT) se utiliza ampliamente para realizar análisis frecuenciales de señales no periódicas. 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.

El algoritmo para el cálculo de la DFT es bastante complicado, ya que implica la realización de múltiples sumas y multiplicaciones con números complejos[11].

La idea subyacente en la FFT es el conocido enfoque *divide y vencerás*, ya que rompe la muestra original de N puntos en dos secuencias de $\frac{N}{2}$. Así, la DFT requiere $(N-1)^2$ productos complejos y $N(N-1)$ sumas complejas frente al enfoque

[11] 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 20000000 de sumas y productos complejos

de la FFT que rompe la señal en una serie de muestras de 2 puntos que simplemente requieren un producto y una suma y la recombinación de los puntos.

De esta manera, resulta natural pensar en una versión paralela del algoritmo para el cálculo de la Transformada Rápida de Fourier, ya que en el mejor de los casos un elemento de proceso se podría dedicar al cálculo de cada uno de los DFTs de dos muestras. En caso de no disponer de tantos elementos de proceso, existen diferentes formas de repartir el trabajo entre los procesadores disponibles [Akl92][KGGK94].

Existen múltiples versiones paralelas del algoritmo para el cálculo de la FFT, tanto de libre distribución como propiedad de algún vendedor. Igualmente, la forma de implementar el algoritmo paralelo es variada, desde el uso de *threads* o el uso de paso de mensajes hasta algoritmos desarrollados con lenguajes especializados como el lenguaje paralelo Cilk, desarrollado en el Instituto Tecnológico de Massachusetts (MIT).

Entre todas estas posibilidades se han hecho pruebas con varias de las bibliotecas disponibles. Entre ellas, destacan la biblioteca FFTW, desarrollada por el MIT, y que como se muestra en la figura 5.4 presenta un rendimiento óptimo (usando Mflops[12] 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.

Sin embargo, se optó por llevar a cabo todas las pruebas del presente trabajo de investigación con la biblioteca de funciones que proporciona FFTW, por tratarse de un software de libre distribución, que facilitaría la consecución de uno de los principales objetivos de la investigación: la producción de algoritmos portables a todo tipo de arquitecturas paralelas.

Otro factor de peso a la hora de elegir a FFTW como la candidata idónea es el hecho de que proporciona versiones paralelas de las funciones de cálculo de la FFT en tres versiones: uso *threads*, uso de MPI y uso de Cilk, lo que permite un mayor campo para experimentar y mejorar los tiempos de cálculo.

[12] Mega-operaciones de coma flotante por segundo.

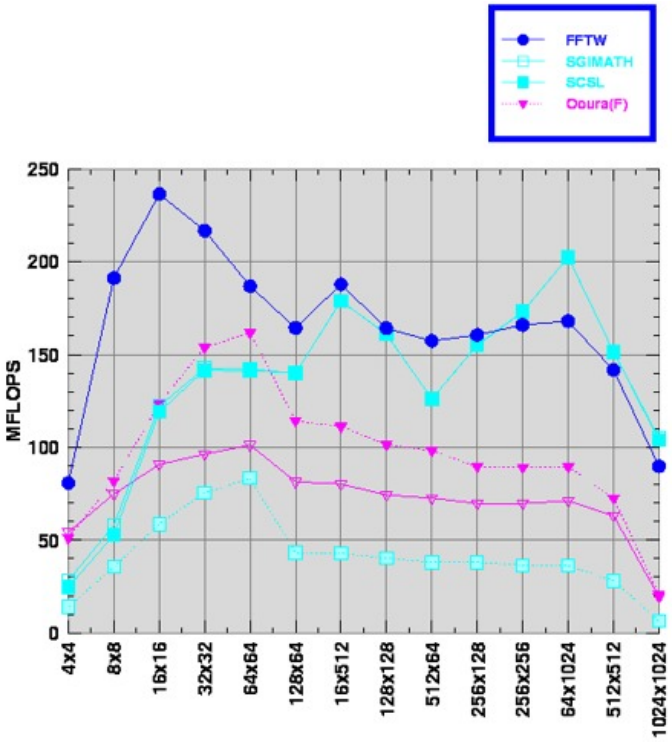
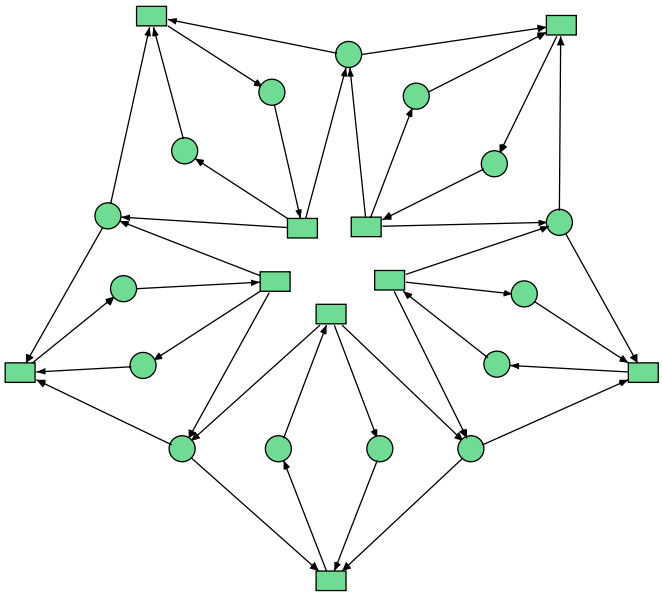


Figura 5.4: Transformadas de Fourier reales en 2D, Mflops para diferentes tamaños (potencia de dos) de muestra

Capítulo 6

Diseño e Implementación Paralela



Si haces mal el trabajo, a veces no te piden que vuelvas a hacerlo.

Felino Maníaco Homicida
BILL WATTERSON

COMO YA SE HA PLANTEADO en la introducción, uno de los propósitos de este trabajo es analizar el formalismo propuesto para la deconstrucción del espacio de las configuraciones en busca de oportunidades de paralelización. Una vez hecho esto de forma genérica, se proponen soluciones paralelas derivadas de los algoritmos presentados en el capítulo 4. Finalmente, utilizando las redes de Petri coloreadas, se elabora un estudio sobre cómo se comportan estos algoritmos cuando se implementa su paralelización, comprobando en qué medida afectan las distintas decisiones de diseño y verificando si el esfuerzo de realizar una implementación paralela se ve justificado por unos resultados alentadores.

Dadas las limitaciones de espacio, solamente se va a presentar el diseño e implementación detallados del caso más complejo de entre los expuestos en el capítulo dedicado a ilustrar la deconstrucción del C-espacio: un robot PUMA de cuatro articulaciones que se mueve en un espacio tridimensional.

Para ello, se procederá en un primer apartado al análisis pormenorizado de las correspondientes expresiones formales y el algoritmo propuestos para este caso. A continuación, se identifican los distintos niveles de paralelismo presentes en el cálculo del C-espacio. Posteriormente, se evalúan los aspectos más relevantes para introducir el cálculo paralelo en el algoritmo. Finalmente, se evalúan, mediante simulación con Renew, tanto el correcto funcionamiento concurrente, como el rendimiento —en términos de tiempo de ejecución— esperado.

6.1 Niveles de Paralelismo

En esta sección se plantean los tres niveles de paralelismo encontrados en la deconstrucción del C-espacio. Los dos primeros se encuentran en el cálculo básico de la deconstrucción, esto es, el cálculo de cada **CB**; el tercer nivel aparece en la forma en que se desarrolla el proceso de deconstrucción.

6.1.1 Concurrencia Inherente al Cálculo de los CB

Como ya ocurría en el formalismo propuesto por [Cur98] —en [The00] se hace una exposición detallada de este estudio—, del análisis de la expresión general para el cálculo del espacio de las configuraciones de cada $\mathbf{CB}_k$ (y de su expresión asociada, 3.17, en la página 56) se deduce la presencia de dos[1] niveles de paralelismo que se pueden aprovechar —por separado o combinados— para aliviar la carga computacional requerida.

Los dos niveles que se pueden encontrar son los siguientes:

- **Nivel de cálculo de la Transformada de Fourier.** En este nivel, gracias al uso del algoritmo de la Transformada Rápida de Fourier (FFT), se puede optimizar el tiempo de cálculo mediante la utilización de varios procesadores que realicen las operaciones en paralelo.

- **Nivel de las variables espaciales.** En este nivel, tras el análisis de los algoritmos presentados, aparece la necesidad de realizar una serie de integrales que pueden llevarse a cabo en paralelo. Efectivamente, para todos los valores entre los que se hace la integral respecto a cada variable espacial $(x_{r+1}, \ldots, x_n)$ se calcular el producto de convolución, lo que delimita otro nivel de paralelismo.

[1] En el caso formalismo propuesto en [Cur98] había un tercer nivel asociado a las variables de configuración que no convolucionan. Éste no está presente en la deconstrucción, ya que para cada grado libertad se va encontrando convolución.

6.1.2 Concurrencia Inherente al Proceso de Deconstrucción

En este caso, la explotación del paralelismo es más compleja. El cálculo de cada $\mathbf{CB}_k$ no es independiente de los anteriores, salvo, obviamente, $\mathbf{CB}_1$. Por tanto, estos resultados previos deben estar disponibles, lo que implica una dependencia de datos que prohibe el cálculo simultáneo de todos los $\mathbf{CB}_k$.

Sin embargo, se podría calcular en paralelo cada $\mathbf{CB}_k$ si los datos previos están disponibles, en concreto, las configuraciones libres de los $k-1$ DOFs anteriores en la cadena cinemática, y, para cada una de éstas, los datos necesarios para transformar los puntos del espacio de trabajo —dar las coordenadas de cada punto respecto al sistema de referencia F_{k-1}.

Esto significa una de dos posibilidades: 1) Se va calculando cada $\mathbf{CB}_k$ por orden, realizando los cálculos pertinentes explotando uno o los dos niveles de paralelismo asociados al cálculo de los $\mathbf{CB}_k$; o 2) Se van conociendo y evaluando configuraciones libres de forma incremental (primero se conocen algunas configuraciones libres del DOF_1, e inmediatamente, sin esperar a conocer todo el C-espacio asociado al primer grado de libertad, se empieza a indagar las configuraciones libres de (DOF_1, DOF_2), y así sucesivamente).

El segundo método, llevado al límite, es, sin duda, muy complejo, pues es preciso establecer un conjunto muy complicado de intercambios de información entre todos los procesos involucrados. Sin embargo, como se explicará en las secciones posteriores, es el principio que se ha utilizado para implementar el caso de estudio que se presenta en este capítulo, aunque eso sí, de una forma simplificada y solamente con el primer grado de libertad.

6.2 Soluciones Paralelas Propuestas

A continuación se proponen distintas formas de aprovechar los niveles de paralelismo que pueden aparecer en la deconstrucción del espacio de las configuraciones para estructuras robóticas.

- **Nivel de Transformada de Fourier.** En este caso se tratará de optimizar el rendimiento mediante la utiliza-

ción de varios procesadores para el cálculo de las operaciones que implica el conocido algoritmo FFT. Así, lo que se propone es una estructura en la que un procesador se va a utilizar para el cálculo del algoritmo principal y se dedican una serie de procesadores al cálculo de transformadas en 2D de forma paralela.

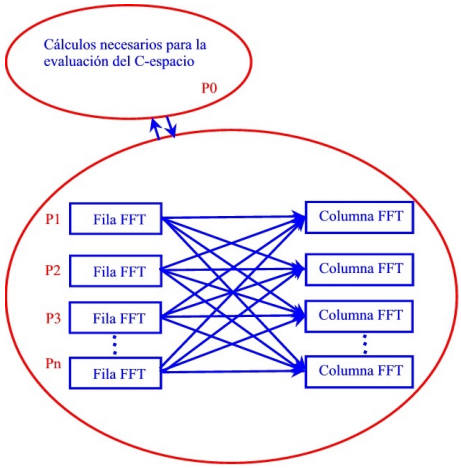


Figura 6.1: Aprovechamiento del nivel de paralelismo inherente a la FFT

La figura 6.1 ilustra este concepto, mientras que en la figura 6.2 se modela, de forma muy simplista, en una red de Petri. Se tiene un proceso central —normalmente equivalente a un procesador dedicado—, el lugar `P0` en la red, que se encarga de realizar los cálculos necesarios para la obtención del C-espacio; cada vez que se tiene que realizar una transformada de Fourier sobre unos datos, estos se envían a unos procesos, que realizan la FFT en paralelo[2], y devuelven los datos transformados al proceso central, que sigue calculando.

- **Nivel de variables espaciales.** En este nivel se va a tener que realizar la acumulación de los productos de

[2]El modo en que se lleva a cabo el cálculo paralalelo de las FFT no aparece modelado en esta red tan simple.

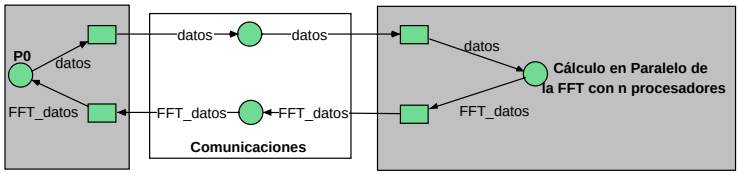


Figura 6.2: Nivel de paralelismo inherente a la FFT. Red de Petri coloreada

convolución, entre los límites definidos por la discretización elegida, respecto a una variable espacial. Así, lo que se propone es una solución muy similar a la anterior: un proceso se encarga de repartir entre cierto número de procesos los diferentes valores a calcular, recopilar los resultados intermedios y sumarlos para obtener el resultado final. En la figura 6.3 se ilustra esta solución propuesta.

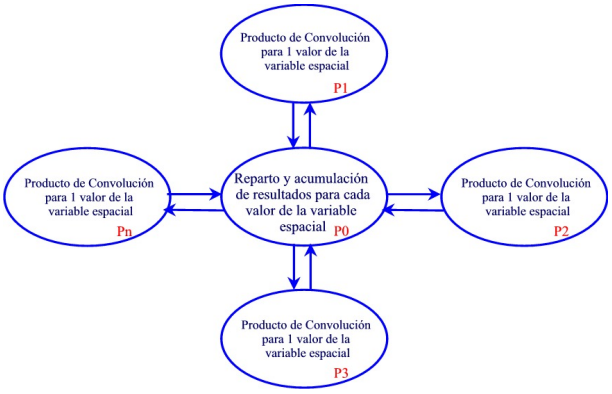


Figura 6.3: Aprovechamiento del nivel de paralelismo inherente a las variables espaciales

En la figura 6.4 se tiene la red de Petri coloreada que modela esta solución. Se tiene un lugar `P0` que va repartiendo a un número de procesos (8 en el ejemplo) valores de la variable espacial para los que se tiene que calcular

el producto de convolución. En el lado derecho de la red se ve cómo se tiene un banco de procesos desocupados (`1;2;3;4;5;6;7;8`, inicialmente) que esperan a que el proceso central les asigne tarea (`guard Proceso == Yo`), realizan el cálculo y envían el resultado al proceso central (`Proceso = Yo; P_conv = V_esp;`), y se colocan de nuevo en el banco de procesos libres.

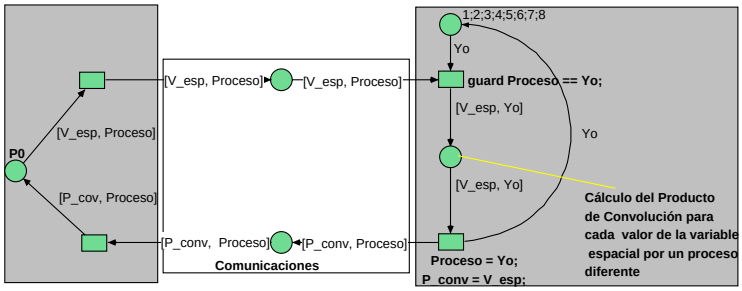


Figura 6.4: Nivel de paralelismo inherente a la FFT. Red de Petri coloreada

- **Convivencia del nivel de FFT y el nivel de variable espacial.** En este caso, se propone una solución paralela que trata de combinar los dos niveles de paralelismo presentes en el cálculo de $\mathbf{CB}_k$. Así, para producir una mejora en los tiempos de cálculo se vuelve a tener un procesador central dedicado a repartir trabajo y a recopilar resultados intermedios. A su vez, cada uno de los procesadores que se dedican al cálculo del producto de convolución para un determinado valor de una variable espacial, se comunicará con subgrupos de procesadores; cada uno de estos estará dedicado al cálculo de la FFT de forma concurrente. La figura 6.5 ilustra esta solución. La red de Petri coloreada correspondiente a este caso sería la de la figura 6.6, en donde, básicamente, el funcionamiento es el mismo que el de la figura 6.4, salvo que cada proceso, a su vez, cuando necesita realizar un cálculo de una FFT, lo solicita a un grupo de subprocesos que lo hace en paralelo. De nuevo, el algoritmo del cálculo paralelo

de la FFT no aparece reflejado.

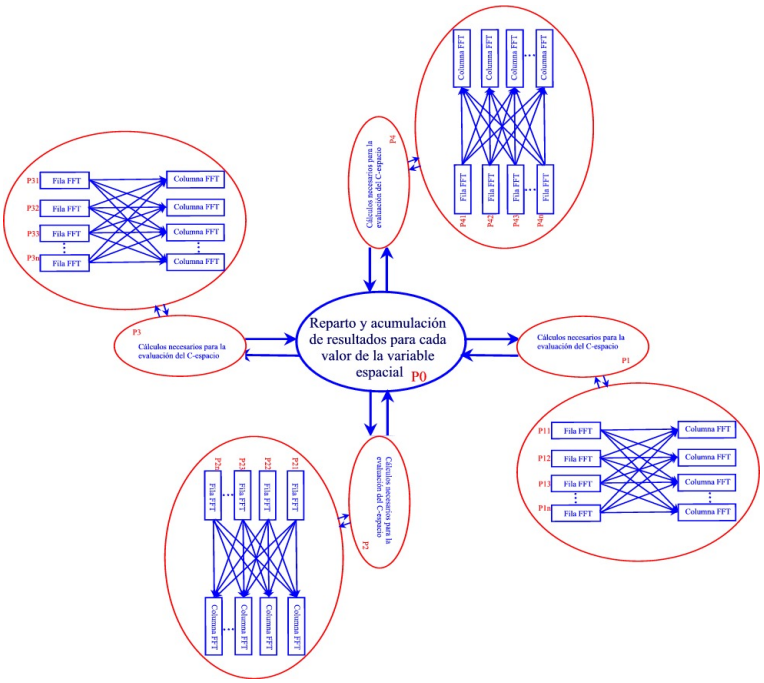


Figura 6.5: Convivencia de los dos niveles de paralelismo asociados al cálculo de $\mathbf{CB}_k$

- **Nivel inherente al proceso de deconstrucción y los otros niveles.** En este caso, como ya se ha comentado, habría que considerar dos posibilidades:

  1. Ir calculando cada $\mathbf{CB}_k$ por orden, lo que en un modelo de CPN, se corresponde a considerar que el proceso central va calculando todos los $\mathbf{CB}_k$ y encarga a otros procesos las tareas de las figuras 6.2, 6.4 o 6.6, según se explote el nivel de la FFT, el nivel de variables espaciales o ambos, respectivamente.
  2. Se van conociendo y evaluando configuraciones libres de forma incremental, lo que supondría un modelo de CPN muy complejo.

