UNIVERSIDAD DE MALAGA
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

TESIS DOCTORAL

ARQUITECTURA DE NAVEGACIÓN DISTRIBUIDA PARA AGENTES ROBÓTICOS

AUTOR: Eduardo Javier Pérez Rodríguez
Ingeniero de Telecomunicación

2006

Dña. CRISTINA URDIALES GARCÍA, PROFESORA TITULAR DEL DEPARTAMENTO DE TECNOLOGÍA ELECTRÓNICA DE LA UNIVERSIDAD DE MÁLAGA

CERTIFICO:

Que D. Eduardo Javier Pérez Rodríguez, Ingeniero de Telecomunicación, ha realizado en el Departamento de Tecnología Electrónica de la Universidad de Málaga bajo mi dirección el trabajo de investigación correspondiente a su Tesis Doctoral titulada:

ARQUITECTURA DE NAVEGACIÓN DISTRIBUIDA PARA AGENTES ROBÓTICOS

Revisado el presente trabajo, estimo que puede ser presentado al Tribunal que ha de juzgarlo.

Y para que conste a efectos de lo establecido en el Real Decreto 56/2005 regulador de los estudios de Tercer Ciclo-Doctorado, AUTORIZO la presentación de esta Tesis en la Universidad de Málaga.

Málaga, a 29 de Mayo de 2006

Fdo.: Cristina Urdiales García
Profesora Titular del Departamento de Tecnología Electrónica

Departamento de Tecnología Electrónica
E.T.S.I. Telecomunicación
Universidad de Málaga

TESIS DOCTORAL

ARQUITECTURA DE NAVEGACIÓN DISTRIBUIDA PARA AGENTES ROBÓTICOS

AUTOR: Eduardo Javier Pérez Rodríguez
Ingeniero de Telecomunicación

DIRECTOR: Cristina Urdiales García
Dra. Ingeniera de Telecomunicación

A mi familia,
refugio inquebrantable
en mi viaje por la vida,

y a ***Rosa****,*
la mejor persona
que jamás he conocido

¿Por qué demonios no elegí la pastilla azul?

Cifra - ***Matrix***

Agradecimientos

Han sido numerosas las personas que con su apoyo han contribuido directa o indirectamente en la realización de esta Tesis. Desde estas líneas quisiera aprovechar la ocasión para manifestarles mi gratitud a todas ellas.

En primer lugar a mi directora de Tesis y amiga Cristina Urdiales, la persona que en mayor medida ha contribuido en este trabajo. Sin tu ayuda y esfuerzo este camino habría sido infinitamente más complejo. Gracias por todo, Cris.

Un especial agradecimiento a Francisco Sandoval, la persona que me ha ayudado a crecer profesionalmente a lo largo de los últimos años. También a la Comisión Interministerial de Ciencia y Tecnología (*CICYT*), al Ministerio de Ciencia y Tecnología (*MCYT*) y al Ministerio de Educación y Ciencia (*MEC*), que han posibilitado el desarrollo de este trabajo mediante los proyectos TIC98-0562, TIC2001-1758 y TIN2004-07741-C02-01.

También quisiera agradecerle a Javier Vázquez la generosa y paciente ayuda que siempre me ha prestado en la utilización del sistema *CBR*.

A José Manuel Cano y a Alberto Poncela, por el apoyo diario que me han proporcionado durante estos años en los que hemos compartido una indescriptible amistad.

A mis padres y hermanos, que con su cariño han ido conquistando el inmenso lugar que ocupan en mi vida. También a Rodolfo y a Milagros, por el inapreciable afecto que me han entregado siempre.

Y por supuesto, quizás en último lugar en estos agradecimientos pero indudablemente en primer lugar en mi corazón, gracias a ti, ***Rosa***, por devolverme la alegría necesaria para seguir afrontando la vida con ilusión y esperanza, por permanecer siempre a mi lado incluso en los momentos más difíciles, y por ser la persona más bondadosa que jamás haya conocido. *Al final cayó mi estrella.*

A todos vosotros, y a todos aquellos que por un inexcusable descuido de mi memoria no estáis aquí pero sí en mi corazón,

Gracias

Resumen

Las últimas décadas han contemplado la silenciosa pero imparable introducción de nuevos robots en nuestras vidas, convirtiéndose en una realidad no sólo evidente sino también imprescindible en la sociedad actual. Todos estos robots que nos acompañan día a día están asistiendo al ser humano en la realización de tareas complejas, repetitivas, desagradables y/o peligrosas, ya sea en entornos hostiles o de difícil acceso.