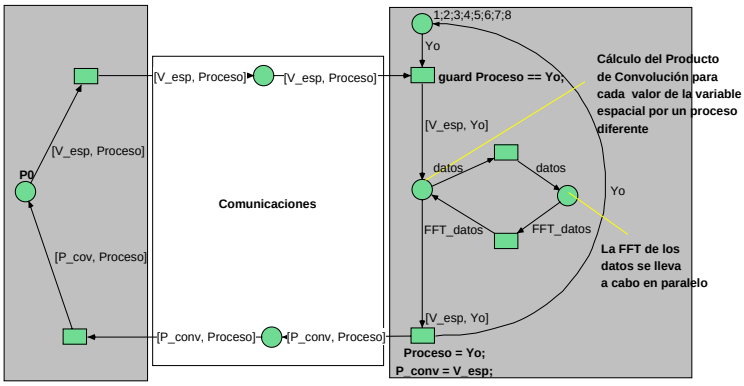


Figura 6.6: Convivencia de los dos niveles de paralelismo asociados al cálculo de CB_k

Es importante comentar que en todas las soluciones anteriores no se han tenido en cuenta cuestiones primordiales que validarían estos modelos propuestos, como es el caso de posibles cuellos de botella debido a la gran demanda que puede llegar a tener el proceso central. Se puede llegar a la situación de que un gran número de procesadores estén sin utilizarse por encontrase a la espera de que el proceso central les proporcione datos que les permita continuar su ejecución. Existen posibles soluciones que habría que investigar. Una alternativa consistiría en la creación dinámica de procesos, guiada por un proceso central, que en el momento en que se requiera hacer un cálculo, compruebe que hay procesadores libres, en lugar de crear los procesos inicialmente y permitir que éstos infrautilicen procesadores. Otra situación sería la utilización de varios procesos centrales que evitarían el exceso de demanda a un único proceso y su eventual sobrecarga que limitaría el comportamiento global. Sin embargo, esto queda fuera del alcance de este trabajo.

Se ha pretendido en esta sección establecer un punto de partida para la consecución de algoritmos óptimos que realicen el cálculo del espacio de las configuraciones para todo tipo de estructuras robóticas.

6.3 Caso de Estudio

Como se ha dicho, el problema es la deconstrucción en paralelo del cálculo del C-espacio para un robot redundante de tipo PUMA (figura 4.17 en la página 97).

En concreto, se elige este caso porque contiene la mayor complejidad de entre los presentados en el capítulo 4 y por tanto cubre toda la casuística que puede aparecer en ellos, además de servir como referencia para cualquier caso más complicado.

Como punto de partida se tienen las expresiones a las que se llega tras aplicar el formalismo. Así, la evaluación del C-espacio se compone de la evaluación de cuatro funciones.

$$\begin{array}{l} CB_1(\theta_1) \\ CB_2(\theta_1,\theta_2) \\ CB_3(\theta_1,\theta_2,\theta_3) \\ CB_4(\theta_1,\theta_2,\theta_3,\theta_4) \end{array} \tag{6.1}$$

que se evalúan como sigue:

$$CB_1(\theta_1)=0 \tag{6.2}$$

$$\mathcal{F}[CB_2(\theta_1,\theta_2)]=\int \mathcal{F}\left[\bar{A}_{1_{(0,0)}}(r,\theta_1,\theta_2)\right]_{(\varphi_1,\varphi_2)} \mathcal{F}[B(r,\theta_1,\theta_2)]_{(\varphi_1,\varphi_2)} dr \tag{6.3}$$

$$\mathcal{F}[CB_3(\theta_1,\theta_2,\theta_3)]=\int \mathcal{F}\left[\bar{A}'_{3_{(0)}}(r',\theta_3)\right]_{\varphi'_1} \mathcal{F}\left[B'(r',\theta_3)\right]_{\varphi'_1} dr' \tag{6.4}$$

$$\mathcal{F}[CB_4(\theta_1,\theta_2,\theta_3,\theta_4)]=\int \mathcal{F}\left[\bar{A}'_{4_{(0)}}(r'',\theta_4)\right]_{\varphi''_1} \mathcal{F}\left[B''(r'',\theta_4)\right]_{\varphi''_1} dr'' \tag{6.5}$$

Además, no debe olvidarse que se deben transformar los puntos del espacio de trabajo según las fórmulas a las que se llegó en el capítulo anterior. Es decir, se deben calcular (r', φ'_1) a partir de $r, \varphi_1, \varphi_2, l_2, \theta_1$ y θ_2; y (r'', φ''_1) a partir de r', φ'_1, l_3 y θ_3.

Por lo demás, como se puede observar en las expresiones 6.3, 6.4 y 6.5, aparecen los dos niveles de paralelismo identificados para el cálculo de cada $\mathbf{CB}_k$. El primero de ellos, a

un nivel más bajo, aparece por el uso de la FFT para realizar las transformadas de Fourier. El segundo nivel de paralelismo viene determinado por la necesidad de realizar las integrales acumulando valores para las variables espaciales: r, r' y r''.

Por otro lado, también se pueden realizar los cálculos de forma concurrente, explotando el modo en que se realiza la deconstrucción en este caso. Para ello, basta examinar el algoritmo propuesto en las tablas 4.10 y 4.11: primero se calcula CB_2, es decir, se encuentran las configuraciones (θ_1, θ_2) que producen colisión entre el elemento $\mathbf{A}_2$ y obstáculos del espacio de trabajo. En este momento, para las c configuraciones libres —par de valores (θ_1, θ_2) que no producen colisión— se ha de calcular las colisiones debidas al resto de elementos en la cadena cinemática del robot. Estas c operaciones son independientes entre sí, lo que permitiría realizar el trabajo de calcular las colisiones debidas al resto de los eslabones para cada configuración $(\theta_1 = \alpha, \theta_2 = \beta)$ de forma concurrente.

Tomando como base la experiencia adquirida en la parelización de códigos similares [The00][TBCM01][TBC+02], se considera que, cuando existe una limitación en el número de procesadores disponibles, no es conveniente explotar varios niveles simultáneamente.

De todas las opciones paralelas propuestas, quedarían entonces tres posibilidades de diseño: explotar el nivel de FFT, explotar el nivel de variables espaciales o explotar el nivel asociado al proceso de deconstrucción.

De éstas, la primera se desestimó porque el número de transformadas de Fourier que hay que realizar es muy elevado, y el tiempo de cálculo individual no es demasiado grande comparado con el tiempo de comunicación de resultados, lo que generaría grandes problemas de balance carga y/o de rendimiento. Por otro lado, habría que diseñar dos tipos de procesos que realicen la FFT en una y en dos dimensiones[3].

Explotar el nivel de variable espacial se presenta como una solución más prometedora, pero hay que tener en cuenta la dependencia de datos. El tener un proceso central que va guiando la deconstrucción y cada vez que necesita un producto de convolución lo solicita a un proceso secundario puede parecer una buena idea. Pero hay que tener en cuenta que para realizar el

[3] O utilizar una biblioteca que disponga de esta característica.

cálculo de 6.5 hay que tener transformados todos los puntos del espacio para cada $(\theta_1 = \alpha, \theta_2 = \beta, \theta_3 = \gamma)$ libre, ya que para obtener las coordenadas (r'', φ_1'') de cada punto de un obstáculo, se necesitan, entre otros valores, las coordenadas (r', φ_1'). Esto obligaría a un complicado esquema de comunicaciones de los puntos transformados —incluyendo envíos de la misma información a procesos diferentes, lo que implica un aumento de la memoria necesaria para guardar esos datos mientras es necesario—, o bien a realizar cálculos redundantes —volver a transformar puntos—. En definitiva, el diseño del protocolo de comunicaciones, en particular, y el diseño de algoritmo paralelo, en general, sería muy costoso y el riesgo de bloqueos por dependencia de datos muy grande.

Finalmente, se optó por explotar la tercera posibilidad, el nivel asociado al proceso de deconstrucción, cuyo diseño y validación se comentan en las siguientes secciones.

6.4 Diseño del Algoritmo Paralelo

A continuación se va a presentar el diseño del algoritmo paralelo que calcula el espacio de las configuraciones de un robot PUMA redundante que se mueve en un espacio tridimensional.

La solución algorítmica adoptada propone calcular un mapa de bits que representa al C-espacio del robot a través de convoluciones de los mapas de bits de los obstáculos y de los elementos del robot en el espacio de trabajo. Como herramienta matemática para realizar la convolución se utiliza la transformada de Fourier y se completa la solución algorítmica con la introducción de la Transformada Rápida de Fourier (FFT). El algoritmo propuesto parte del presentado en el cuarto capítulo (tablas 4.10 y 4.11), que se analiza detalladamente a continuación.

En él, $(\theta_1, \theta_2, \theta_3, \theta_4)$ son los parámetros que definen una configuración, siendo estos los valores discretos del ángulo de rotación asociado a cada una de las cuatro articulaciones de revolución; B^* es el mapa de bits de los puntos (en esféricas) del espacio de trabajo donde se mueve el robot, respecto a F_W; $\bar{A}^*_{2_{(0,0)}}$ es el mapa de bits (en esféricas) del elemento $\mathbf{A}_2$, considerado en la configuración $(\theta_1, \theta_2) = (0, 0)$; $\bar{A}^*_{3_{(0)}}$ es el mapa de bits (en polares) del elemento $\mathbf{A}_3$, considerado en la

posición ($\theta_3 = 0$); $\bar{A}^*_{4_{(0)}}$ es el mapa de bits (en polares) del elemento $\mathbf{A}_4$, considerado en la posición ($\theta_4 = 0$); $B'^*_{(\theta_1,\theta_2)}$ es el mapa de bits de los puntos (en polares) del espacio de trabajo transformados respecto a F'_W; y $B''^*_{(\theta_1,\theta_2,\theta_3)}$ es el mapa de bits de los puntos (en polares) del espacio de trabajo transformados respecto a F''_W.

Como se puede observar hay dos partes bien diferenciadas:

- La correspondiente al cálculo de CB_2 (tabla 4.10), donde es necesario el cálculo iterativo de la porción del C-espacio correspondiente a cada uno de los radios (`rad`), siendo estas operaciones totalmente independientes entre sí. El número de iteraciones necesarias depende de la discretización elegida para cubrir la longitud del robot. Para realizar el cálculo de la porción correspondiente a cada esfera se usa la FFT bidimiensional. Por otro lado, se debe destacar que se trabajará con discretizaciones que sean potencia de dos. Esta particularidad se debe al algoritmo para la FFT, que obliga a que el tamaño de los datos a transformar cumplan este requisito.

- La correspondiente al cálculo de CB_3 y CB_4 (tabla 4.11), donde para cada configuración libre (θ_1, θ_2) se debe realizar el cálculo iterativo de las porciones de C-espacio correspondientes a los radios de dos circunferencias, una que recorre la longitud de $\mathbf{A}_3$ y otra la de $\mathbf{A}_4$. Sin embargo, estos cálculos no son independientes, ya que se necesita realizar las transformaciones de los puntos del espacio de trabajo, primero respecto a F'_W, y después, una vez obtenida ésta, respecto a F''_W. Además, para saber qué porciones de CB_4 hay que calcular, hay que saber qué configuraciones $(\theta_1, \theta_2, \theta_3)$ son libres.

Como ya se ha explicado en la sección anterior, y a la vista del algoritmo, la opción que se presenta más prometedora en cuanto al balance de carga, independencia de datos, complejidad del diseño y posibilidades de escalado, es evaluar primero CB_2[4], y realizar el cálculo paralelo de CB_3 y CB_4 para cada

[4]Este cálculo se podría realizar en paralelo explotando los niveles de paralelismo asociados al cálculo de CB_k, pero no contrinuiría demasiado en el resultado final, pues en la implmentación del algoritmo secuencial sólo supone un 0.0032 % del tiempo total.

configuración (θ_1, θ_2) libre.

Así, de entre los posibles niveles de paralelismo, sólo se explota el inherente al proceso de deconstrucción, para el que es necesario llevar a cabo un buen diseño de algoritmo paralelo que contemple la concurrencia, la escalabilidad y la localidad; El diseño de algoritmos paralelos no se puede reducir a una simple receta; todo lo contrario, requiere esa clase de pensamiento integrador que normalmente se denomina *creatividad*; sin embargo, el diseño se puede beneficiar de un análisis metódico que maximice el número de posibilidades consideradas, permita evaluar alternativas y reduzca el coste de recuperación de malas elecciones.

Además, la paralelización de este algoritmo tiene una vocación general, es decir, se pretende diseñar un algoritmo paralelo susceptible de ser trasladado a diferentes entornos paralelos, por lo que en el diseño se pretenderá incidir especialmente en la portabilidad de la aplicación.

6.4.1 Diseño PCAM del Algoritmo

La mayoría de los problemas de programación tienen múltiples soluciones paralelas, y no siempre la solución óptima es aquella sugerida por el algoritmo secuencial. Para llevar a cabo un estudio del diseño adecuado para el problema que nos ocupa, se ha seguido un método conocido como PCAM [Fos95], el cual recibe este nombre por ser un acrónimo de las cuatro etapas que incluye: particionamiento, comunicación, aglomeración y asignación (*mapping*).

El método, en las dos primeras etapas, tiene como objetivo la concurrencia y la escalabilidad, mientras que en las dos últimas se centra en la localidad y otros conceptos relativos al rendimiento.

Particionamiento

En esta fase se tienen en cuenta el cálculo que se va a realizar y los datos sobre los que se tiene que operar: estos se descomponen en pequeñas tareas. Los problemas de tipo práctico como el número de procesadores en el computador objetivo se ignoran en esta primera fase. Simplemente se trata de reconocer las oportunidades de ejecución paralela presentes.

Por tanto, el objetivo es definir un gran número de pequeñas tareas para alcanzar una descomposición del problema de grano fino —división del trabajo en un número elevado de tareas de pequeño coste—. Esto es así porque el grano fino proporciona una mayor flexibilidad en los algoritmos paralelos.

Como ya se ha comentado, en el algoritmo secuencial objeto de la paralelización existe un gran número de operaciones independientes, determinadas por la resolución elegida para el cálculo, que constituyen el tamaño menor de tarea a realizar. Por tanto, en este caso, el grano fino estará constituido por el cálculo de la porción de C-espacio asociada a CB_3 y CB_4 para una determinada configuración libre (θ_1, θ_2).

Sin embargo, en etapas avanzadas del diseño óptimo para la implementación, tras el análisis de los requisitos de comunicaciones, la arquitectura objetivo, o aspectos de Ingeniería del Software, hay que revisar el particionamiento y la aglomeración iniciales e incrementar el tamaño de las tareas, o lo que es lo mismo, la granularidad. Con esta forma de particionamiento primero se dividen los cálculos asociados al problema y después se controla cómo tener disponibles los datos sobre los que se realizan estos cálculos. Así, en el grano fino, para calcular el C-espacio debido al tercer y cuarto elementos del robot para una configuración libre (θ_1, θ_2) se debe indicar qué configuración es ésta. Si el grano es grueso, se le pedirá a los procesos que realicen los cálculos para varias configuraciones libres. Esta técnica de particionamiento se denomina *descomposición funcional*.

Para indicar al proceso las configuraciones que tiene que calcular, se recurre a una codificación mediante un número entero $Tarea$, tal que

$$\theta_1 = Tarea/N$$

y

$$\theta_2 = Tarea \ mod \ N$$

para una resolución de discretización N. Si el grano fuera grueso, se envía un vector de tareas codificadas de la misma manera.

El particionamiento elegido, grano fino, es óptimo por presentar las siguientes características deseables:

- Normalmente, se tienen más tareas que procesadores en el computador objetivo, lo que aporta flexibilidad en las

subsecuentes etapas. Efectivamente, si se considera por ejemplo una discretización de $256 \times 256 \times 256 \times 256$, se puede llegar a obtener 256×256 configuraciones (θ_1, θ_2) libres, lo que significa 65536 tareas, número que con toda seguridad superará la cantidad de procesadores disponibles.

- Se evitan los cálculos y almacenamientos redundantes. Esto facilita el escalado futuro del algoritmo. Sólo se va a almacenar una copia del resultado final y de los resultados intermedios en cada elemento de procesamiento. Similarmente, sólo se van a realizar los cálculos en una única ocasión: un elemento de procesamiento realizará el cálculo para una configuración (θ_1, θ_2) libre.

- El número de tareas crece con el tamaño del problema. Así, el algoritmo podrá resolver problemas mayores cuando esté disponible un mayor número de procesadores. Como se deduce de todo lo expuesto hasta el momento, el problema crece al aumentar la resolución con que se discretiza, con lo que cuando se dispone de mayor número de procesadores se puede aumentar la resolución de trabajo, obteniendo así resultados más precisos.

- Se han previsto particiones alternativas, lo que flexibiliza el trabajo en etapas siguientes. Es posible dividir el problema de forma distinta, eligiendo como grano fino un número mínimo de orientaciones a calcular, o un conjunto de orientaciones compuestas por un orden específico de las mismas.

Comunicación

Se pretende que las tareas que se generan en una partición se ejecuten concurrentemente, pero generalmente no es posible que se ejecuten de un modo absolutamente independiente. Normalmente, los cálculos que realiza una tarea necesitan de los datos asociados a otra tarea.

Como se ha utilizado en la etapa anterior una descomposición funcional, los requisitos de comunicaciones para el algoritmo paralelo son directamente identificables. En caso de haber utilizado una *descomposición de dominio* —donde se dirige la

atención sobre cómo particionar los datos del problema para luego decidir cómo asociar los cálculos a los datos—, la tarea habría sido más complicada.

En el algoritmo se van a encontrar cuatro contraposiciones en cuanto a las comunicaciones que se deben estudiar delicadamente: local/global, estructurada/no estructurada, estática/dinámica y síncrona/asíncrona.

- Se necesitarán comunicaciones locales (una tarea se comunica con pocas tareas, sus *vecinas*) y globales (una tarea se comunica con el resto de las tareas). Dado que inicialmente se realiza el cálculo de la FFT de los elementos del robot, se necesitan, en todas las tareas definidas, las matrices con estos bitmaps ($\bar{A}^*_{3_{(0)}}$ y $\bar{A}^*_{4_{(0)}}$). Será necesario que éstas se envíen desde un elemento de procesamiento a todos los demás, mientras que los resultados parciales de cada tarea se deben enviar al elemento de procesamiento encargado de recomponer los resultados intermedios en un resultado final.

- En una comunicación estructurada, una tarea y sus vecinas forman una estructura regular, como un árbol o una malla; en una comunicación no estructurada, se tienen grafos arbitrarios. Para el problema que nos ocupa se tiene una estructura de comunicaciones bien definida, ya que las tareas se comunican en forma de estrella: en el centro una tarea envía un trabajo y recibe resultados de las que se encuentran en las puntas de la estrella (estructura similar a la de la figura 6.3).

- En las comunicaciones estáticas, la identificación de las parejas de comunicación no cambian en el tiempo, frente a la comunicación dinámica, donde estas variables cambian en tiempo de ejecución. En el caso que nos ocupa, todas las comunicaciones están definidas previamente, no va a cambiar el esquema de comunicaciones en tiempo de ejecución por variación de las entradas al problema, por ejemplo.

- En las comunicaciones síncronas, los productores y consumidores de información se ejecutan de forma coordinada. Dadas las particulares necesidades del cálculo, las

comunicaciones ocurren de forma coordinada: se envían los bitmaps de los elementos tercero y cuarto del robot, que todas las tareas están esperando, a continuación se va indicando por turno la configuración (θ_1, θ_2) que cada tarea debe realizar y cuando esta operación está realizada, se envía el resultado intermedio que se está esperando. De esta forma se suceden las comunicaciones hasta recibir la última porción del resultado final.

Con las elecciones tomadas en esta fase de diseño se consigue que:

- Todas las tareas realizan, aproximadamente, el mismo número de operaciones de comunicación. Esto asegura unas comunicaciones balanceadas. De esta forma, cada configuración a calcular estará caracterizada por un envío de información indicando a la tarea de qué configuración se trata seguido de una comunicación para enviar el resultado intermedio.

- La mayoría de las comunicaciones son locales lo que facilita la definición de las tareas de envío y recepción en la tarea productora y consumidora respectivamente. Queda perfectamente definido en qué momentos se van a realizar las comunicaciones entre tareas lo que permite la coordinación deseada: se espera a recibir qué configuración realizar, y una vez calculada se envía.

- No todas las comunicaciones se pueden hacer de forma concurrente, por lo que habrá que asegurarse de que el cálculo avance adecuadamente cuando una tarea tiene que esperar a poder completar una comunicación. Puesto que hay una tarea central que va a ir recibiendo resultados y comunicando nuevas configuracione, existe la posibilidad de que se termine el cálculo de una configuración, y ésta tenga que esperar a que la central termine otra comunicación y le pueda atender. Esto se resuelve si el tiempo de comunicación es mínimo en relación con el tiempo que se tarda en calcular.

Aglomeración

En esta etapa de diseño se pasa de un caso abstracto a uno concreto. Se revisan las etapas anteriores con la idea de obtener un algoritmo que se ejecute eficientemente en distintas clases de computadores paralelos.

Para ello se decide si es conveniente combinar o aglomerar, tareas identificadas en la fase de particionamiento, para proporcionar menor número de tareas, pero cada una de las cuales con un tamaño mayor. Se puede decidir en esta fase si se reduce el número de tareas a exactamente el número de procesadores o si se posterga a la fase de asignación. También se puede realizar la aglomeración replicando datos o cálculos en caso de necesidad. Se debe realizar un adecuado estudio de los costes de las comunicaciones y para ello se experimentará con la granularidad y aglomeración de las tareas. La forma de aglomerar es creando un grupo de n configuraciones, pero obligando a que el número total de tareas a calcular sea un múltiplo de n.

Las características alcanzadas en esta fase de diseño son:

- Cuando se aglomera se reducen los costes de comunicación. El tiempo de comunicación no consiste únicamente en el tiempo dedicado al paso de una determinada cantidad de datos, sino que también lleva asociado un coste por establecimiento de la comunicación.

- No se replican datos ni cálculos, con lo que se asegura una escalabilidad del algoritmo. Como ya se ha comentado, los cálculos se realizan una única vez y se distribuyen los resultados cuando es necesario. Los datos sólo se almacenan en una única localización.

- Cuando se aglomera, las nuevas tareas tienen exactamente la misma carga. Efectivamente, la aglomeración de tareas propuesta va a venir formada únicamente por el cálculo en una tarea de más de una configuración, pero se aglomera de un modo uniforme, es decir, el algoritmo puede tener que calcular 1328 configuraciones, y se puede definir que cada tarea tenga un tamaño de 8 configuraciones, por lo que habrá una totalidad de 166 tareas a realizar.

- De nuevo, el número de tareas crece con el tamaño del problema. Dada la forma de aglomerar, al aumentar el tamaño del problema, es decir, al crecer N, si se mantiene n constante crecerá el número de tareas. En el caso que se comentaba en el punto anterior, si se pasara de una resolución 64 a una resolución 256, en vez de 1328 configuraciones libres, se tendrían muchas más y si se mantuviera n igual a 8, se tendrían que realizar más de 166 tareas.

- La aglomeración no elimina oportunidades de ejecución concurrente. Aún en casos sencillos, pasar, por ejemplo, de 1328 tareas a 166, no elimina oportunidades de concurrencia; todavía es posible realizar 166 tareas en paralelo, en caso de contar con otros tantos procesadores.

- La aglomeración puede llegar hasta el punto de definir una única tarea para cada procesador sin introducir desajustes graves en el balance de la carga. Como se puede comprobar y siguiendo con el ejemplo anterior, si se tuviera únicamente cuatro procesadores, se podría definir un tamaño de grano de 332, con lo que tendríamos 4 tareas que realizaría cada uno de los procesadores disponibles. Si se tratara de una resolución mayor, únicamente habría que definir un tamaño de grano mayor.

- La aglomeración es óptima desde el punto de vista de Ingeniería del Software, ya que permite una amplia reutilización del código desarrollado para la implementación del algoritmo secuencial. Dado que la aglomeración viene únicamente de la agrupación del mismo tipo de tareas, lo que hará falta es escribir un código que utilice las mismas funciones, pero en vez de para una configuración, lo haga para el número de configuraciones que definen la tarea.

Asignación

En la etapa final del diseño, el modelo abstracto alcanzado se traduce en uno real, es decir, se decide dónde se va ejecutar cada tarea. Dado que se ha realizado una descomposición funcional lo óptimo para realizar un balance de carga es la utilización de algoritmos de planificación de tareas de computación,

que asignan las tareas a procesadores que están desocupados o que se prevé que vayan a estar desocupados.

El aspecto más complicado de estos algoritmos es cómo asignar problemas a los trabajadores. La estrategia elegida representará un compromiso entre los conflictos requeridos para operaciones independientes —para reducir los costes de las comunicaciones— y el conocimiento global del estado de los cálculos —para mejorar el balance de la carga.

Como estrategia se ha elegido el enfoque maestro-esclavo, de forma que va a ser el proceso maestro el que se va a encargar de ir repartiendo tareas a cada uno de los procesos esclavos y, paulatinamente, ir recibiendo los cálculos de cada esclavo, para construir el resultado final (las figura 6.4 ilustra el enfoque maestro-esclavo en una red de Petri coloreada. Sin embargo, el lugar donde se realiza el cálculo cambiará de cometido para este caso de estudio y el proceso central, en lugar de pedir cálculos para una variable espacial, lo hará para una configuración determinada).

Dado el análisis previo que se ha hecho, todo el diseño se ve avocado a este enfoque. Se definían las tareas como los cálculos necesarios para evaluar el espacio de las configuraciones de los elementos tercero y cuarto de un robot PUMA redundante, para configuraciones (θ_1, θ_2) libres, según una determinada discretización. Se elegía una estructura de comunicaciones en estrella, con una tarea central indicando cuáles eran las próximas configuraciones a calcular y recibiendo los resultados intermedios para construir el resultado final. Estas operaciones que se acaban de describir son las que realizará el maestro. Se tendrán tantos esclavos como procesadores disponibles, de forma que en el caso ideal, se pueden definir tantos esclavos como tareas hay que realizar.

Si esto último no es posible en la realidad, simplemente se considera que los esclavos sigan estando activos mientras el maestro les dé trabajo. Esta idea consiste en lo siguiente: se definen tantos esclavos como procesadores —o como el número de procesadores menos uno si se quiere dedicar un procesador al maestro— el maestro va repartiendo tareas, los esclavos van calculando, y según van terminando comunican los resultados intermedios y piden más tareas que calcular. Este proceso se repite hasta que no quedan más tareas por realizar y es el maestro el que indica a los esclavos que finalicen su ejecución.

6.5 El Diseño en una Red de Petri Coloreada

En la figura 6.7 se presenta mediante una red de Petri coloreada el algoritmo paralelo diseñado según se ha comentado en los apartados previos. Así, se sigue el paradigma maestro-esclavo, donde las operaciones independientes se corresponden con cálculos, para configuraciones libres de colisiones del segundo eslabón de la cadena cinemática, de la porción de C-espacio debida a los dos elementos restantes del robot. De esta forma, se entiende, en este caso, por tarea los cálculos necesarios para la obtención de una porción formada por una o varias configuraciones, siendo el tamaño del grano el número de configuraciones (θ_1, θ_2) que constituyen una tarea. Esto conlleva un balance adecuado de la carga si se utilizan tantos esclavos como el número de procesadores disponibles.

La CPN de la figura 6.7 modelan la operaciones esenciales que deben realizar el maestro y el esclavo. No se tienen en cuenta detalles como operaciones anteriores o posteriores al cálculo paralelo, ni cálculos intermedios o de organización; esto es así porque la red se va a usar para validar el diseño, es decir, el correcto funcionamiento concurrente del algoritmo, para justificar su posterior implementación. No se pretende una simulación realista de todos los aspectos involucrados en la evaluación del C-espacio.

El maestro se encarga de calcular los datos preliminares que necesitan los esclavos, dirige su ejecución, marca la finalización de todos los procesos, y construye el resultado final. Por otro lado, la tarea de cada esclavo consiste en calcular la porción del C-espacio correspondiente a la configuración o configuraciones que el maestro le haya asignado y proporcionar el resultado al maestro. La red sirve tanto para grano fino como para grano grueso, ya que sólo se modela a nivel de tarea.

En el centro, se han usado dos lugares para modelar las comunicaciones entre el proceso maestro y el esclavo.

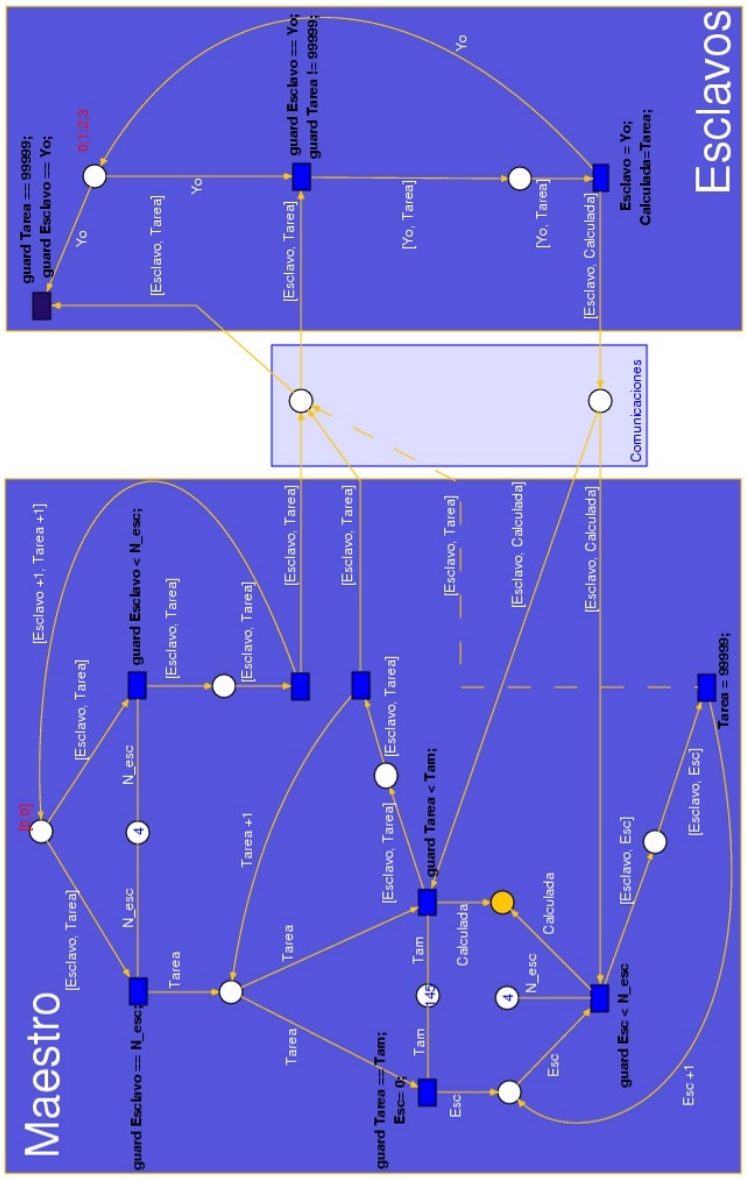


Figura 6.7: Red de Petri para la deconstrucción en paralelo para un robot PUMA redundante de cuatro DOFs

En el ejemplo de la figura, se tiene un banco de 4 procesos esclavos (`0;1;2;3`, en el lado de los esclavos y `4`, en los lugares con un arco de test[5], `N_esc`, en el maestro). A continuación se desglosan las operaciones que realizan los dos tipos de procesos.

6.5.1 El Proceso Maestro

Como se ha dicho, el maestro es el que se encarga de dirigir todo el cálculo. En el ejemplo, se van a calcular 145 tareas (el valor que leen los arcos de test `Tam`). Arriba del todo, un lugar tiene un marcado inicial `[0,0]`, lo que indica que se va a empezar el cálculo solicitando que el esclavo `0` realice la tarea `0`. Justo debajo, otro lugar tiene la marca inicial `4`, indicando que éste es el número de procesos esclavos que se está simulando. Así, del lugar de arriba salen dos arcos que llevan `[Esclavo, Tarea]` a dos transiciones con dos guardas diferentes: `guard Esclavo < N_esc`, a la derecha, y `guard Esclavo == N_esc`, a la izquierda. A ambas transiciones, mediante sendos arcos de test, les llega el valor de `N_esc`, `4`. Como el marcado inicial era `[0,0]` y `4`, esto es lo que va a llegar a ambas transiciones, pero sólo se disparará la de la derecha, pues `0` es menor que `4`. La transición quita `[0,0]` del lugar de arriba y lo coloca en el de salida. A la siguiente transición llega `[0,0]`, que se coloca en el lugar que simula las comunicaciones en la sentido maestro-esclavo. Además, dado que hay un arco con la inscripción `[Esclavo+1, Tarea+1]`, colocará en el lugar de arriba del todo `[1,1]`. Lo que significa que la tarea `1` se asignará al esclavo `1`. Este ciclo se repite hasta que se hayan repartido tareas para todos los esclavos disponibles; 4 en el ejemplo.

Cuando ya se hayan repartido tareas para todos los esclavos, se cumplirá la guarda de la transición de la izquierda —a la vez que dejará de cumplirse la de la derecha—. En el ejemplo, esto ocurre cuando en el lugar de arriba se tiene la marca `[4,4]`. Entonces se dispara la transición de la izquierda, que lleva `[4,4]` a las transiciones con guardas `guard Tarea == Tam`, a la izquierda, y `guard Tarea < Tam`, a la derecha. Además, a ambas, a través de dos arcos de test,

[5] Los arcos de test son aquellos que leen el valor de una marca en un lugar sin quitarla de ese lugar. Se simbolizan con una línea sin terminaciones en punta

les llega el valor de `Tam`, `145` en este caso. Por otro lado, a la transición de la derecha le llega un arco con la inscripción `[Esclavo, Calculada]`. Éste sale desde el lugar que simula las comunicaciones en la sentido esclavo-maestro, y lo que simula es una porción de C-espacio calculado y una indicación de qué esclavo es el que la ha calculado y, por tanto, ha quedado libre. De manera análoga al bucle anteriormente explicado, ahora se va a repetir un bucle 141 veces —el número de tareas que queda por asignar, `Tam` − `N_esc` = 145 − 4—, en las que se dispara la transición de la derecha, lo que hace que, por un lado, se envíe un número de tarea asociado a un esclavo que se sabe que está libre (`[Esclavo, Tarea]`) y se incremente el contador de tareas (`Tarea+1`); y por otro lado, se coloque el resultado parcial en el resultado final (`Colocada`).

Después de recibir resultados parciales y enviar nuevas peticiones de cálculo, llegará un momento en que sólo se cumpla la guarda de la izquierda; en el ejemplo, cuando se llegue al valor `145`. En este momento queda unicamente recibir `N_esc` (`4` en nuestro caso) tareas, e indicar a los esclavos que ya pueden terminar su ejecución. En la parte inferior, y de modo similar a los dos bucles anteriores, se modela este lazo, en el que se recibe `[Esclavo, Calculada]`, se guarda `Calculada` en el lugar que le corresponda y se le indica a ese esclavo que puede terminar (`[Esclavo, Tarea]` en línea discontinua). Obsérvese que en la transición de más abajo se ha hecho la asignación `Tarea = 99999`, ya que este número de tarea se utiliza para indicar fin de ejecución.

6.5.2 El Proceso Esclavo

Como se ha comentado previamente el proceso esclavo se encarga de calcular la porción del C-espacio correspondiente a un conjunto de configuraciones determinado por el maestro. Esta operación, que constituye una tarea, se puede desglosar en las acciones que se detallan a continuación.

En la parte que modela (figura 6.7) los esclavos en la red se tiene arriba un lugar que tiene el marcado inicial `0;1;2;3`, lo que indica que hay cuatro esclavos libres al principio. La transición de abajo se disparará siempre que reciba un marca `[Esclavo, Tarea]`, desde el lugar que modela las comunicaciones en el sentido maestro-esclavo; reciba una marca `Yo`, des-

de el banco de esclavos libres; y se cumplan simultáneamente las dos guardas que tiene asociadas: `guard Esclavo == Yo` y `guard Tarea != 99999`. Es decir, que un esclavo pasará a calcular la tarea que le asigne el maestro, quitando su marca del conjunto de marcado inicial. Así, el primero que se pondrá a trabajar es el esclavo `0`, con la tarea `0`, y el banco de esclavos libres quedará `1;2;3`. Cada esclavo tardará un tiempo en realizar la tarea, de forma que una vez que los esclavos estén trabajando no se puede determinar cuál de ellos quedará libre antes. Cuando un esclavo ha calculado su tarea, envía los resultados (`[Esclavo, Calculada]`) al lugar que modela las comunicaciones en el sentido esclavo-maestro. Obsérvese que en la transición de la parte inferior se hacen las asignaciones `Esclavo = Yo`, para indicarle al maestro qué esclavo ha quedado libre, y `Calculada = Tarea`, donde se asocian los resultados correspondientes a la tarea. Además esta transición coloca el identificador del esclavo `Yo` de vuelta al lugar de esclavos libres.

Finalmente, en al parte superior se tiene una transición con las guardas `guard Tarea == 99999` y `Esclavo == Yo`. Cuando esto se cumple, se quita el esclavo del banco de esclavos libres. Es decir, aquí se modela el momento en que un esclavo termina su ejecución por orden del maestro.

Con esta red de Petri coloreada se puede comprobar el correcto funcionamiento concurrente para cualquier número de procesos esclavos —menor que el número de tareas, si no se quiere infrautilizar recursos—, del algoritmo paralelo diseñado para evaluar el espacio de las configuraciones de un robot PUMA redundante. Pero hasta este punto no se puede decir nada de cómo se podría comportar una implementación de este algoritmo, pues no se han tenido en cuenta los tiempos asociados al cálculo y comunicaciones.

En la siguiente sección se realiza este estudio.

6.6 Simulación de Rendimiento del Algoritmo Paralelo

En esta sección tiene como objetivo presentar la validación del algoritmo paralelo diseñado, verificando su portabilidad, es de-

cir, que podría implementarse eficientemente en una variedad de máquinas paralelas.

6.6.1 Modelos de Medida del Rendimiento Paralelo

El objetivo de los modelos es ofrecer una representación de los algoritmos paralelos independiente de las características específicas de una arquitectura determinada. Este objetivo se materializa normalmente expresando el rendimiento del algoritmo paralelo en función de un número básico de parámetros, cuyo valor se pueden obtener empíricamente (mediante test ping-pong[6] habitualmente).

Uno de los modelos más utilizados históricamente es el PRAM[FW78], pero se trata de un modelo no realista porque asume que los procesadores trabajan síncronamente y sin penalizaciones por la comunicación interprocesadores. A partir de este modelo se han desarrollado otros que intentan superar esta limitación.