Para el desarrollo de tareas cada vez más complejas y avanzadas, suele ser necesario que estos robots estén dotados de la capacidad de navegar en entornos dinámicos posiblemente no estructurados. Este problema es bastante complejo de resolver y, aunque existen numerosos esquemas de navegación para agentes autónomos móviles, la evidente falta de estandarización existente dificulta enormemente su adaptación a otros sistemas distintos de aquellos para los que fueron concebidos.

En esta Tesis se ha desarrollado una infraestructura básica que permite incorporar la capacidad de navegación en entornos dinámicos no estructurados a un agente autónomo móvil genérico, permitiendo su posterior adaptación a distintos agentes físicos y una fácil ampliación de sus capacidades. Para ello, se ha desarrollado y empleado una nueva arquitectura de control denominada ***DLA*** (***D****istributed and* ***L****ayered* ***A****rchitecture*), que permite combinar algoritmos cooperativos mediante la interacción de procesos libremente distribuidos. Esta arquitectura ha sido concebida con el objetivo prioritario de la transparencia en su uso, concretada en la sencillez y portabilidad de la misma. Así mismo, presenta la posibilidad de distribuir los procesos en varias máquinas dependiendo de la carga de la red y de los procesadores implicados, controlándose así la velocidad de respuesta de los módulos.

La infraestructura básica de navegación desarrollada utiliza una estructura híbrida, la cual combina adecuadamente y de forma asíncrona rápidos comportamientos reactivos con procesos deliberados de planificación. De esta forma se puede responder rápidamente ante los estímulos captados del entorno al tiempo que se efectúa una planificación de la operación del sistema para el desarrollo de tareas complejas.

Los comportamientos reactivos del sistema se han desarrollado mediante un esquema basado en aprendizaje capaz de adquirir restricciones cinemáticas durante el proceso de entrenamiento. De esta forma se facilita la adaptación de los comportamientos reactivos a distintos agentes con diferentes características dinámicas sin necesidad de un estudio analítico específico. Para su implementación se ha utilizado el paradigma del razonamiento basado en casos, que permite un aprendizaje rápido tanto por entrenamiento como por experiencia propia del agente.

Los procesos deliberados incorporados en el sistema permiten aumentar la eficiencia y las capacidades de navegación en la operación del sistema. Se han desarrollado dos niveles distintos de planificación: un planificador de caminos que permite calcular un camino libre de obstáculos para alcanzar el destino final, y un planificador de rutas que permite calcular una ruta formada por las regiones a atravesar hasta alcanzar el destino final. Para llevar a cabo estos procesos de planificación se ha generado una representación métrica de tipo probabilístico a partir de la información sensorial captada del entorno, y una representación topológica a partir de la

representación métrica mediante una estructura jerárquica piramidal. Ambas permanecen constantemente relacionadas mediante una estructura de enlaces, lo que permite propagar los planes con rapidez de un nivel a otro.

La arquitectura implementada ha demostrado ser muy robusta, operando correctamente en entornos reales dinámicos no estructurados para distintos agentes físicos después de un corto entrenamiento.

Abstract

Robots have steadily stepped into our lives through the last decades, becoming not only an evident but also an essential reality in current society. All these robots working for us day by day are helping humans with complex, repetitive, unpleasant and/or dangerous tasks, both in hostile or harsh environments.

In order to develop progressively more complex and advanced tasks, it is usually necessary that these robots are able to navigate in dynamic and potentially not structured environments. This problem is quite complex to solve, and although many navigation schemes for autonomous mobile agents have been proposed, obvious lack of standardization makes their adaptation to systems different from those for which they were conceived enormously difficult.

This Thesis presents a basic infrastructure to support navigation in not structured dynamic environments for any general autonomous mobile agent. It allows intuitive adaptation to physically different agents and simple expansion of their capacities. This infrastructure is supported by a new control architecture named ***DLA*** (***D****istributed and* ***L****ayered* ***A****rchitecture*), that allows combination of cooperative algorithms through the interaction of freely distributed processes. This architecture has been conceived keeping transparency in mind, by means of a high simplicity and portability in its use. Also, it presents the possibility of distributing processes among different machines depending on the network and working processors loads. Thus, the response speed of the modules can be controlled.

The developed basic navigation infrastructure relies on a hybrid structure, which successfully connects fast reactive behaviours and deliberate planning processes in an asynchronous way. This strategy allows fast reaction to environment stimuli and also planning for the development of complex tasks at the same time.

Reactive behaviours have been developed by means of a learning scheme that supports acquisition of kinematics restrictions through a training stage. Thus, adaptation of reactive behaviours to agents with different dynamic characteristics can be achieved without a specific analytical study. Its implementation is based on the case-based reasoning paradigm, which allows fast learning both through supervised training and own experience.

Deliberate processes increase the efficiency and navigation capacities of the system. Two different levels of planning have been developed: a path planner that computes a trajectory free of obstacles to reach the target, and a route planner that provides a succession of regions to reach the target. These planning processes are supported by a metric probabilistic representation built from environment stimuli, and a topologic representation extracted from the metric one by means of a hierarchical pyramidal structure. Both representations are constantly related through a set of links, so that plans can be quickly propagated from one level to the other.

The developed architecture has proven to be very robust. It works correctly in unstructured dynamic environments with different physic agents after a short training.

Índice General

Índice de Figuras

Índice de Tablas

Lista de Símbolos

2β	Anchura del lóbulo principal del sensor sonar en la generación del mapa métrico
A_{min}	Número mínimo de celdas para formar una región en el mapa topológico
C_{dist}	Coeficiente del factor de distancia en la función de evaluación de la base de casos
C_{free}	Nivel de probabilidad para las celdas libres en la generación del mapa topológico
C_{not}	Nivel de probabilidad para las celdas no exploradas en la generación del mapa topológico
C_{occ}	Nivel de probabilidad para las celdas ocupadas en la generación del mapa topológico
C_{sec}	Coeficiente del factor de seguridad en la función de evaluación de la base de casos
C_{soft}	Coeficiente del factor de suavidad en la función de evaluación de la base de casos
$Dist\_Cluster$	Distancia máxima entre casos para formar una clase en la optimización de la base de casos
$Dist\_Max$	Distancia máxima entre regiones para formar una región en el mapa topológico
G_{attrac}	Coeficiente de ponderación de la fuerza atractiva hacia el destino
G_{rep}	Coeficiente de ponderación de la fuerza repulsiva de los obstáculos del entorno
K_{dist}	Coeficiente de ponderación del factor de distancia en la función de evaluación de la base de casos
K_{sec}	Coeficiente de ponderación del factor de seguridad en la función de evaluación de la base de casos
K_{soft}	Coeficiente de ponderación del factor de suavidad en la función de evaluación de la base de casos
Num_Col	Número de columnas del mapa métrico
Num_Row	Número de filas del mapa métrico
$Obst\_Grow$	Celdas utilizadas para el crecimiento de obstáculos
P_{free}	Reducción de probabilidad para las celdas libres en la generación del mapa métrico
P_{obst}	Umbral de probabilidad para los obstáculos en el proceso de planificación de caminos
P_{occ}	Incremento de probabilidad para las celdas ocupadas en la generación del mapa métrico
$Size\_Cell$	Tamaño de la celda del mapa métrico
U_{adap}	Distancia para adaptar el caso devuelto en el sistema CBR
U_{disp}	Distancia para la captura de nuevos casos durante el entrenamiento del sistema CBR

U_{free}	Umbral de probabilidad para las celdas libres en la generación del mapa topológico
U_{occ}	Umbral de probabilidad para las celdas ocupadas en la generación del mapa topológico
U_{path}	Celdas modificadas en el mapa métrico para relanzar la planificación de caminos
U_{prox}	Distancia para activar la evitación de obstáculos
U_{sec}	Distancia de seguridad para detener al agente
U_{topo}	Celdas modificadas en el mapa métrico para relanzar la planificación de rutas

Lista de Acrónimos

ACE	*Adaptive Communication Environment*
AP	*Adversarial Planner*
BERRA	*BEhavior-based Robot Research Architecture*
CBR	*Case Based Reasoning*
CORBA	*Common Object Request Broker Architecture*
DLA	*Distributed and Layered Architecture*
EUROP	*EUropean RObotics Platform*
GPL	*General Public License*
GPS	*Global Positioning System*
HRI	*Human Robot Interface*
JPL	*Jet Propulsion Laboratory*
KEMLg	*Knowledge Engineering & Machine Learning Group*
LAAS	*Laboratoire d'Analyse et d'Architecture des Systèmes*
MIT	*Massachusetts Institute of Technology*
NASA	*National Aeronautics and Space Administration*
PM	*Process Manager*
PRS	*Procedural Reasoning System*
RAP	*Reactive Action Package*
SPA	*Sense-Plan-Act*
SRI	*Stanford Research Institute*
TCA	*Task Control Architecture*
TES	*Task Execution Supervisor*
VLSI	*Very Large Scale Integration*