Uno de los modelos más ampliamente utilizados para el procesamiento paralelo en sistemas de paso de mensajes es LogP[CKP+93]. Este modelo utiliza cuatro parámetros para proporcionar expresiones del rendimiento de los algoritmos: L, *latencia*, o retardo en comunicar un mensaje con una palabra desde el origen al destino; g, (*gap*), el mínimo intervalo de tiempo entre dos transmisiones o recepciones de un mensaje consecutivas, para un procesador —reciprocamente se tiene el *ancho de banda de comunicación por procesador*—; o (*overhead*), *carga*, tiempo que el procesador está ocupado en la transmisión o recepción de un mensaje; y P, el número de módulos procesador/memoria. Estos parámetros no son igualmente importantes en todas las situaciones y se pueden ignorar uno o más parámetros según la aplicación.

Finalmente, desde la aparición del modelo LogP han surgido otros que siguen la misma filosofía, entre ellos se encuentra Dimemas [GLB00], un modelo que permite predecir el rendi-

[6] Normalmente se trata de tener dos procesadores con dos procesos, el primero está en un ciclo enviando y recibiendo, y el otro en un ciclo recibiendo y enviando.

miento de los programas de paso de mensajes, y que es el que se ha utilizado en este trabajo.

6.6.2 Métricas de Rendimiento Paralelo

Las métricas son las herramientas de las que se dispone para evaluar el rendimiento de los algoritmos paralelos. Se pueden dividir en dos grupos:

- Métricas que miden el rendimiento punto a punto;
- Métricas colectivas, que ven el sistema paralelo como un todo.

Las métricas colectivas se basan en el tiempo de ejecución paralela y reflejan la contribución de cada procesador. Estas métricas se centran en el rendimiento y la escalabilidad y entre ellas destacan:

- Tiempo de ejecución paralela: T_p
- *Speedup*: $\frac{T_l}{T_p}$
- Eficiencia: $\frac{T_l}{pT_p}$
- Rendimiento: $\frac{MFLOPS}{sec}$
- Coste-Eficiencia: $\frac{\frac{MFLOPS}{sec}}{coste}$, que considera el *coste* económico.

con T_l, el tiempo de ejecución del mejor algoritmo secuencial, y p, el número de procesadores.

Las dos métricas punto a punto más extendidas son la latencia y el ancho de banda. Estas métricas reflejan completamente el subsistema de comunicaciones, incluyendo las capa física y la de enlace de datos de la red de interconexión, el sistema operativo, los controladores de red y el software intermedio de paso de mensajes.

6.6.3 Estimación del Tiempo de Cálculo y Comunicaciones

Para llevar a cabo el estudio del rendimiento del algoritmo paralelo, se modelarán tres tipos de máquinas paralelas: un multiprocesador de memoria compartida, una red de estaciones

de trabajo conectadas con Fast Ethernet, y una red de estaciones de trabajo conectadas con una red Gigabit Ethernet. No obstante, este estudio no pretende ser realista, simplemente se usa como una forma de validar el diseño paralelo; si los resultados teóricos de estas simulaciones son prometedores, se justificará la implementación en alguno de los entornos citados.

Se considerarán tres resoluciones de discretización: 64; 128 y 256.

Para simular los tiempos asociados a las comunicaciones se utiliza la simulación temporal que proporciona la herramienta Renew. Basta con asociar a cada arco de salida de una transición un valor: se añade un retardo a la inscripción del arco mediante el símbolo @ y una expresión que evalúa el número de unidades de tiempo.

Se estiman teóricamente los tiempos de comunicación de los mensajes de los distintos tamaños y para las distintas arquitecturas del estudio. Para ello se modela el tiempo de comunicaciones punto a punto para MPI seguiendo la fórmula simplificada, como aparece en [GLB00], un modelo de comunicaciones para MPI heredero de LogP, de la siguiente forma:

$$T_{com} = L + \frac{Tam}{AB} \tag{6.6}$$

donde L es la latencia, Tam es el tamaño del mensaje, y AB el ancho de banda.

Esta fórmula sólo se puede aplicar a una red sin contención, con un número ilimitado de recursos. Por otro lado, aunque los modelos de computación paralela utilizan la latencia y el ancho de banda para proporcionar expresiones del rendimiento total de la aplicación, no se ha comprobado que los test ping-pong reflejen realmente los parámetros latencia y ancho de banda que requieren estos modelos[DS00].

Sin embargo, estas estimaciones son una buena forma de acercarse al problema, por lo que están ampliamente extendidos y son suficientes para esta fase previa a la implementación. Para realizar el estudio se toman los valores promediados de ancho de banda y latencia, para MPI, proporcionados en un informe técnico, del *High Performance Computing Segment* de Compaq [Seg00], sobre las interconexiones Fast Ethernet y Gigabit Ethernet, y en [GZ97] para el multiprocesador Origin 200. La tabla 6.1 muestra estos valores.

	Ancho de Banda (MB/s)	Latencia(μs)
Fast Ethernet	7	40
Gigabit Ethernet	75	32
Origin 200	128	16

Tabla 6.1: Latencia y ancho de banda para diferentes arquitecturas paralelas

Queda ahora asociar un tiempo de cálculo a cada tarea, para ello, se tiene en cuenta que éste variará con el número de configuraciones de θ_3, para la(s) configuración(es) (θ_1, θ_2), que produzcan colisión, y con la resolución de discretización. La forma de estimar el tiempo de cálculo es la siguiente:

$$T_{cal} = (T_{\theta_3}(N) + T_b(N) \cdot C_{\theta_{3_l}}) \cdot Grano$$

donde T_{θ_3} es el tiempo que se tarda en evaluar el espacio de las configuraciones para el tercer DOF, para una configuración (θ_2, θ_3) dada; T_b es el tiempo básico de cálculo, dependiente de la resolución elegida, del C-espacio debido al cuarto elemento del robot para una configuración $(\theta_1, \theta_2, \theta_3)$ dada; $C_{\theta_{3_l}}$ es el número de configuraciones de θ_3 libres de colisión, para una configuración (θ_1, θ_2) dada; y $Grano$ es el número de configuraciones (θ_1, θ_2) que conforman una tarea.

Empíricamente, mediante la ejecución del algoritmo secuencial con los diferentes tamaños de resolución elegidos se tienen los siguientes tiempos básicos de cálculo del C-espacio para el tercer (tabla 6.2) y el cuarto elemento (tabla 6.3).

N	$T_{\theta_3}(\mu s)$
64	1978
128	7711
256	30011

Tabla 6.2: Tiempos de cálculo básicos para la tercera articulación con diferentes resoluciones

Como se puede observar, los tiempos son mayores para la tercera articulación porque la transformación de los obstácu-

los es más costosa ($\approx 130\mu s$ por punto transformado) que la asociada al cuarto elemento ($\approx 8\mu s$ por punto transformado).

N	$T_b(\mu s)$
64	1610
128	7694
256	29224

Tabla 6.3: Tiempos de cálculo básicos para la cuarta articulación con diferentes resoluciones

Si ahora se calcula el tiempo estimado para todos los valores de $C_{\theta_{3l}}$, desde 1 hasta N, y se calcula la media y la desviación típica para cada una de las tres resoluciones, se obtienen las siguientes distribuciones normales[7] para el cálculo con grano fino (en segundos): N(0.05, 0.03) para $N = 64$, N(0.11, 0.06) para $N = 128$ y N(0.21, 0.12) para $N = 256$. En la figura 6.8 se observan las representaciones gráficas de las funciones de densidad que corresponde a tales distribuciones.

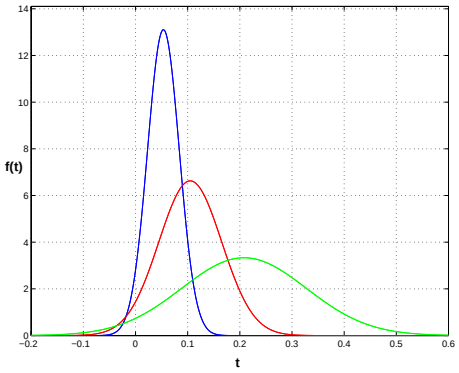


Figura 6.8: Funciones de densidad correspondientes a las distribuciones del tiempo de cálculo. Azul:64, Rojo: 128 y Verde:256.

Por otra lado, los tamaños de los mensajes varían también con la resolución. Hay dos tipos de mensaje: los del maestro a

[7] Mediante llamadas a Java, Renew permitirá utilizar estas distribuciones en la simulación

los esclavos, que son un número entero o $Grano$ números enteros, en que se indica a cada esclavo la(s) configuración(es) que tiene que calcular; y los de los esclavos al maestro, es decir, los resultados, que serán $N \cdot N \cdot Grano$ enteros. Suponiendo un tamaño de dos bytes por entero, se tendrán entonces los tiempos de comunicación (expresión 7.20), para las tres arquitecturas del estudio, que se muestran en la tablas 6.4, 6.5 y 6.6.

N	Tarea (2 bytes)	Resultados ($2N^2 bytes$))
64	41 μs	1561 μs
128	41 μs	4504 μs
256	41 μs	17897 μs

Tabla 6.4: Tiempos de comunicación básicos en una red Fast Ethernet para diferentes resoluciones ($T_{com} = 0{,}000040 + \frac{Tam}{7}$) con $Grano = 1$

N	Tarea (2 bytes)	Resultados ($2N^2 bytes$))
64	32 μs	136 μs
128	32 μs	449 μs
256	32 μs	1699 μs

Tabla 6.5: Tiempos de comunicación básicos en una red Gibabit Ethernet para diferentes resoluciones ($T_{com} = 0{,}000032 + \frac{Tam}{75}$) con $Grano = 1$

N	Tarea (2 bytes)	Resultados ($2N^2 bytes$))
64	16 μs	77 μs
128	16 μs	260 μs
256	16 μs	993 μs

Tabla 6.6: Tiempos de comunicación básicos en una Origin 200 para diferentes resoluciones ($T_{com} = 0{,}000016 + \frac{Tam}{128}$) con $Grano = 1$

6.6.4 Simulación y Validación

Una vez hechas las estimaciones previas, se construyen con Renew dos versiones temporales de la red de Petri coloreada de la figura 6.7: una igual, pero con inscripciones en los arcos para simular los tiempos de cálculo y comunicaciones según las expresiones y distribuciones de la sección anterior; y otra que represente el cálculo secuencial.

En primer lugar, se ha de tener en cuenta que con estas redes de Petri coloreadas no se pueden reproducir dos casos exactamente iguales, ya que los obstáculos para el tercer elemento del robot se han modelado mediante las distribuciones normales en los tiempos de cálculo.

Por tanto, no se pueden comparar dos casos exactamente iguales para el algoritmo paralelo y el secuencial, por lo que se realizan suficientes experimentos y se promedian los resultados.

Por otro lado, los tiempos de cálculo totales, no són realistas pues no se han tenido en cuenta otros elementos que intervendrán en la implementación real de los algoritmos, como son las lecturas y escrituras en disco, los tiempos de organización de resultados, etc.

p	N	Tareas	T_l (s)	T_p(s)	*Speedup*	Eficiencia
4	64	145	15.17	4.13	3.67	0.92
4	64	4092	408.26	103.99	3.92	0.98
11	64	145	15.17	1.49	10.18	0.93
20	64	563	62.74	3.17	19.79	0.99
11	64	563	62.74	6.01	10.44	0.95
11	128	145	22.65	2.23	10.17	0.92
40	128	6021	944.58	25.65	36.83	0.92
8	256	145	45.95	6.42	7.16	0.90
40	256	212	60.24	1.64	36.73	0.92

Tabla 6.7: Resultados de simulación en una red Fast Ethernet

De esta manera, como muestran las tablas 6.7, 6.8 y 6.9 se obtienen unos resultados de simulación cercanos al límite teórico, con eficiencias de aproximadamente 1. Se observan ligeras mejorías a medida que disminuye la latencia y aumenta el ancho de banda del entorno simulado. Incluso, se ha recogido

algún caso no realista en el que se obtiene *speedup* superlineal (11.39 con once procesadores para el caso de 563 tareas en una Origin 200), pero hay que considerar que estos son valores teóricos óptimos, por lo que se espera que los resultados de la implementación no sean tan buenos, ya que se producirán fenómenos que la red de Petri no modela, como es el caso de que las máquinas no sean dedicadas y los procesos tengan que compartir procesadores con trabajos de otros usuarios.

p	N	Tareas	T_l (s)	T_p(s)	*Speedup*	Eficiencia
4	64	145	15.17	3.97	3.82	0.96
4	64	4092	408.26	102.89	3.97	0.99
11	64	145	15.17	1.41	10.76	0.98
20	64	563	62.74	3.16	19.85	0.99
11	64	563	62.74	5.97	10.51	0.96
11	128	145	22.65	2.18	10.39	0.95
40	128	6021	944.58	24.58	38.42	0.96
8	256	145	45.95	5.85	7.86	0.98
40	256	212	60.24	1.58	38.13	0.95

Tabla 6.8: Resultados de simulación en una red Gigabit Ethernet

Por tanto, se concluye que el diseño del algoritmo paralelo es el apropiado, ya que es válido para cualquiera de los tres entornos considerados, en los que la relación entre el tiempo de comunicación y el tiempo de cálculo de cada tarea es tal que siempre se acerca al máximo el rendimiento paralelo. Además, se observa que teóricamente el escalado del algoritmo esta garantizado.

Finalmente, se añade que habría que hacer un estudio, similar al presentado anteriormente, para cada tamaño de grano. Pero dado que, al aumentar el tamaño del grano, la relación de tiempos de cálculo frente a tiempos de comunicaciones es similar a la estudiada para el grano fino, y dado el hecho de no poder reproducir exactamente el mismo caso de simulación para la versión paralela y para la secuencial del algoritmo, los resultados serían de la misma índole que los presentados para el grano fino.

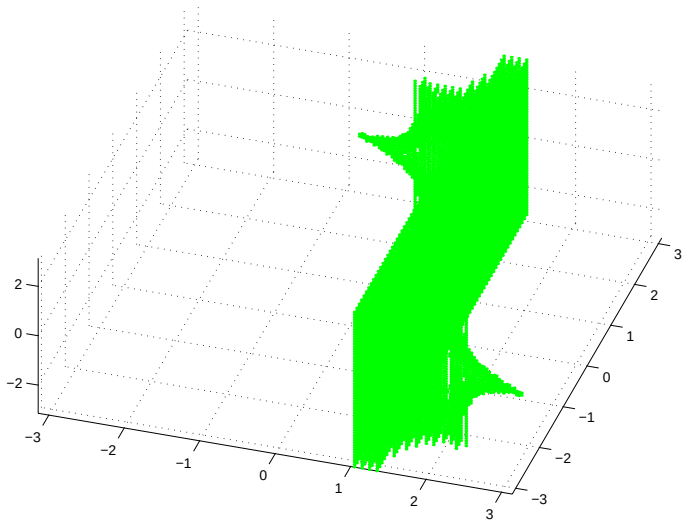
p	N	Tareas	T_l (s)	T_p(s)	*Speedup*	Eficiencia
4	64	145	15.17	3.92	3.87	0.97
4	64	4092	408.26	102.09	3.99	0.99
11	64	145	15.17	1.40	10.84	0.99
20	64	563	62.74	3.16	19.85	0.99
11	64	563	62.74	5.51	11.39	1.04
11	128	145	22.65	2.08	10.89	0.99
40	128	6021	944.58	23.90	39.52	0.99
8	256	145	45.95	5.84	7.87	0.98
40	256	212	60.24	1.52	39.63	0.99

Tabla 6.9: Resultados de simulación en una máquina multiprocesadora Origin 200

Por otro lado, como ya se ha comentado, el objetivo principal de la simulación era comprobar el escalado del algoritmo y estimar el rendimiento que se va a obtener. Si se quisiera analizar más a fondo el rendimiento del algoritmo, entonces habría que modificar el diseño de la red de Petri coloreada y estudiar la medida en que influyen cálculos intermedios, escrituras en disco, etc., lo cual cae fuera del objeto de este trabajo de investigación.

Capítulo 7

Resultados



Calvin: Yo creo que si algo es tan complicado que no puedes explicarlo en 10 segundos, seguramente no vale la pena conocerlo.

El Gran Calvin y Hobbes
Ilustrado
BILL WATTERSON

Se van a considerar dos vertientes en la exposición de los resultados a los que ha llevado el trabajo de investigación de esta tesis doctoral. Por un lado, en los capítulos 3 y 4 se exponían un formalismo matemático para la deconstrucción del C-espacio y los algoritmos resultantes de su aplicación a distintas estructuras robóticas. Por otro, en el capítulo anterior se valoraban las distintas posibilidades de cálculo paralelo inherentes al método de la deconstrucción y se hacía especial hincapié en el diseño e implementación del algoritmo paralelo para la evaluación del espacio de las configuraciones de un robot PUMA redundante.

Por tanto, la primera sección de este capítulo se dedicará a mostrar y comentar los resultados de los algoritmos presentados en capítulos anteriores. Por razones de espacio sólo se presentan algunos resultados correspondientes a los casos más interesantes.

La segunda sección, siguiendo la misma premisa de concisión, está dedicada únicamente a analizar el rendimiento de la implementación del algoritmo paralelo que se explicó con detalle en el capítulo 6 y que, además, es el caso más representativo.

Finalmente, se añade que los resultados de ambas secciones de obtuvieron en un multiprocesador Origin 200 de Silicon Graphics con cuatro procesadores MIPS R10000 y 768 Mbytes de memoria.

7.1 C-obstáculos

Como ya se ha comentado, se procede en este punto a presentar gráficamente algunos de los resultados que proporcionan los algoritmos obtenidos al aplicar el método de la deconstrucción a diferentes estructuras robóticas.

Es interesante el hecho de que para todos los casos se consideran elementos con forma en la cadena cinemática que constituye el robot, en contra de lo que hacen otros trabajos [Lat91][Cur98], que simplifican cada eslabón como un segmento de línea.

Los C-obstáculos correspondientes a manipuladores bien conocidos, como son el planar y el robot PUMA, son similares a los obtenidos por otros autores.

Por otro lado, es importante destacar que para el caso de los robots redundantes no se conocen representaciones gráficas de los C-obstáculos. Por tanto, ésta es una de las aportaciones más interesantes de la presente memoria, pues, como se verá a continuación, aquí se representa el C-espacio para estructuras de este tipo. Sin embargo, estas representaciones no son sencillas de interpretar y requieren de un análisis pormenorizado para llegar a comprobar que los puntos que conforman los C-obstáculos se correspondenden uno a uno con las configuraciones que producen colisión. Un problema más díficil de resolver surge con la visualización de C-espacios con más de tres grados de libertad.

7.1.1 Manipulador Planar

Para este tipo de manipulador es para el que más resultados se encuentran en la bibliografía existente. Por ello se considera interesante mostrar algunos resultados obtenidos en este trabajo y así constatar su validez.

La figura 7.1 muestra tres C-obstáculos correspondientes a tres obstáculos puntuales situados a diferentes distancias, todos ellos sólo son alcanzables por el segundo elemento del manipulador. Si se comparan los resultados con los de la figura 2.5 (página 31) se puede observar que son completamente coherentes: el C-obstáculo de la izquierda se corresponde con el obstáculo puntual de radio más alejado ya que su forma es casi una línea recta; el C-obstáculo central se corresponde con

el obstáculo de radio más cercano, pues tiene un forma de S invertida muy pronunciada; el tercer C-obstáculo pertenecerá a un obstáculo intermedio entre los dos anteriores.