Capítulo 1

Introducción

En este capítulo se describen los objetivos perseguidos con la realización de la presente Tesis, destinada al desarrollo de un sistema modular y flexible que permita la implementación de agentes autónomos móviles con capacidad de navegación en entornos dinámicos no estructurados.

Este capítulo se ha organizado como se indica a continuación. En el apartado 1 se realiza un recorrido histórico del nacimiento y evolución de los agentes autónomos móviles hasta nuestros días. En el apartado 2 se analiza el concepto de navegación y se realiza una clasificación de los principales esquemas existentes desde el punto de vista de los organismos biológicos. En el apartado 3 se presenta la motivación y los objetivos perseguidos con la realización de la Tesis. Por último, en el apartado 4 se describe la estructura y la organización de la Tesis.

1 Robótica: origen, evolución y tendencias

Las últimas décadas han sido testigo de la silenciosa pero imparable introducción de innumerables robots en nuestras vidas, fruto de la vertiginosa evolución que está sufriendo la Robótica como rama de la ciencia. Todos estos robots han asistido activamente al ser humano en la realización de tareas complejas, repetitivas, desagradables y peligrosas, en todo tipo de entornos hostiles o de difícil acceso como volcanes, simas abisales, zonas catastróficas, entornos radiactivos y cada vez más lejanos planetas.

Los continuos avances que se están produciendo en la Robótica actual está originando una eclosión de robots cada vez más versátiles, más eficientes, más robustos y por qué no, más "humanos". Estos robots están extendiendo su campo de aplicación a parcelas que hasta hace poco estaban reservadas para el campo de la ciencia ficción, introduciéndose en nuestros hogares, en nuestros hospitales, en nuestras oficinas, apoyando a la educación de nuestros hijos, cuidando de nuestros ancianos, velando por nuestra seguridad y facilitando nuestros quehaceres diarios.

Lenta, muy lentamente, estos robots están comenzando a convertirse en una realidad no sólo evidente sino también imprescindible en nuestras vidas, siguiendo el mismo recorrido de otros

avances tecnológicos de nuestra era como la telefonía móvil, la conectividad a *Internet* o la localización por *GPS* (*Global Positioning System*), sin los cuales no podría concebirse la sociedad actual que conocemos.

Pero, ¿qué puede ofrecernos la Robótica actual en nuestra vida cotidiana? Para responder a esta pregunta se hace necesario realizar una mirada retrospectiva al origen y evolución de la Robótica como ciencia.

1.1 Origen de la Robótica

El sueño de desarrollar seres artificiales dotados de inteligencia capaces de tomar decisiones por sí mismos es bastante antiguo dentro de la humanidad. Ya en la antigua Grecia el propio Aristóteles estableció la conveniencia de disponer de esclavos mecánicos capaces de obedecer órdenes: "si cada herramienta, cuando se le ordenase, o incluso bajo sus propias decisiones, hiciera el trabajo que le correspondiese ... entonces no habría necesidad ni de aprendices para los maestros de oficio ni de esclavos para los señores".

Han sido muchos los pasos que el ser humano ha dado a lo largo de la historia con el fin de desarrollar estos seres artificiales. Alrededor del año 400 A.C. el filósofo y matemático griego Archytas de Tarento construyó una paloma capaz de volar agitando sus alas. Posteriormente, sobre el año 200 A.C. el inventor y físico griego Ctesibus de Alejandría construyó un reloj de agua con figuras móviles, sustituyendo los incómodos relojes de arena utilizados hasta entonces. Ya en las primeras décadas de nuestra era, Herón de Alejandría construyó sus famosos autómatas, una serie de aparatos mecánicos dotados de movimiento.

Durante el Renacimiento los diseños e inventos de Leonardo da Vinci crearon figuras mecánicas que se movían imitando el comportamiento de los seres humanos. Basados en estos diseños Hans Bullmann construyó a principios del siglo XVI un androide capaz de tocar instrumentos musicales. En 1533 Johann Müller creó una mosca de hierro y un águila capaces de volar, hazaña que repetiría John Dee en 1543, esta vez con un escarabajo de madera.