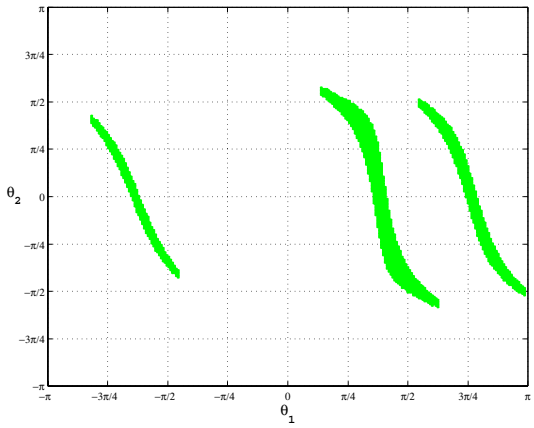


Figura 7.1: Tres C-obstáculos para un manipulador planar

Como ya se ha comentado, los C-obstáculos tienen grosor debido a una implementación más realista, en la que se considera la forma de los eslabones, en lugar de su simplificación a través de un *modelo de alambres*.

Finalmente, se añade que el cálculo para una resolución de 256×256, se realiza en un tiempo promedio de 4.57 s con un consumo de memoria de aproximadamente 5 Mbytes. Esto ya supone un avance respecto al método propuesto por Curto [Cur98] donde sólo se daban resultados para una resolución de 128×128, con un tiempo de 3.04 s frente a los 0.68 s que tarda la implementación de la deconstrucción para tal resolución (con un cosumo aproximado de 3 Mbytes).

7.1.2 Robot PUMA

Este tipo de manipulador es uno de los más extendidos en la industria. De esta forma, constituye un elemento de prueba válido y de gran interés para mostrar el carácter general del método propuesto.

Se considera una porción de corona circular situada a una distancia del hombro suficiente para que únicamente suponga un obstáculo para el tercer elemento del manipulador (figura 7.2). El centro de coordenadas se considera en la articulacion del hombro. Se define de una anchura pequeña para poder obtener curvas semejantes a las que se obtienen en el robot planar para un determinado valor de la cintura.

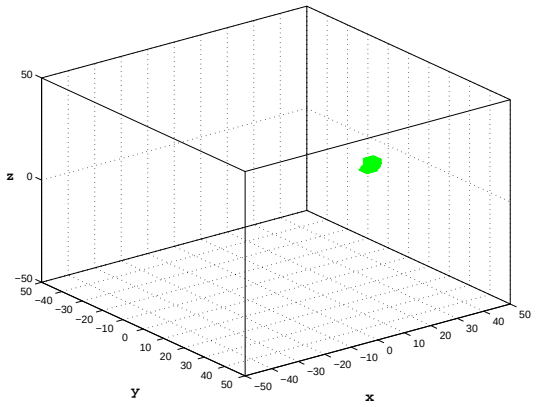


Figura 7.2: Obstáculo con forma de sector de corona en coordenadas cartesianas

Una representación que resulta útil es la de los valores de las coordenadas esféricas de los puntos que constituyen los obstáculos. Así, se presenta en la figura 7.3 la representación en esféricas del obstáculo propuesto. Su interés se debe a que se utilizan estas coordenadas para realizar el producto de convolución de las funciones que representan al robot y a los obstáculos, y, lo que es má importante, permite una comprobación visual directa de que los C-obstáculos resultantes ocupan el lugar correcto en el C-espacio.

En la figura 7.4 se tiene la representación de los dos C-obstáculos que genera la porción de corona. En ella se pueden observar dos siluetas con forma de S invertida que incluyen las cuatro configuraciones posibles del robot que producen un contacto con el obstáculo: *codo arriba/abajo, configuración directa/inversa*. Para cada una de estas últimas se asocia cada

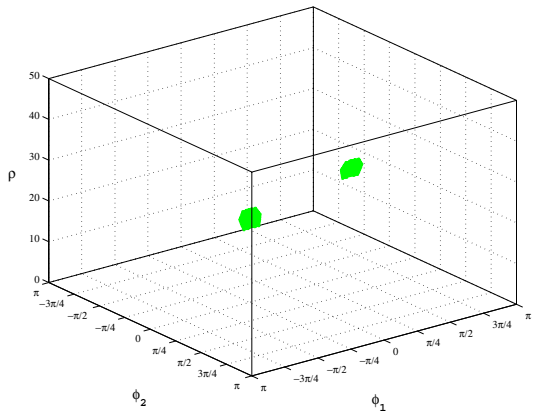


Figura 7.3: Obstáculo de la figura 7.2 en coordenadas esféricas

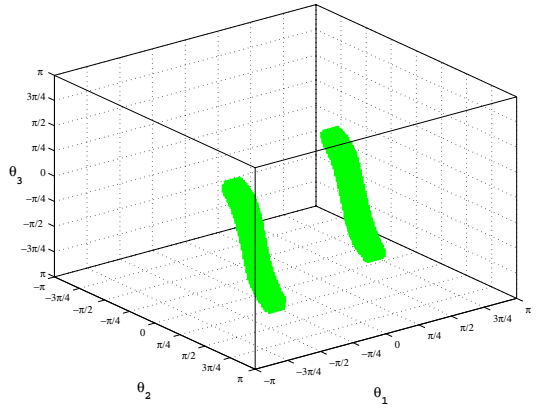


Figura 7.4: C-obstáculos para un robot PUMA para un sector de corona

una de las curvas y la posición de *codo arriba/abajo* se diferencia analizando el valor del ángulo de la articulación θ_3.

Comparando las figuras 7.4 y 7.3, se puede comprobar que los dos C-obstáculos se distribuyen en los valores de θ_1 iguales a los valores φ_1 de los puntos que forman los obstáculos, es

decir, para un ángulo de giro de la cintura que suponga una colisión del tercer eslabón del robot.

Finalmente, se debe comentar que estos resultados se corresponden —salvando el uso de elementos con forma— con los obtenidos en [Cur98]. Una vez más, para una resolución de $128 \times 128 \times 128$, los resultados[1], 242.99 s (con 35 Mbytes de consumo de memoria), son mejores que los obtenidos por Curto, 324.78 s / 375.04 s.

Resolución y Tiempo

El tiempo de cálculo es una limitación muy importante a la hora de aplicar en la realidad estos algoritmos, si bien en muchos casos no es necesaria una resolución de discretización excesivamente alta.

Como curiosidad se añade que para una resolución de $256 \times 256 \times 256$, se consumen 260 Mbytes y se tarda un tiempo promedio 1 hora y 20 minutos.

7.1.3 Manipulador Planar Redundante de Tres Elementos

Para este tipo de manipulador se han obtenido unos resultados muy interesantes y que suponen una novedad en cuanto a la representación del espacio de las configuraciones. No se ha encontrando en la bibliografía ningún caso en que se represente el C-espacio para robots redundantes, por lo que esta sección y la siguiente constituyen una de las partes más relevantes de este trabajo.

Aunque se trata de un robot redundante, todavía se puede utilizar la tradicional representación tridimensional, pues el C-espacio asociado a un manipulador planar con tres DOFs tiene dimensión 3. No obstante, la interpretación de los resultados no es sencilla, por lo que únicamente se van a considerar en esta memoria obstáculos puntuales, para facilitar la comprensión al lector.

Además, para ayudar en el último aspecto mencionado, se van a presentar por orden los C-obstáculos cuando proceden de un obstáculo situado fuera del radio de alcance de los dos primeros elementos (tipo O_{2d}-*3*), después para un obstáculo colocado fuera del radio de alcance del primer elemento (tipo O_{2d}-*2*), y para finalizar, se considerará un obstáculo en el radio de alcance del primer eslabón (tipo O_{2d}-*1*).

[1] Para un obstáculo puntual se tarda un promedio de 107.22 s.

Obstáculo Puntual Tipo O_{2d}-3

En las figuras 7.5, 7.6 y 7.7 se tiene la representación del mismo C-obstáculo desde distintos puntos de vista, correspondiente a un obstáculo puntual sólo alcanzable con el tercer elemento de la cadena del robot. En la figura 7.5 se observa el C-obstáculo de perfil, donde se puede observar la típica forma de S invertida.

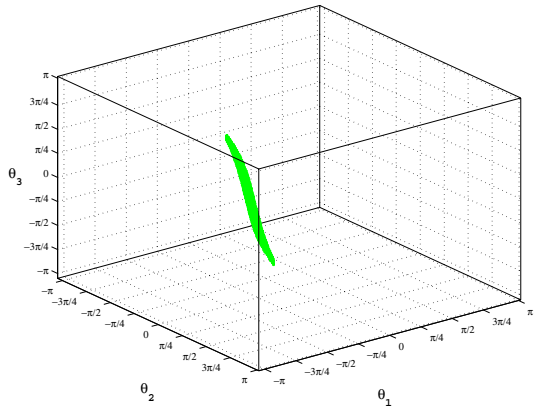


Figura 7.5: C-obstáculo para un robot planar redundante correspondiente a un obstáculo tipo O_{2d}-3

Sin embargo, existirá un intervalo de valores de θ_1 en los que se tendrá una S invertida (configuraciones *codo arriba/abajo para un valor de* θ_1 *fijo* de los otros dos elementos de la cadena). Así, si se cambia el punto de vista, figura 7.6, se puede ver la superficie del C-obstáculo que resulta al considerar cada S invertida, debida a un valor de θ_1, junto a la siguiente. Para ayudar a visualizar este concepto, en la figura 7.7 se han utilizado colores alternos para separar la parte correspondiente a cada valor de θ_1.

Finalmente, se puede añadir que para obtener estos resultados (resolución $128 \times 128 \times 128$) se requiere un tiempo medio de computación de 106.91 s y un consumo de aproximadamente 21 Mbytes de memoria.

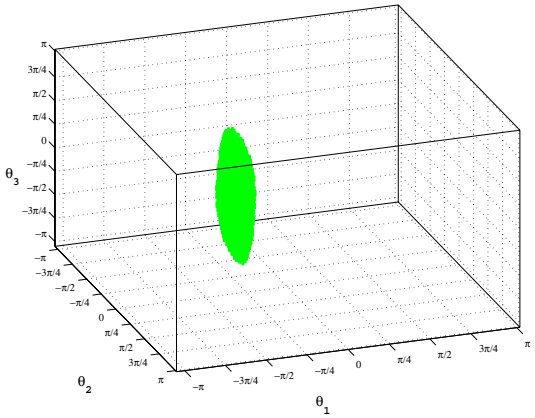


Figura 7.6: Otro punto de vista para el mismo C-obstáculo

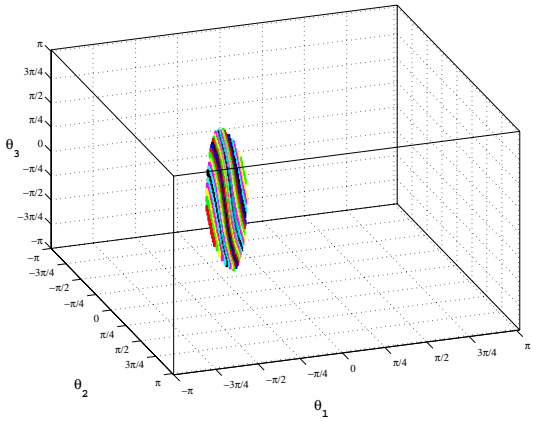


Figura 7.7: Porciones de C-obstáculo para cada valor de θ_1

Obstáculo Puntual Tipo O_{2d}-2

En este caso el C-obstáculo (figura 7.8) es más grande porque el número de configuraciones a las que afecta un obstáculo puntual fuera del radio de alcance del primer elemento es mucho mayor que en el caso anterior.

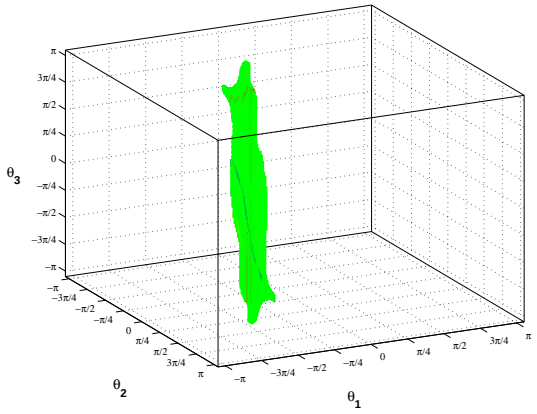


Figura 7.8: C-obstáculo tipo O_{2d}-*2* para un robot planar

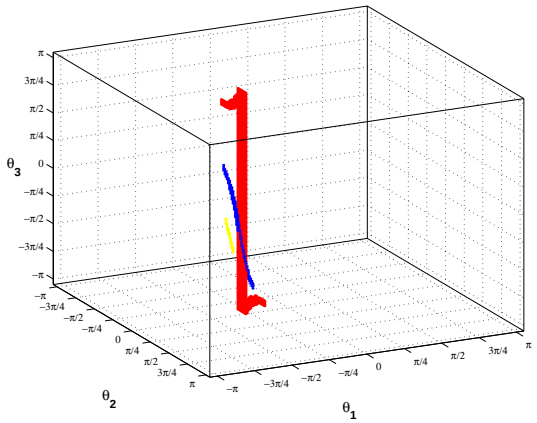


Figura 7.9: Porciones de C-obstáculo para tres valores de θ_1

Si se analiza la figura 7.8 con detenimiento se puede observar que la forma se debe a la contribución de los dos tipos de colisiones que aparecían en la figura 2.5 (página 31). Para ayudar a reconocerlas se muestran en la figura 7.9 las porciones de C-obstáculo para tres valores de θ_1. En rojo, se ve el intervalo de valores de θ_2 para los que el segundo elemento colisiona.

Para estos cualquier valor de θ_3 sigue siendo una configuración con colisión, por eso existen líneas que recorren todo el rango de este DOF. Los extremos de esa figura son debidos, siempre para el mismo valor de θ_1 a diferentes configuraciones de (θ_2, θ_3) que hacen que la colisión se produzca con el tercer elemento del robot.

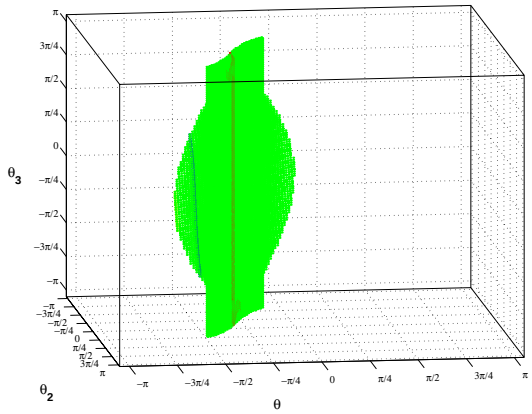


Figura 7.10: Otro punto de vista para el mismo C-obstáculo

En azul y amarillo se muestran dos ejemplos del otro tipo de colisión, la bien conocida S invertida, que en este caso se corresponde a colisiones del tercer elemento, pero para las que si se mantiene el valor de θ_1 no existe configuración posible que haga colisionar al elemento segundo. En las figuras 7.10 y 7.11 se tiene una vista diferente que permite apreciar mejor las formas del C-obstáculo completo (figura 7.8) y de las porciones comentadas (figura 7.9), respectivamente.

Obstáculo Puntual Tipo O_{2d}-1

De nuevo, el tamaño del C-obstáculo (figura 7.12) es mayor. Esto es lógico, pues el obstáculo está en el radio de alcance del primer elemento, por lo que para cada configuración de θ_1 en que se produzca colisión, se generan planos (en rojo en la figura) de C-obstáculo ocupando todos los valores posibles de (θ_2, θ_3). A estas porciones de C-obstáculo hay que

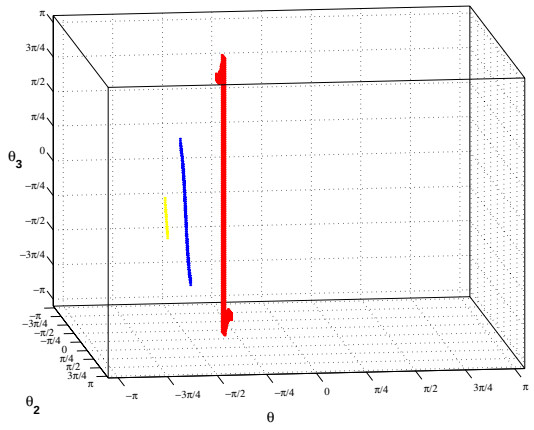


Figura 7.11: Otro punto de vista para las tres porciones de C-obstáculo

añadir las debidas al tercer elemento —el robot se repliega sobre sí mismo—; éstas tienen forma de S invertida (en azul y amarillo), y son de gran tamaño porque el número de configuraciones $(\theta_1, \theta_2, \theta_3)$ que producen este tipo de colisión es elevado. Por otro lado, el obstáculo está tan cerca de la primera articulación que nunca es posible que el segundo elemento colisiones con él.

En la figura 7.13 se proporciona otro punto de vista con tres porciones de C-obstáculo debidas a tres valores de configuración de θ_1. En rojo se puede observar el plano generado por una colisión con el primer elemento. En azul y amarillo se ilustra el otro tipo de colisión comentado; desde esta otra posición se puede ver cómo la forma de S es continua para un intervalo de valores de θ_2, pero se divide en dos partes debido a que en la representación $+\pi$ no aparece como vecino de $-\pi$.

7.1.4 Robot PUMA Redundante de Cuatro Elementos

Finalmente, se comentan los resultados para la cadena cinemática más complicada de las estudiadas a lo largo de este trabajo. Como el C-espacio de un PUMA con cuatro DOFs es de

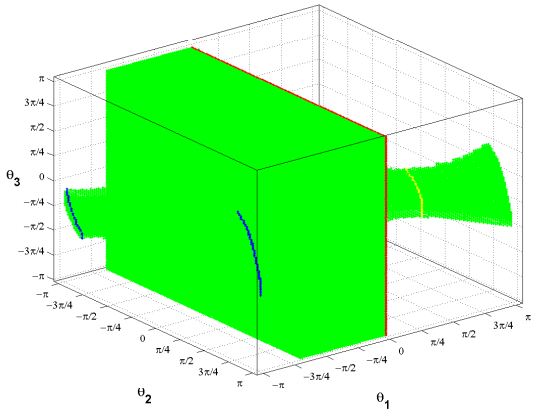


Figura 7.12: C-obstáculo tipo O_{2d}-*1* para un robot planar

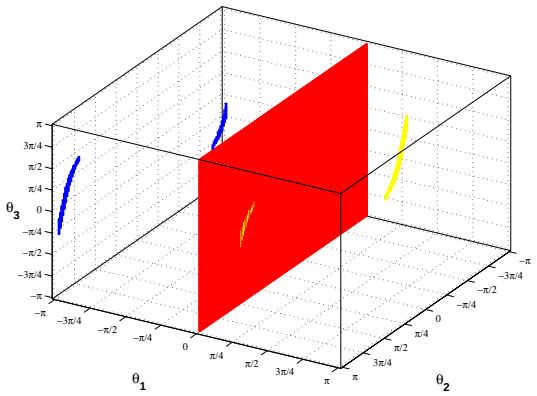


Figura 7.13: Tres porciones de C-obstáculo desde otro punto de vista

dimensión 4 se plantea un problema de representación y de interpretación.

Si se recuerda que no se consideran las colisiones con el primer elemento, entonces se tienen tres tipos de obstáculos puntuales a estudiar: obstáculo tipo O_{3d}-*4*, fuera del radio de alcance de los tres primeros elementos; obstáculo tipo O_{3d}-

3, fuera del radio de alcance de los dos primeros elementos; y obstáculo tipo O_{3d}-*2*, en el radio de alcance del segundo elemento.