La Revolución Industrial supondría un avance mecánico que se reflejó claramente en el desarrollo de máquinas dotadas de movimiento, como el teatro mecánico de Heilbrunn de 1725, que incorporó 119 figuras animadas que representaban una obra sobre la vida en una aldea. Jacques Vaucanson crearía en 1737 un pato mecánico que simulaba un animal real, así como un músico capaz de tocar 11 melodías diferentes. En 1770 distintos inventores reunidos por Henri-Louis Jaquet-Droz construyeron diversos autómatas capaces de escribir, tocar música y pintar cuadros.

Estos primeros intentos por desarrollar seres artificiales se centraron en crear máquinas que imitaran el comportamiento de diferentes seres vivos. Sin embargo, en 1801 Joseph Jacquard construyó un telar programable controlado con tarjetas perforadas, creando una máquina capacitada para desarrollar una tarea más allá de la mera imitación de ciertos comportamientos. Además, el telar programable de Jacquard introduce por primera vez la posibilidad de modificar la tarea a realizar por estas máquinas autónomas. Esta programabilidad utilizando tarjetas

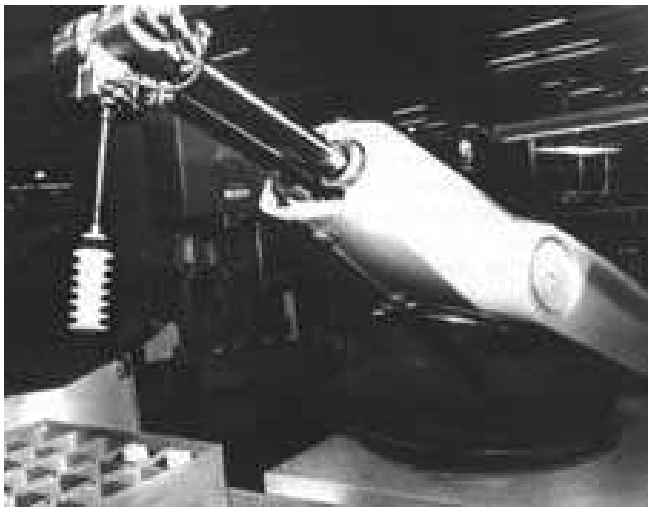

Figura 1.1: Robot industrial *Unimate*.

perforadas sería posteriormente utilizada por el motor analítico de Charles Babbage en 1822, la primera máquina programable con capacidad de cómputo. Aunque el motor analítico de Babbage nunca llegó a ser plenamente operativo, se considera como el origen de los modernos ordenadores.

No obstante, el término **robot** fue utilizado por primera vez en 1921 por el escritor checo Karel Capek en su obra "*Rossum´s Universal Robot*", donde máquinas con apariencia humana suplen a trabajadores humanos. El origen de dicho término proviene de la palabra eslava *robota*, que se refiere al trabajo tedioso que hay que realizar obligatoriamente. Esta misma temática la repite en 1927 Fritz Lang en su obra "*Metropolis*". No obstante, la **Robótica** como ciencia se recoge por primera vez en 1942 en el relato "*Runaround*" de Isaac Asimov, autor de numerosos relatos de ciencia ficción relacionados con la Robótica y marcados por sus tres famosas leyes, cuya obra más conocida es quizás "*I, Robot*", editada en 1950.

Los primeros pasos de esta nueva ciencia conocida como Robótica estuvieron claramente encaminados al campo de la producción industrial y de la operación supervisada, pues no se disponía de los conocimientos necesarios para dotar de inteligencia a esta primera generación de robots que el ser humano estaba intentando desarrollar.

En 1946 George Devol patentó un dispositivo programable que permitía reproducir secuencias de control sobre una determinada máquina. Este importante descubrimiento permitió que en 1954 el propio George Devol diseñara el primer robot manipulador programable, sembrando así el germen de *Unimation*, la primera empresa dedicada al desarrollo de robots industriales. En 1960 comenzó el diseño del primer robot industrial comercial, el *Unimate* de George Devol y Joseph Engelberger (figura 1.1), que en 1962 sería incorporado por *General Motors* en sus cadenas de montaje.