Para el estudio de los C-obstáculos obtenidos se tiene en cuenta que el robot considerado es equivalente al manipulador planar redundante una vez que la cintura se ha colocado en una posición, determinando el plano de acción. Si el obstáculo es alcanzable, como se trabaja en una esfera, será alcanzable desde dos configuraciones distintas del robot: *directa e inversa*; lo que corresponde a dos valores de θ_1. Como consecuencia, en los C-obstáculos, utilizando un color distinto para cada valor de θ_1 apareceran formas semejantes a las que aparecían para el robot planar de tres articulaciones.

Obstáculo Puntual Tipo O_{3d}-*4*

En la figura 7.14, el C-obstáculo que aparece está constituido por dos porciones con la misma forma que un C-obstáculo tipo O_{2d}-*3*, que se estudió para el manipulador planar redundante (figura 7.6).

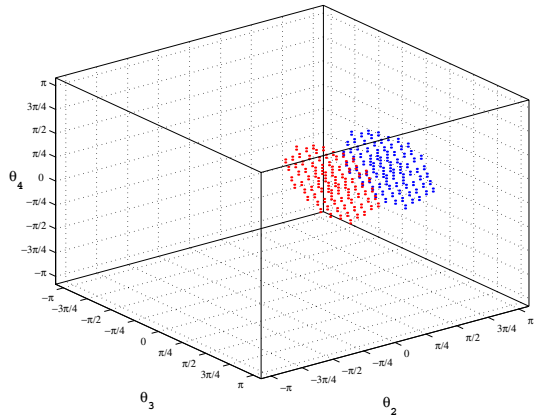


Figura 7.14: Las dos porciones de un C-obstáculo tipo O_{3d}-*4* para un robot PUMA de 4 DOFs

Obstáculo Puntual Tipo O_{3d}-3

Como se puede observar en la figura 7.15, para este caso el C-obstáculo está formado por dos porciones similares a los C-obstáculos tipo O_{2d}-2 que se vieron en la sección anterior (7.10). Así, se pueden observar las porciones con forma de S invertida, que se corresponden a colisiones con el cuarto elemento, así como las típicas barreras, que se corresponden a colisiones con el tercer elemento.

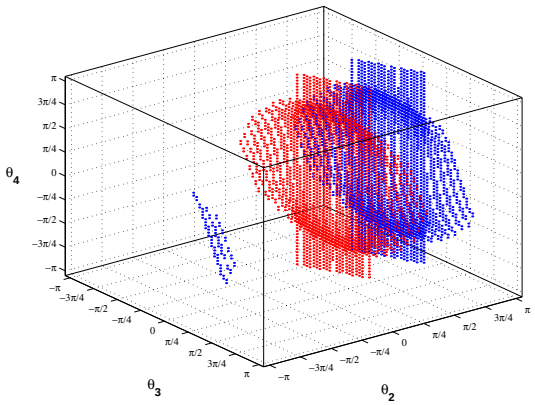


Figura 7.15: Las dos porciones de un C-obstáculo tipo O_{3d}-3 para un robot PUMA de 4 DOFs

Obstáculo Puntual Tipo O_{3d}-2

Finalmente, el C-obstáculo que aparece en la figura 7.16 está formado por dos porciones donde se pueden reconocer dos C-obstáculos de tipo O_{2d}-1, como se vieron en la sección anterior (figura 7.12). En esta última representación es más patente la dificultad de la visualización para C-espacios de más de tres dimensiones. Gracias al uso del color se puden distinguir los muros típicos de un C-obstáculo tipo O_{2d}-1, pero encontrar colisones particulares en la figura es una tarea muy complicada.

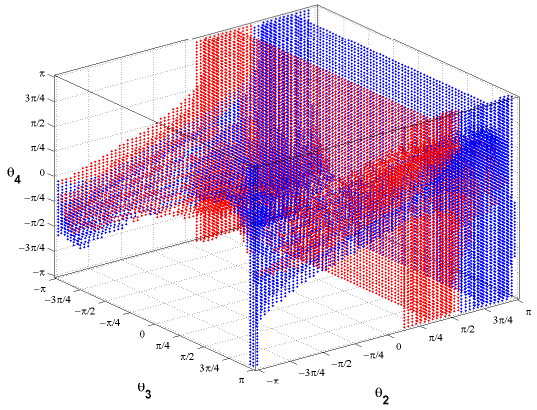


Figura 7.16: Las dos porciones de un C-obstáculo tipo O_{3d}-*2* para un robot PUMA de 4 DOFs

7.2 Rendimiento Paralelo

En este apartado se presentan los resultados de los experimentos realizados en la implementación de un algoritmo paralelo para el cálculo del espacio de las configuraciones de un robot PUMA redundante, así como un análisis de los mismos.

Primeramente, hay que comentar que aunque todas las pruebas se realizaron en una máquina multiprocesadora (Origin 200 de Silicon Graphics con cuatro procesadores MIPS R10000 y 768 Mbytes de memoria), la implementación es independiente de la plataforma paralela elegida.

Por otro lado, aunque en el capítulo 5 se comentó que la biblioteca para el cálculo de la transformada rápida de Fourier utilizada, FFTW, proporciona versiones paralelas, en [The00] se comprobó que para poder explotar el paralelismo a este nivel se necesita un elevado número de procesadores. Como en el entorno en que se desarrollaron las pruebas sólo se podía contar con 4 procesadores, había que tomar una decisión de compromiso: demostrar las ventajas que aportaba un único nivel de paralelismo, ya que el nivel de transformadas de Fourier no supone, con dichas bibliotecas, ventajas significativas. Por tanto, se realizaron todos los experimentos con la versión secuencial

de la biblioteca FFTW, por ser la que mejores resultados aporta en las condiciones expuestas.

En la programación paralela el objetivo no es sólo optimizar una métrica como la velocidad, sino que un buen diseño tratará de ser óptimo en función del tiempo de ejecución, de los requisitos de memoria, de los costes de implementación, de los costes de mantenimiento, etc. Sin embargo, el esfuerzo principal de este trabajo se ha dedicado, en primer lugar, al desarrollo de un método general para evaluar el espacio de las configuraciones para cualquier estructura robótica, y en segundo lugar, a demostrar su caracter fuertemente paralelo.

Así, aunque en el diseño se han tenido en cuenta aspectos esenciales de ahorro de memoria o de cantidad de datos comunicados entre los distintos elementos de procesamiento, a la hora de la implementación ha primado el producir un código flexible, aplicable —con pequeñas modificaciones— a la totalidad de los casos estudiados, antes que llegar a un código óptimo en cuanto al consumo de memoria, por ejemplo; esta labor de optimización se ha postergado a futuros desarrollos.

Otro aspecto importante a tener en cuenta es la dimensión del C-espacio para un robot PUMA redundante, que es 4. El entorno de pruebas está limitado a 768 Mbytes, y debido a la implementación elegida, se guarda cada punto del C-espacio en un entero[2]. Trabajar con una resolución de 256 resulta prohibitivo, pues sólo para almacenar el resultado de la evaluación del C-espacio, guardando cada punto en un bit, se se necesitarían $256^4 = 512$Mbytes. A esta cantidad hay que añadir todas las variables intermedias que hay que utilizar: para almacenar los puntos del espacio de trabajo, los puntos de cada elemento del robot, las tranformadas que se van calculando, etc.

Además, trabajar con una resolución tan grande supondría, dado el algoritmo paralelo diseñado, tener la posibilidad de realizar $256^2 = 65536$ tareas, a repartir entre tan sólo 4 procesadores.

Teniendo en cuenta esta limitación de elementos de procesamiento, es suficiente, a efectos de un análisis del rendimiento

[2]Hubiera sido más óptimo guardar cada punto en un bit, y más óptimo aún, utilizar matrices *sparse*, pero dada la forma en que trabaja la biblioteca FFTW, y debido a otros problemas de carácter técnico, se optó por el método más directo y flexible.

paralelo, estudiar el caso de la resolución 64, lo cual proporciona $64^2 = 4096$ tareas a repartir.

El siguiente paso, por tanto, es la evaluación de las diferentes implementaciones del algoritmo paralelo. Para ello, se tienen que estudiar las distintas componentes del tiempo de ejecución: tiempo de cálculo, de comunicación y de inactividad. Todos ellos deben ser función del número de procesadores, del tamaño del problema y del número y tamaño de las tareas. Además, se realizará un estudio respecto a la aglomeración considerando dos tipos de grano, grano fino (una única configuración (θ_1, θ_2) por tarea) y grano grueso (dos o más configuraciones).

Se debe destacar que todos los tiempos que se especifican son valores medios obtenidos a partir de un conjunto de diez pruebas por experimento, de las cuales se eliminan los correspondientes al mejor y al peor tiempo. Además, las pruebas se planatean para los peores casos, pues se considera únicamente un obstáculo puntual en el radio del hombro, con lo que la mayoría de las configuraciones son libres y, por tanto, se debe evaluar el resto del C-espacio para los elementos tercero y cuarto.

7.2.1 Grano Fino

Se corresponde al caso más simple: una resolución de $64 \times 64 \times 64 \times 64$ con un grano fino (una configuración (θ_1, θ_2) por tarea). En la tabla 7.1 se muestran los tiempos obtenidos para un conjunto de situaciones elegidas, desde la más simple, un maestro y un esclavo, hasta la más compleja, un maestro y cuatro esclavos.

El consumo de memoria es de aproximadamente 73 Mbytes para el proceso servidor más 3 Mbytes por cada proceso esclavo.

Como se muestra en la tabla 7.1, el resultado con menor tiempo de cálculo se obtiene cuando hay cinco procesos, uno de ellos maestro y los otros cuatro realizando los cálculos. Además, es interesante analizar la tendencia en esta evolución. Así, se produce una gran reducción del tiempo cuando el número de esclavos es mayor que uno (nótese que el tiempo para la situación con un maestro y un esclavo equivaldría al resultante de sumar al tiempo de ejecución del algoritmo se-

Tiempo (s)	N^o Procesos (1 maestro)	N^o Procesadores
506.00	2	2
254.71	3	3
170.62	4	4
126.51	5	4
Secuencial: 500.04 s		

Tabla 7.1: Tiempos de ejecución para una implementación de grano fino y una resolución de $64 \times 64 \times 64 \times 64$

cuencial los tiempos debidos a las comunicaciones). Cuando se tienen dos esclavos o más crece el grado de concurrencia. Por otro lado, se puede observar que cuando se pasa de un esclavo a dos, se reduce el tiempo de cálculo un 50 %; si se aumenta el número de esclavos a tres, la reducción es de un 33 %; finalmente si se aumenta en uno el número de procesos —aunque sólo se disponga de cuatro procesadores— la reducción es de un 25 %. Esto se debe a que no hay penalización producida por las comunicaciones frente a los cálculos. Además, es interesante comprobar que cuando el número de esclavos es igual al de procesadores, el proceso maestro puede compartir un procesador con uno de los procesos esclavos, ya que pasa la mayor parte del tiempo esperando.

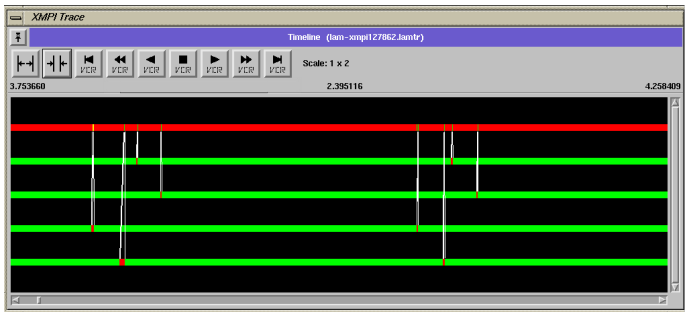


Figura 7.17: Instantánea de ejecución: grano fino, resolución $64 \times 64 \times 64 \times 64$, un maestro y cuatro esclavos

Como ya se comentó en el capítulo 5, se utiliza la herramienta XMPI para la visualización de la traza de cada una de las ejecuciones. Ésta muestra tres tipos de estados para un proceso: bloqueado esperando una comunicación, en rojo; calculando, en verde; y en amarillo, ocupado en operaciones de MPI —sobrecarga del sistema: conversión de datos, empaquetado de paquetes, etcétera.

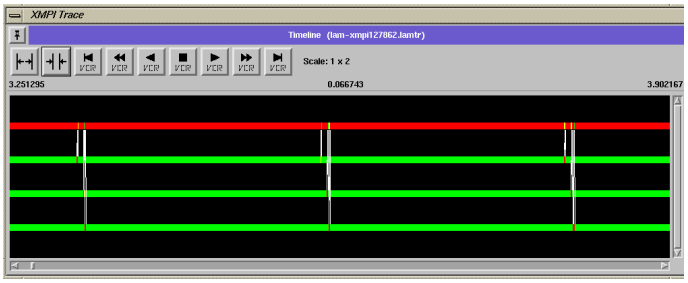


Figura 7.18: Instantánea de ejecución: grano fino, resolución $64 \times 64 \times 64 \times 64$, un maestro y tres esclavos

En las figuras 7.17, 7.18 y 7.19 se muestra la ejecución y comunicaciones del experimento para las combinaciones maestro-esclavo citadas (exceptuando el caso de tener un maestro y un esclavo). Como se puede observar, el proceso maestro (en la primera fila) pasa la mayor parte del tiempo inactivo (color rojo), esperando las comunicaciones de resultados por parte de los esclavos, y a continuación les asigna nueva tarea.

En cuanto a los esclavos, se puede observar un predominio de periodos de trabajo, frente a los tiempos de comunicaciones, que en comparación son despreciables.

Por lo tanto el balance de carga para el caso de grano fino —como predijo la simulación con las redes de Petri— es óptimo y se consigue un tiempo total de cálculo excelente. Además, es previsible que al aumentar el número de procesadores se reduzcan los tiempos de cálculo.

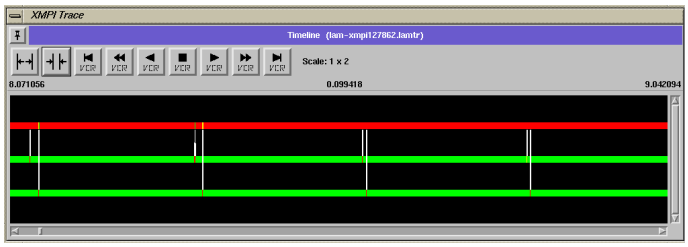


Figura 7.19: Instantánea de ejecución: grano fino, resolución $64 \times 64 \times 64 \times 64$, un maestro y dos esclavos

7.2.2 Aglomeración. Grano Grueso

Como ya se explicó en el capítulo anterior, un elemento importante a considerar en el diseño de algoritmos paralelos es la aglomeración. Así, un factor interesante para estudiar es la influencia del tamaño del grano en los tiempos de ejecución.

Hay que tener en cuenta que el maestro adapta el tamaño del grano para el que ha sido configurado. Así, en tiempo de ejecución calcula cuál es el tamaño más próximo por debajo del tamaño inicial, en función del número de esclavos con los que se va a trabajar y del número de configuraciones (θ_1, θ_2) libres; este procedimiento asegura un adecuado balance de la carga. No obstante, dejando momentáneamente aparte las penalizaciones de comunicación, dependiendo del número de tareas que finalmente tenga que realizar cada esclavo, será mejor una elección del tamaño del grano que otra.

La tabla 7.2 explica este concepto. Como se ha explicado, se considera la mejor opción, tener cinco procesos, de los cuales uno dirige la ejecución del resto. Si se parte de un número de tareas de grano fino igual a 4092, se pueden elegir granos de tamaño muy variado. En función del tamaño elegido, el resultado será un número igual de tareas de grano grueso para cada esclavo o puede producirse un desequilibrio.

- Si todos los esclavos tocan al mismo número de tareas, se espera un rendimiento óptimo para ese tamaño de grano.
- Si hay dos esclavos que tienen que hacer una tarea más que los otros dos, al final de la ejecución no se aprovechan

Grano	Tareas	Reparto	Estimación
1	**4092**	$4 \rightarrow 1023$	**Óptimo**
2	2046	$2 \rightarrow 511, 2 \rightarrow 512$	Normal
3	1364	$4 \rightarrow 341$	Óptimo
6	682	$2 \rightarrow 170, 2 \rightarrow 171$	Normal
11	372	$4 \rightarrow 93$	Óptimo
12	341	$3 \rightarrow 85, 1 \rightarrow 86$	Peor
22	186	$2 \rightarrow 46, 2 \rightarrow 47$	Normal
31	124	$4 \rightarrow 33$	Óptimo
33	124	$4 \rightarrow 31$	Óptimo
44	93	$3 \rightarrow 23, 1 \rightarrow 24$	Peor
93	44	$4 \rightarrow 11$	Óptimo
186	22	$2 \rightarrow 5, 2 \rightarrow 6$	Normal
341	12	$4 \rightarrow 3$	Óptimo
372	11	$3 \rightarrow 3, 1 \rightarrow 2$	Mejor
682	6	$2 \rightarrow 2, 2 \rightarrow 1$	Normal
1023	4	$4 \rightarrow 1$	Óptimo

Tabla 7.2: Estimación de rendimiento para diferentes tamaño de grano grueso

los cuatro procesadores disponibles sino sólo dos.

- Si hay tres procesos que tienen que calcular una tarea más que el esclavo restante, se trata de de un caso mejor que el anterior, pues se aprovechan tres de los cuatro procesadores al final de la ejecución.
- Por último, si hay un esclavo que realiza una tarea más que el resto, esta tarea final es equivalente a hacerla secuencialmente más el coste de la comunicación del resultado.

Así, en la tabla 7.2 la última columna se estima cómo va a ser el tiempo de ejecución en relación con los tamaños de grano inferiores. Además, se ha de tener en cuenta la influencia de las comunicaciones de resultados, más voluminosos cuanto mayor es el tamaño del grano. En la figura **??** se presenta la gráfica de tiempos de comunicaciones consumidos para enviar

una tarea de cada uno de los granos que se están estudiando. Para calcular estos tiempos se hizo un test ping-pong de mil envíos de cada uno de los tamaños y se calculó la media; los valores aparecen en la tabla 7.3. Además, en ella aparecen los tiempos de cálculo asociados a una tarea de cada uno de los granos que se estudia, cuyo tamaño también se muestra en la tabla.

Grano	T_{com}	T_{cal}	Tamaño
1	4403 μs	0.13 s	16 Kbytes
2	5916 μs	0.25 s	32 Kbytes
3	7517 μs	0.37 s	48 Kbytes
6	11.94 ms	0.75 s	96 Kbytes
11	19.63 ms	1.36 s	176 Kbytes
12	21.37 ms	1.50 s	192 Kbytes
22	35.46 ms	2.71 s	352 Kbytes
31	50.25 ms	3.79 s	496 Kbytes
33	53.28 ms	4.03 s	528 Kbytes
44	69.05 ms	5.40 s	704 Kbytes
93	142.74 ms	11.30 s	1.45 Mbytes
186	286.73 ms	22.80 s	2.90 Mbytes
341	516.17 ms	41.62 s	5.33 Mbytes
372	569.63 ms	45.68 s	5.81 Mbytes
682	1.03 s	83.57 s	10.66 Mbytes
1023	1.50 s	125.13 s	15.98 Mbytes

Tabla 7.3: Tiempo de comunicaciones, tiempo de cálculo y tamaño de una tarea para para diferentes tamaño de grano grueso

En la figura 7.21 se representan los tiempos de cálculo de tareas de diferentes granos, mientras que en la figura 7.22 se muestra la relación de porcentaje de tiempo que suponen las comunicaciones respecto al cálculo para una tarea de diferente tamaño. En esta gráfica se observa que los porcentajes están entre 3.5, correspondiente al grano fino, y 1.2 del mayor tamño de grano, lo que significa que no hay una variación sustancial de la influencia de las comunicaciones.

Sin embargo, aunque aperentemente es mejor comunicar tamaños de grano mayores, si en algún punto de la ejecución

dos esclavos envían sus resultados en instantes muy próximos, la penalización será mayor cuanto mayor sea el tamaño del grano.

De este modo, se elaboraron diferentes pruebas para comprobar este comportamiento del algoritmo. La tabla 7.4 muestra los resultados, que responden a lo previsto. En la misma tabla se dan los consumos de memoria asociados a cada tamaño de grano implementado.

Tiempo (s)	Grano	Mbytes por proceso	*Speedup*
126.51	1, 2, 3	3	**3.95**
131.40	6	3.1	3.81
129.82	11	3.1	3.94
130.13	12	3.1	3.84
129.25	22	3.3	3.87
127.47	31	3.4	3.92
127.94	33	3.5	3.91
131.84	44	3.7	3.79
127.68	93	4	3.92
138.70	186	5.9	3.61
127.76	341	8.4	3.91
139.47	372	8.9	3.59
168.38	682	14	2.97
132.13	1023	19	3.78
Secuencial: 500.04 s Consumo básico: 73 Mbytes			

Tabla 7.4: Tiempos de cálculo y consumos de memoria para grano grueso y una resolución de $64 \times 64 \times 64 \times 64$

7.2.3 Grano Fino vs. Grano Grueso

Como se puede observar en la figura 7.23, el comportamiento de la aplicación con grano grueso (6 tareas), un caso no óptimo, no hace variar sustancialmente el rendimiento (es un 3.5 % más lento) con respecto al caso equivalente con grano fino (figura 7.17). Basta con comparar los periodos de cálculo con los de comunicaciones en estas gráficas para comprobar que las últimas toman un tiempo my pequeño comparado con

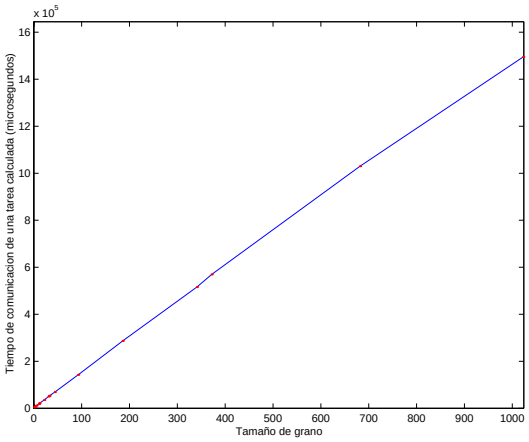


Figura 7.20: Tiempo de comunicación de una tarea con diferente tamaño de grano

el tiempo de cálculo, lo que permite al maestro dar servicio a los esclavos con la suficiente rapidez, evitando que los esclavos pasen largos intervalos inactivos a la espera de que el maestro les asigne más trabajo. En la figura 7.24 se muestra otro caso distinto: un grano de 682 para dos procesos esclavos.

Por tanto, a la vista de los resultados, se concluye que cuanto más pequeño sea el grano, menor será la influencia del tiempo de comunicación, y mejor el balance de carga. Si se utilizan granos gruesos se debe garantizar que exista una equipartición entre esclavos, y aún así, si por alguna razón (máquina no dedicada, por ejemplo) algún esclavo se retrasa y coinciden dos envíos de resultados, el rendimiento se verá afectado.

En cuanto al tamaño del grano, el mejor caso se da para un tamaño igual a 2 o 3, con tiempos muy similares a los tiempos que alcanza el grano unidad. Esta tendencia es lógica, ya que 2 o 3 configuraciones entre 4092, junto con el pequeño coste de comunicaciones que supone, se pueden considerar prácticamente grano fino.

En cualquier caso parece garantizado que el algoritmo sea también escalable para grano grueso, aunque a la vista de los tiempos obtenidos y los consumos de memoria, parece más

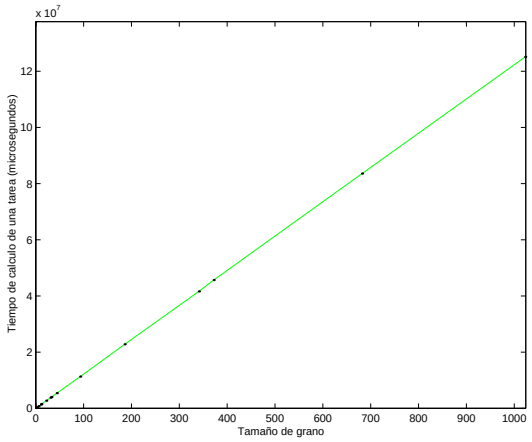


Figura 7.21: Tiempo de cálculo de una tarea con diferente tamaño de grano

razonable utilizar siempre grano fino.

Finalmente, es interesante resaltar que para el caso en que se reparte exactamente el trabajo en cuatro grandes tareas (grano 1023 en el ejemplo de la tablas 7.2 y 7.4), se aprovecha al máximo el cálculo concurrente, aunque al final se paga un precio alto para comunicar los resultados; no obstante, este tamaño es una solución mejor que granos mucho menores.

7.2.4 *Speedup*

Para medir el rendimiento del algoritmo y una vez vistos los problemas de las comunicaciones y comentados los mejores casos, se exponen en la tabla 7.4 los tiempos de cálculo obtenidos en la ejecución de los algoritmos paralelos con diferentes granos en comparación con el tiempo que consume el algoritmo secuencial.

Analizando los tiempos obtenidos para las diferentes configuraciones maestro-esclavos se obtienen *speedups* relevantes. Sirvan de ejemplos los valores de la tabla 7.1, donde para la explotación de dos procesadores (un maestro y dos esclavos) se obtiene un *speedup* de 1.96, mientras que para la explotación

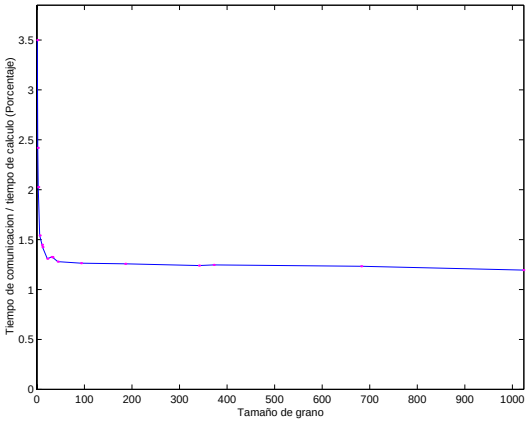


Figura 7.22: Relación en tanto por ciento entre el tiempo de comunicación y el tiempo de cálculo para diferentes tamaños de grano

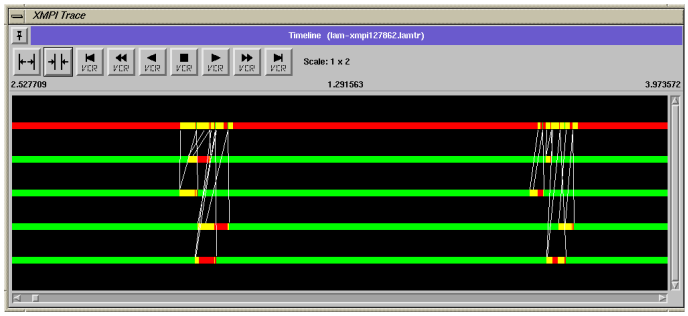


Figura 7.23: Instantánea de ejecución: grano grueso de tamaño 6, resolución $64 \times 64 \times 64 \times 64$, un maestro y cuatro esclavos

de tres procesadores (un maestro y tres esclavos) se obtiene un valor de 2.93.

Para todas las situaciones, se trabajó con una resolución de $64 \times 64 \times 64 \times 64$ y se utilizó el mismo obstáculo puntual, que generaba una totalidad de 4092 tareas. Las versiones paralelas utilizaban el esquema que demostró ser el óptimo, esto es, un

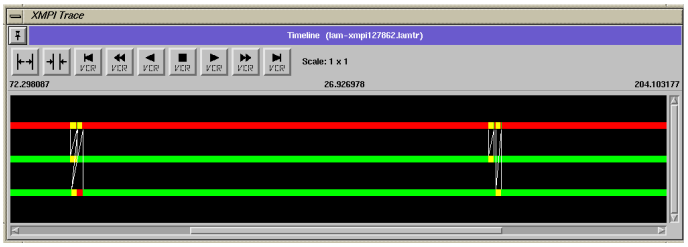


Figura 7.24: Instantánea de ejecución: grano grueso de 682, resolución $64 \times 64 \times 64 \times 64$, un maestro y dos esclavos

proceso maestro y cuatro esclavos.

En el mejor caso, correspondiente a un grano fino o tamaños muy pequeños, se ha conseguido alcanzar aproximadamente una reducción del tiempo de ejecución a una cuarta parte del tiempo del algoritmo secuencial, lo que supone alcanzar el límite teórico para una máquina tetraprocesadora, y coincide con la simulación realizada mediante las redes de Petri coloreadas que se explicó en el capítulo anterior. Además, se observa una tendencia decreciente en el *speedup* —incremento de velocidad de ejecución en un programa paralelo— con el aumento del grano, debida a que las comunicaciones penalizan cuando coinciden comunicaciones de resultados parciales de tamaños muy grandes.

Además, la eficiencia, que se puede obtener dividiendo el *speedup* entre el número de procesadores, es prácticamente 1 (0.99) en el mejor de los casos, y para el resto se mantiene en valores muy altos.

Otro resultado importante es que, gracias al diseño realizado, se puede asegurar que el algoritmo es escalable y que su eficiencia aumentará al incrementar el número de procesadores y al ampliar la resolución. Sin embargo, trabajar con resoluciones mayores (128 o 256) coparía la totalidad de la memoria disponible para el entorno de ejecución (más de 740 Mb).

Por tanto, los requisitos de memoria constituyen uno de los cuellos de botella más importantes para este algoritmo.

Finalmente, se puede añadir que la consideración de todos los aspectos expuestos en este capítulo constituye un aliento

para continuar en trabajos futuros con la línea de investigación emprendida.

Capítulo 8

Conclusiones y Trabajos Futuros

Calvin: ¡¿Por qué tengo que trabajar para todo?! ¡Es como decir que no me lo merezco!

En Todas Partes Hay Tesoros
BILL WATTERSON

La realización de este trabajo de investigación ha dado como resultado una serie de aportaciones que se reunen, junto a las conclusiones a las que se ha llegado a la vista de los objetivos alcanzados, en los siguientes puntos:

- Se ha propuesto un nuevo procedimiento general de evaluación de espacio de las configuraciones: la deconstrucción. El nuevo método tiene un carácter sistemático, de forma que permite descomponer un robot en los elementos de la cadena cinemática que lo forman, y evaluar, para cada uno de ellos, el conjunto de configuraciones asociadas a cada elemento que producen colisión.
- Un elemento importante que habilita la deconstrucción reside en el hecho de utilizar para cada eslabón de la cadena cinemática el sistema de referencia que permita calcular las colisiones asociadas a él de forma independiente. El método propuesto utiliza el procedimiento de Denavit-Hartenberg en la determinación de los sistemas de referencia, un procedimiento clásico utilizado comúnmente en la bibliografía.
- La determinación de las colisiones debidas a cada elemento por separado permite disminuir la porción de espacio evaluado, pues una vez que se sabe que un elemento colisiona en una configuración, no es necesario evaluar las colisiones de los elementos siguientes en la cadena, para esa misma configuración.

- Para cada uno de los elementos, el método de cálculo está sustentado por un sólido formalismo matemático, cuya base es el cálculo de los C-obstáculos como una integral extendida sobre el espacio de trabajo del producto de dos funciones, una que describe al elemento y la otra a los obstáculos.

- La deconstrucción permite explotar ventajas de procedimientos existentes en la bibliografía que buscan la optimización de los tiempos de cálculo, ya que en el cálculo básico se hace uso de la convolución, lo que permite utilizar la transformada de Fourier como herramienta matemática, y la Transformada Rápida de Fourier como solución algorítmica.

- El formalismo matemático permite afrontar de forma natural el cálculo de C-obstáculos para robots redundantes, para los que no se conocía hasta ahora ningún trabajo que lo hiciera. Por tanto, el nuevo método propuesto aporta el carácter general del que carecen otros métodos.

- Se ha aplicado el método y se han obtenido algoritmos para el cálculo de espacio de las configuraciones de varios tipos de manipuladores, tanto en espacios bidimensionales como en tridimensionales.

- Se ha validado el método sobre las estructuras cinemáticas más conocidas como son un manipulador planar y un robot PUMA. Posteriormente se ha aplicado, y se han obtenido los correspondientes algoritmos que permiten evaluar el C-espacio, para manipuladores redundantes como un planar redundante y un PUMA redundante. No se conocen trabajos previos que hayan afrontado estas cadenas cinemáticas.

- El tiempo de ejecución de los algoritmos desarrollados es dependiente tanto del número como de la forma de los obstáculos, al contrario que los propuestos por otros autores.

- Se ha hecho un estudio topológico de los C-obstáculos para robots redundantes, dado que en la bibliografía no se ha encontrado, y se ha comprobado que existe una

relación coherente con C-obstáculos conocidos, como son los de un manipulador planar y los de un robot PUMA.

- Se ha constatado el carácter inherentemente paralelo del método de deconstrucción del C-espacio. Tras un análisis detallado, se han identificado tres niveles diferentes de paralelismo: dos asociados al cálculo básico del procedimiento, y uno asociado al procedimiento en sí.

- Se ha estudiado el caso de paralelización del algoritmo para el cálculo del C-espacio de un robot PUMA redundante y se han valorado las diferentes soluciones aplicables.

- En el diseño paralelo se ha llevado a cabo un adecuado estudio del balance de la carga, así como de la granularidad, que ha sido completado con un modelado mediante Redes de Petri, cuyos resultados para tres entornos paralelos diferentes, han hecho prever un rendimiento óptimo para su implementación.

- Se ha implementado el algoritmo paralelo para una máquina tetraprocesadora y se han obtenido valores de *speedup* próximos al máximo. Se ha comprobado que el algoritmo es óptimo cuando se trabaja con grano fino. Y se ha determinado que el uso del grano grueso puede tener para algunos casos una eficiencia cercana a la del grano fino. Además, se han identificado los problemas que pueden surgir cuando se utilizan tamaños de grano grandes.

- Por último, se ha comprobado que el algoritmo, tanto para grano fino como para grano grueso, es escalable.

A partir de las conclusiones se pueden destacar algunos puntos en los que es posible extender el trabajo de investigación recogido en esta memoria:

- Optimizar las implementaciones en lo relativo al consumo de memoria. Estudiar la posibilidad de introducir estructuras jerárquicas en el método de deconstrucción del C-espacio.

- Aplicar los algoritmos paralelos a estructuras robóticas más complejas como pueden ser composiciones de estructuras existentes, para las que cabe esperar resultados similares a los obtenidos en el presente trabajo.
- Aplicar los algoritmos para realizar planificación de movimientos en un entorno estructurado.
- Implementar los algoritmos en entornos debidamente equipados con dispositivos de cálculo especializados de alto rendimiento; en concreto, utilizar procesadores digitales de la señal, DSP, para optimizar los tiempos de cálculo debidos al uso de la FFT.
- Ampliar las plataformas sobre las que probar las implementaciones. Concretamente, se pretende realizar pruebas con resoluciones mucho mayores en sistemas donde la memoria o el número de procesadores no signifique una limitación.
- Proponer y analizar otros diseños no centralizados. Se podrían plantear soluciones que optimicen el rendimiento de los procesadores realizando una gestión dinámica de las tareas. Asimismo, las labores realizadas por el maestro se podrían desdoblar o separar para evitar bloqueos en la recepción de nuevos trabajos.

Bibliografía

[AB88] F. Avnaim and J. D. Boissonnat. Polygon placement under translation and rotation. Technical Report 889, INRIA, Sophia-Antipolis, France, 1988.

[AG97] Juan M. Ahuactzin and Kamal Gupta. A motion planning based approach for inverse kinematics of redundant robots: The kinematic roadmap. In *proceedings of the IEEE International Conference on Robotics and Automation*, pages 3609–3614, 1997.

[Akl92] Selim G. Akl. *Diseño y análisis de algoritmos paralelos.* Ra-Ma, Madrid, 1992.

[AWG83] I. Asimov, P.S. Warrick, and M. H. Greenberg. *Machines That Think.* Holt, Reinhart and Wilson, London, 1983.

[Bag96] Boris Baginski. Local motion planning for manipulators based on shrinking and growing geometry models. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 4, pages 3303–3308, 1996.

[BCP01] Salvatore Concialdi Benedetto Colajanni and Giuseppe Pellitteri. Construction or deconstruction: Wich is the best way to learn architecture. In *Education for Computer Aided Architectural Design in Europe*, pages 299–304, Helsinki, 2001.

[BDJ+00] Gianfranco Balbo, Jörg Desel, Kurt Jensen, Wolfgang Reisig, Grzegorz Rozenberg, and Manuel Sil-

va. *Introductory Tutorial Petri Nets.* 21st International Conference on Application and Theory of Petri Nets, 2000.

[BK00] Robert Bohlin and Lydia E. Kavraki. Path planning using lazy prm. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 1, pages 521–528, 2000.

[BTCM01] F. J. Blanco, R. Therón, B. Curto, and V. Moreno. Towards an effecient use of memory in evaluation of configuration space of a robot. In *Proceedings of the IFAC workshop on mobilr robot technology*, pages 160–165, 2001.

[Can88] J. F. Canny. *The complexity of robot motion planning.* MIT Press, Cambridge, 1988.

[CGK93] D. Challou, M. Gini, and V. Kumar. Parallel search algorithms for robot motion planning. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 2, pages 46–51, 1993.

[CKP+93] David E. Culler, Richard M. Karp, David A. Patterson, Abhijit Sahay, Klaus E. Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten von Eicken. Logp: Towards a realistic model of parallel computation. In *Proceedings of the 4th ACM Symposium On Principles and Practice of Parallel Programming*, pages 1–12, SanDiego, California, 1993.

[Cur98] B. Curto. *Fornalismo matemático para la representación de obstáculos en el espacio de las configuraciones de un robot.* PhD thesis, Universidad de Salamanca, 1998.

[Der88] Jacques Derrida. El filósofo y los arquitectos. *Diagonal*, 73:37–39, 1988.

[Der99] Jacques Derrida. *No escribo sin luz artificial.* Cuatro ediciones, Valladolid, 1999.

[DH55] J. Denavit and R. S. Hartenberg. A kinematic notation for lower-pair mechanisms on matrices. *Journal of Applied Mathematics*, pages 215–221, 1955.

[DHS89] F. Dehne, A. L. Hassenklover, and J. R. Sack. Computing the configuration space for a robot on a mesh-of-processors. *Parallel Computing*, 12(2), 1989.

[Don84] R. Donald. Motion planning with six degrees of freedom. Technical Report AI-TR-791, Artificial Intelligence Loboratory MIT, 1984.

[DS00] Rossen Dimitrov and Anthony Skjellum. Impact of latency on applications'performance. In *Proceedings of MPI Developersánd Users'Conference*, Ithaca, USA, 2000. http://www.mpi-softtech.com.

[Fav86] B. Faverjon. Object level programming of industrial robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1406–1412,, 1986.

[Fos95] I. Foster. *Designing and Building Parallel Programs. Concepts and Tools for Parallel Software Engineering.* Addison-Wesley P.C., 1995. versión HTML: http://www-unix.mcs.anl.gov/dbpp/.