Los avances producidos desde entonces en el campo de la producción industrial estuvieron encaminados a perfeccionar los sucesivos robots industriales. Victor Scheinman diseñó en 1970 un

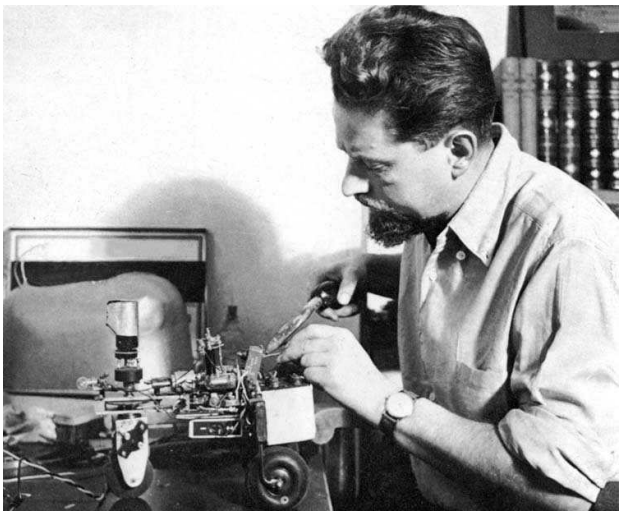

Figura 1.2: Robot *ELSIE*.

brazo articulado utilizando novedosos conceptos cinemáticos que se convirtieron en un estándar utilizado incluso hoy en día. En 1974 el propio Victor Scheinman comercializó una versión industrial de su brazo articulado, el cual estaba controlado por ordenador y permitía una gran configurabilidad en su operación. Posteriormente, Takeo Kanade construyó en 1981 un nuevo brazo articulado que incorporaba directamente en su estructura los distintos motores necesarios para su funcionamiento, aumentando la velocidad y precisión en su operación.

Fuera del campo de la producción industrial se comenzaron a dar algunos tímidos pasos en la construcción de robots, aunque rápidamente se evidenciaron las limitaciones existentes en el desarrollo de los mismos. En 1950 Grey Walter construyó *ELSIE* (*Electro-Light-Sensitive Internal-External*), el cual se puede considerar como el primer agente autónomo móvil de la historia. *ELSIE* se limitaba a seguir una fuente de luz utilizando un sistema mecánico realimentado (figura 1.2), aunque debido a su extrema sencillez no poseía ningún tipo de inteligencia.

Si bien es cierto que *ELSIE* reunió por primera vez autonomía y movilidad en un robot, su sencillo comportamiento evidenció que para desarrollar robots inteligentes era necesario disponer de un sistema de control eficiente, capaz de desarrollar tareas diferentes y de tomar decisiones en función de la información procedente del mundo exterior.

1.2 La influencia de la Inteligencia Artificial

Por aquella época comenzó a experimentar un extraordinario auge una nueva disciplina conocida como **Inteligencia Artificial**. Su inicio se suele datar sobre 1943, cuando Warren McCulloch y Walter Pitts publicaron un artículo combinando novedosas ideas sobre la computación, la lógica y el sistema nervioso, en lo que parecía una teoría revolucionaria en los campos de la psicología, la filosofía y la tecnología [MP43]. Esta nueva teoría presentaba una metodología para calcular proposiciones lógicas mediante simples redes neuronales, y en 1947 McCulloch y Pitts publicaron otro artículo donde describían el cerebro humano como un sistema de procesamiento paralelo

que cambia siguiendo ecuaciones estadísticas [MP47].

Los artículos de McCulloch y Pitts presentaban en definitiva un modelo neuronal del cerebro humano, así como de la representación simbólica del mismo. Puesto que el ser humano es la forma de vida orgánica más inteligente que se conoce, la posible comprensión y modelado del funcionamiento del cerebro humano incentivó la idea de imitar su comportamiento mediante dispositivos electrónicos, con el objetivo de desarrollar sistemas que manifestasen un nivel de inteligencia similar al mostrado por los seres humanos. La Inteligencia Artificial, pues, trata de emular el nivel de la inteligencia humana mediante sistemas electrónicos, usando como elementos cruciales la representación y la abstracción del conocimiento, cuya manipulación adecuada es la base del razonamiento.