[FW78] S. Fortune and J. Wyllie. Parallelism in random access machines. In *Proceedings of the 10th Annual Symposium on Theory of Computing*, pages 114–118, 1978.

[GJ91] J. S. González and D. I. Jones. An implementation on multiple transputers of a configuration space approach to robot obstacle avoidance. In et al. T. S. Durrani, editor, *Proceedings of the Third International Conference on Applications of Transputers*, volume Aplications of Transputers 3, pages 168–173, Amsterdam, 1991. IOS Press.

[GLB00] Sergi Girona, Jesús Labarta, and Rosa M. Badia. Validation of dimemas communication model for

mpi collective operations. In *Proceedings of EuroPBM/MPI*, Balatonfüred, Hungary, 2000.

[GM89] Q. J. Ge and J. M. McCarthy. Equations for boundaries of joint obstacles for planar robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 418–423, 1989.

[GM90] Q. J. Ge and J. M. McCarthy. An algebraic formulation of configuration-space obstacles for spatial robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 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.

[GZ97] Michael Griebel and Gerhard Zumbusch. Parnass:porting gigabit-lan components to a workstation cluster. In W. Rehm, editor, *Proceedings of the 1st Workshop Cluster-Computing*, pages 101–124, Chemnitz, Germany, 1997.

[HA92] Y. K. Hwang and N. Ahuja. Gross motion planning – a survey. *ACM Computing Surveys*, 24(3):219–291, 1992.

[Hen97] Dominik Henrich. Fast motion planning by parallel processing. a review. *Journal of Intelligent and Robotic Systems*, 20(1):45–69, 1997.

[HT95] Richard Hooper and D. Tesar. Multicriteria inverse kinematics for general serial robots. Technical report, Robotic Research Group. The university of Texas at Austin, 1995.

[Hwa90] Y. K. Hwang. Boundary equations of configurations obstacles for manipulators. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 298–303, 1990.

[HWW98] Dominik Henrich, Christian Wurll, and Heinz Wörn. 6 dof path planning in dynamic environments - a parallel on-line approach. In *Proceedings of IEEE International Conference on Robotics and Automation*, 1998.

[Jen97a] Kurt Jensen. A brief introduction to coloured petri nets. In Brinksma, E., editor, *Lecture Notes in Computer Science: Tools and Algorithms for the Construction and Analysis of Systems. Proceedings of the TACAS'97 Workshop, Enschede, The Netherlands 1997*, volume 1217, pages 201–208. Springer-Verlag, 1997.

[Jen97b] Kurt Jensen. *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use*, volume 1–3 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Berlin, 1997.

[Jen98] Kurt Jensen. An introduction to the practical use of coloured petri nets. *Lecture Notes in Computer Science: Lectures on Petri Nets II: Applications*, 1492, 1998.

[JL95] Jing Fu Jenq and WingÑing Li. Computing the configuration space for a convex robot on hypercube multiprocessors. In *Proceedings of the 7th IEEE Symposium on Parallel and Distributed Processing*, 1995.

[Kav93] Lydia E. Kavraki. Computation of configuration-space obstacles using the fast fourier transform. In *IEEE International Conference on Robotics and Automation*, pages 255–61, Atlanta, GA, USA, 1993.

[Kav95] Lydia E. Kavraki. Computation of configuration space obstacles using the fast fourier transform. *IEEE Tr. on Robotics and Automation*, 11(3):408–413, 1995.

[KGGK94] Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis. *Introduction to Parallel Computing : Design and Analysis of Parallel Algorithms.*

Benjamin Cummings Addison-Wesley Publishing Company, 1994.

[KK91] A. R. Khoogar and J. K. Obstacle avoidance of redundant manipulators using genetic algorithms. In *Proceedings of 1991 Southeastcon*, volume 1, pages 317–320. IEEE Press, 1991.

[KP99] Bernhard Klaassen and Karl L. Paap. Gmd-snake2: A snake-like robot driven by wheels and a method for motion control. In *IEEE International Conference on Robotics and Automation*, 1999.

[KSLO94] Lydia E. Kavraki, Petr Svestka, Jean-Claude Latombe, and Mark Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. Technical Report CS-TR-94-1519, 1994.

[KT96] Chetan Kapoor and D. Tesar. A reusable operational software architecture for advanced robotics. Technical report, Robotic Research Group. The university of Texas at Austin, 1996.

[KWD01] Olaf Kummer, Frank Weinberg, and Michael Duvigneau. *Renew — User Guide*. Theoretical Foundations Group. Distributed Systems Group. Department for Informatics. University of Hamburg, 2001.

[Lat91] J. C. Latombe. *Robot motion planning*. Kluwer Academic Publishers, Boston, MA, 1991.

[LGF86] C. S. Lee, R. C. Gonzalez, and K. S. Fu. *Tutorial on robotics*. IEEE Computer Society, Los Angeles, CA, 1986.

[LOH94] T. Lee, T. Ohm, and S. Hayati. A highly redundant robot system for inspection. In *Conference on Intelligent Robotics in the Field, Factory, Service and Space (CIRFFSS'94)*, pages 142–149, Houston, Texas, 1994.

[LP83] T. Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, 32:108–120, 2 1983.

[LP87] T. Lozano-Pérez. A simple motion-planning algorithm for general robot manipulators. *IEEE Journal of Robotics and Automation*, 3(3):224–238, 1987.

[LPP91] T. Lozano-Pérez and P.O'Donnell. Parallel robot motion planning. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pages 1000–1007, 1991.

[LPW79] Tomás Lozano-Pérez and M. A. Wisley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, October 1979.

[Lum87] V. Lumelsky. Effect of kinematics on motion planning for planar robot arms moving amidst unknown obstacles. *IEEE Transactions on Robotics and Automation*, 3(3):207–223, 1987.

[MF93] A. A. Maciejewski and J. J. Fox. Path planning and the topology of configuration space. *IEEE Tr. on Robotics and Automation*, 9(4), 1993.

[NB91] W.Ñewman and M. Branicky. Real-time configuration space transforms for obstacle avoidance. *The International Journal of Robotics Research*, 6, 10 1991.

[Nil69] N. J. Nilsson. A mobile automaton: An application of artificial intelligence techniques. In *Proceedings of the First International Joint Conference on Artificial Intelligence*, pages 509–520, Washintong D. C., 1969.

[NS83] C.Ñash and S. Sen. *Topology and geometry for physicists*. Academic Press, London, 1983.

[Pet62] Carl Adam Petri. *Kommunikation mit Automaten*. Bonn: Institut f¨ur Instrumentelle Mathematik, Schriften des IIM Nr. 2, 1962.

[Pet66] Carl Adam Petri. Kommunikation mit automaten. *New York: Griffiss Air Force Base, Technical Report RADC-TR-65–377*, 1:Suppl. 1, 1966. English translation.

[Rei98] W. Reisig. *Elements of Distributed Algorithms: Modeling and Analysis with Petri Nets.* Springer-Verlag, 1998.

[RM96] R.G. Roberts and A.A. Maciejewski. A local measure of fault tolerance for kinematically redundant manipulators. *IEEE Transactions on Robotics and Automation*, 12(4), 1996.

[ROS98] Grupo ROS. *Robótica.* Anaya Multimedia, 1998.

[Seg00] CustomSystems High Performance Computing Segment. Considerations in specifying beowulf clusters. Technical report, Compaq, 2000.

[Sil85] Manuel Silva. *Las Redes de Petri: en la Automática y la Informática.* Editorial AC. Madrid, 1985.

[Spi82] M. Spivak. *Cálculo en variedades.* Reverté, Barcelona, 1982.

[SV89] M. W. Spong and M. Vidyasagar. *Robot dynamics and control.* John Wiley and Sons, New York, 1989.

[TBC+02] Roberto Therón, Francisco Javier Blanco, Belén Curto, Vidal Moreno, and Francisco José García. Parallelism and robotics: The perfect marriage. *ACM Crossroads*, 8.3 Parallel computing, Spring 2002. http://www.acm.org/crossroads/xrds8-3/prm.html.

[TBCM01] R. Therón, Francisco Javier Blanco, Belén Curto, and Vidal Moreno. Evaluation of the configuration space of robots upon a distributed computing system. In *Proceedings of the IEEE-IFAC European Control Conference*, Porto, Portugal, september 2001.

[Tea96] The LAM Team. *MPI Primer developing with LAM.* Ohio Supercomputer Center, 1996.

[Tea98] The LAM Team. *Getting Started with LAM/MPI.* University of Notre Dame, Department of Computer Science, *h*ttp://www.lam-mpi.org/, 1998.

[The00] Roberto Therón. Hacia una evaluación rápida del espacio de las configuraciones de estructuras robóticas. Memoria de Grado, Universidad de Salamanca, julio 2000.

[TMCB02] Roberto Therón, Vidal Moreno, Belén Curto, and Francico Javier Blanco. Assessing methods for the evaluation of the configuration space for a planar revoluted manipulators. In In press, editor, *Proceedings of the International Conference on Computational and Mathematical Methods in Science and Engineering*, 2002.

[Wig95] Mark Wigley. *The Architecture of Deconstruction: Derrida's Haunt.* MIT Press, 1995.

Apéndices

Apéndice A

Método de Denavit-Hartenberg: Matrices de transformación

La notación Denavit-Hartenberg (D-H) planteada por primera vez en 1955 por Denavit y Hartenberg, de ahí su nombre, es una notación que permite representar cualquier mecanismo (no solamente robots) en una matriz de cuatro parámetros por cada eslabón de una cadena cinemática.

Estos cuatro parámetros son:

- a_i, la longitud normal entre los ejes de las articulaciones asociadas;
- α_i, el ángulo entre los dos ejes de las articulaciones;
- d_i, la posición relativa de los dos eslabones (distancia entre a_i y a_{i-1});
- θ_i, el ángulo entre a_i y a_{i-1}.

De ellos, los dos primeros, a_i y α_i definen la estructura del eslabón, mientras que los otros dos, d_i y θ_i, determinan la posición del eslabón vecino.

Basándose en la teoría de las transformadas homogéneas, si se establecen sistemas de referencia para cada eslabón y se

relacionan entre sí, se puede describir la posición del efector o del sistema de referencia del eslabón más externo en función del sistema de coordenadas establecido en la base del manipulador. Denavit y Hartenberg desarrollaron esta relación en términos de los cuatro parámetros asociados a cada eslabón.

Para poder hacer esto correctamente, es necesario seguir las siguientes reglas para establecer los sistemas de referencia:

1. El eje z_{i-1} se sitúa a lo largo del eje de movimiento de la $i-$ésimo articulación.
2. El eje x_i es normal al eje z_{i-1} en dirección hacia el eje z_i.
3. El eje y_i se define de forma que forme un triedo, siguiendo la regla de la mano derecha, con los ejes z_i y x_i

Estas reglas contienen la siguiente información implícita:

- Dado que por definición el eje x_i es perpendicular al z_i (regla 3), la regla 2 implica que el eje x_i se sitúe a lo largo de la normal común, a_i, de los ejes z_{i-1} y z_i con la dirección del eje z_{i-1} al eje z_i.
- El origen del sistema de referencia $i-$ésimo se localiza en la intersección de la normal común del eje de articulación i y el eje de articulación $(i+1)$.
- La posición del origen del sistema de referencia 0 se puede elegir en cualquier punto mientras el eje z se sitúe a lo largo del eje de movimiento de la primera articulación.
- El último sistema de coordenadas puede colocarse en cualquier punto del efector mientras el eje x sea normal al eje z de la articulación que une el último sistema de referencia al manipulador.

Si se examinan cuidadosamente estas reglas se observará que la asignación de los sistemas de referencia no es necesariamente única.

Una vez establecidos los sistemas de referencia, se pueden determinar los parámetros $(a_i, \alpha_i, d_i, \theta_i)$ que describen cada par eslabón-articulación. Para ello se siguen las siguientes normas:

1. θ_1 es el ángulo del eje x_{i-1} al eje x_i medido sobre el eje z_{i-1}. Esto se hace utilizando la regla de la mano derecha, ya que x_{i-1} y x_i son perpendiculares a z_{i-1}. La dirección de giro es positiva si el triedo con x_{i-1} y x_i define el eje z_{i-1}. θ_i es la variable de articulación si la articulación es de revolución, si es prismática, es una constante que puede ser o no cero.

2. d_i es la distancia del eje x_i al eje x_{i+1} medida a lo largo del eje z_{i-1}. Si la articulación es prismática, d_i es la variable de la articulación, si es de revolución, es una constante que puede ser o no cero.

3. a_i es la distancia más corta entre los ejes z_{i-1} y z_i. Se mide como la distancia a lo largo de la dirección de x_i a partir de la intersección de z_{i-1} y x_i hasta el origen del sistema de coordenadas $i-$ésimo. Para ejes de articulación que intersectan, a_i es cero. No tiene sentido para articulaciones prismáticas y se pone a cero en tal caso.

4. α_i es el ángulo que se mide desde el eje z_{i-1} al z_i sobre el eje x_i, de nuevo usando la regla de la mano derecha. Para la mayoría de los manipuladores comerciales, α_i son múltiplos de 90^o.

Una vez hecho esto, se define la matriz de Denavit-Hartenberg como un producto de transformaciones básicas:

$$^{i}_{i-1}A = Rot(x_i, \alpha_i)Trans(a_i, 0, 0)Trans(0, 0, d_i)Rot(z_{i-1}, \theta_i)$$

con

$$Rot(z_{i-1}, \theta_i)$$

rotar un ángulo θ_i sobre z_{i-1};

$$Trans(0, 0, d_i)$$

trasladar el origen del sistema de referencia $(i-1)$ una distancia d_i;

$$Trans(a_i, 0, 0)$$

trasladar el origen del sistema de referencia $(i-1)$ una distancia a_i;

$$Rot(x_i, \alpha_i)$$

rotar un ángulo α_i sobre x_i.

Con lo que se llega a la siguiente matriz:

$$^{i}_{i-1}A = \begin{bmatrix} C\theta_i & -C\alpha_i S\theta_i & S\alpha_i S\theta_i & a_i C\theta_i \\ S\theta_1 & C\alpha_i C\alpha_i & -S\alpha_i C\theta_i & a_i S\theta_i \\ 0 & S\alpha_i & C\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

la cual relaciona los puntos definidos en el sistema de referencia i con los del sistema de referencia $(i-1)$.

Finalmente, para un manipulador de n grados de libertad, la expresión para determinar la posición del efector (último sistema de referencia) con respecto al sistema de coordenadas de la base, se tienen la siguiente expresión:

$$^{n}_{0}\mathbf{T} = ^{n}_{0}A = ^{1}_{0}A^{2}_{1}A^{3}_{2}A \cdots ^{n}_{(n-1)}A$$

Apéndice B

Discretización de los espacios W y C

Sea x un punto de W y q una configuración de C, parametrizados, respectivamente, por $x = (x_1, x_2, \cdots, x_n)$ y $q = (q_1, q_2, \cdots, q_m)$. En general, n es 2 ó 3, mientras que m depende del número de grados de libertad del robot considerado. Sobre el intervalo de definición de cada una de estas coordenadas se distribuyen de manera uniforme un número finito de puntos (N). Por ejemplo, si una de las coordenadas de x o de q es una variable lineal $d \in [a, b]$, se distribuyen uniformente N puntos, dando lugar a un vector cuya componente i-ésima sería

$$a + i\frac{b-a}{N}$$

y si la coordenada es angular $\theta \in [-\pi, \pi]$, sería

$$-\pi + i\frac{2\pi}{N}$$

Para simplificar la presentación del método, se supone que todas las coordenadas se discretizan con el mismo número de puntos. Realmente, en la implementación realizada no todas las coordenadas se discretizan con la misma resolución. Esta se debe fijar de forma que la representación de los obstáculos del robot en ambos espacios, W y C, sea correcta.

Se puede utilizar el índice $i \in D$, siendo $D = \{1, \cdots, N\}$, como variable independiente para representar cada uno de los N puntos que se han distribuido sobre el intervalo de definición de cada coordenada de $x \in W$ o $q \in C$. Así, se obtiene una matriz de dimensión $\underbrace{N \times \cdots \times N}_{n}$ para un espacio de trabajo de dimensión n y otra de dimensión $\underbrace{N \times \cdots \times N}_{m}$ para el espacio de las configuraciones.

B.0.5 Discretización de las funciones A y B

Las funciones B, A y A' definidas sobre dominios discretos se denominarán, respectivamente, B^*, A^* y A'^*. Si se utiliza el índice como variable independiente entonces B^*, A^* y A'^* son funciones definidas sobre D, un subconjunto de los naturales, y, por tanto, se pueden representar como matrices. Debido a la propia definición de B, A y A', las correspondientes muestreadas dan como resultado matrices binarias.

Así, la función $B^* : \underbrace{D \times \cdots \times D}_{n} \rightarrow R$ proporciona una matriz binaria que representa a los obstáculos en el espacio de trabajo. Un elemento de esta matriz B^* tomará un valor '1'si existe un obstáculo en la celda que representa dicho elemento, y si no el valor será '0'. De la misma forma, la función $A^* : \underbrace{D \times \cdots \times D}_{m} \times \underbrace{D \times \cdots \times D}_{n} \rightarrow R$ proporciona una matriz binaria que representa al robot en una determinada configuración q en el espacio de trabajo. Mientras que si se utiliza la función $A'^* : C' \times W' \rightarrow R$ se obtiene una matriz binaria que representa al robot en una configuración q' en W'. Cabe recordar que las funciones A' se introducían para simplificar el cálculo de $CB(q)$. Como se podrá comprobar con el siguiente ejemplo se puede reducir además las necesidades de memoria, ya que las matrices con las que se trabaja son de menor tamaño.

B.0.6 Discretización de la función CB

De la misma forma que para las funciones previas, se denomina CB^* a la función CB que toma valores sobre el C-espacio

discreto, y estaría definida por

$$CB^*(q_j) = \sum_{i=0}^{N-1} A^*(q_j, x_i)B^*(x_i) \quad \forall q_j \in C, \quad \forall x_i \in W \quad \text{(B.1)}$$

A partir de ella, la región del C-obstáculo $\mathbf{CB_f}$, que es el subconjunto puntos del C-espacio discreto donde se proyecta el obstáculo $\mathbf{B}$, se define como

$$\mathbf{CB_f} = \{q_j \in C / CB^*(q_j) > 0\} \quad \text{(B.2)}$$

Por tanto, se cumple que un punto del C-espacio discreto es libre si y sólo si $CB^*(q_j) = 0$.

Si se trabaja con los índices el C-espacio se puede representar por una matriz m-dimensional (o por m submatrices $(m-1)$-dimensionales). Así, la función $CB^* : \underbrace{D \times \cdots \times D}_{m} \rightarrow R$ proporciona una matriz que representa a los obstáculos en el espacio de trabajo. Un determinado elemento de esta matriz CB^* tomará un valor no nulo si el robot en la configuración que esa celda representa colisiona con los obstáculos; o un valor '0'si es libre.

El joven Rossum inventó un trabajador con la mínima cantidad de requisitos. Tuvo que simplificarlo. Rechazó cualquier cosa que no contribuyera directamente al progreso del trabajo. Rechazó todo lo que hace al hombre más caro. De hecho, rechazó al hombre y fabricó al Robot. Mi querida señorita Glory, los Robots no son personas. Mecánicamente son más perfectos que nosotros, tienen una inteligencia enormemente desarrollada, pero no tienen alma. ¿Ha visto alguna vez cómo es un Robot por dentro?

——————— Karel Čapek,
Rossum's Universal Robots