Sobre los años 50 comenzó el trabajo de simulación mediante computadoras de los procesos mentales, centrado principalmente en dos vertientes [Bod95]: la **simbólica** basada en la aplicación de reglas lógicas, y la **conexionista** basada en el estudio de las redes neuronales, estructuras formadas por unidades de computación simples interconectadas entre sí. En 1950 Alan Turing presentó un test para verificar el nivel de inteligencia alcanzado por una máquina, mientras que Alan Newell y Herbert Simon construyeron en 1956 el primer sistema experto capaz de ayudar en la resolución de problemas matemáticos complejos.

Estos tempranos éxitos crearon unas grandes expectativas de reproducir la inteligencia humana, reforzando el interés despertado en la investigación en Inteligencia Artificial. En 1959 John McCarthy y Marvin Minsky crearon el Laboratorio de Inteligencia Artificial en el *MIT* (*Massachusetts Institute of Technology*). Posteriormente, John McCarthy abandonó el *MIT* y fundó en 1963 el Laboratorio de Inteligencia Artificial en la *Universidad de Stanford*. En 1964 se abrieron los laboratorios de investigación en Inteligencia Artificial en el *SRI* (*Stanford Research Institute*) y en la *Universidad de Edimburgo*.

Como fruto del esfuerzo investigador llevado a cabo se empezaron a cosechar algunos logros destacados. En 1966 Joseph Weizenbaum creó *ELIZA* en el *MIT*, un pequeño programa de ordenador capaz de establecer una pequeña conversación con interlocutores humanos, mientras que Richard Greenblatt desarrolló en 1967 *MacHack*, un programa capaz de jugar al ajedrez.

1.3 Evolución de la Robótica

Las expectativas que la Inteligencia Artificial despertó en los comienzos de la Robótica fueron realmente prometedoras, pues la posible imitación del razonamiento humano mediante sistemas electrónicos permitiría la implementación de los sistemas de control inteligentes que necesitaban los robots para alcanzar la tan deseada autonomía y tomar decisiones por sí mismos.

De hecho, la ciencia ficción captó muy rápidamente las posibles implicaciones resultantes de la fusión entre la Robótica y la Inteligencia Artificial, materializando el sueño de desarrollar seres artificiales inteligentes capaces de operar al mismo nivel que el propio ser humano. En 1968 Stanley Kubrick creó a HAL 9000 en "*2001: Una Odisea en el Espacio*", un potente ordenador

central dotado de inteligencia que decide prescindir del factor humano en su operación. Posteriormente "*La Guerra de las Galaxias*" de George Lucas dió vida en 1977 a *R2-D2* y *C-3PO*, quizás los androides más conocidos de la historia. En 1982 Ridley Scott creó sus famosos *Replicantes* en "*Blade Runner*", androides tan inteligentes como los seres humanos pero superiores en fuerza, que tras rebelarse fueron declarados proscritos y perseguidos para ser exterminados.

La aparición de esta nueva Robótica en la ciencia ficción agudizó el interés por desarrollar seres inteligentes capaces de sustituir al ser humano en determinadas tareas, planteando una nueva visión que iba mucho más allá de la utilización de robots en la producción industrial. Surge así un nuevo concepto de robot, y con él una rápida expansión de la Robótica hacia distintos campos de aplicación:

- **Agentes autónomos móviles:** (figura 1.3)

  El desarrollo de agentes autónomos móviles con comportamientos inteligentes ha sido uno de los sueños más perseguidos por la Robótica. No es de extrañar, pues, que en los comienzos de esta nueva Robótica se destinaran numerosos esfuerzos al desarrollo de robots móviles dotados de inteligencia. La visión predominante en aquella época era utilizar la Inteligencia Artificial para implementar sistemas de control que emularan el razonamiento humano e incorporar dichos sistemas de control en los robots móviles.

  Con esta filosofía, en 1968 se implementa *Shakey* en el *SRI*, provisto de sensores de contacto, un sistema de triangulación por láser y visión artificial. *Shakey* fue el primer agente autónomo móvil de la historia con capacidad de razonar acerca de sus propias acciones, mostrando una funcionalidad mucho más avanzada que la de *ELSIE*.

  Debido al creciente interés que estaba despertando esta nueva Robótica apoyada por la Inteligencia Artificial, en 1979 se funda el Instituto de Robótica en la *Universidad de Carnegie Mellon* , y en 1986 *Honda* comienza su programa de investigación en Robótica, bajo la premisa de que "un robot debería coexistir y cooperar con los seres humanos, haciendo lo que las personas no pueden hacer y proporcionando una nueva dimensión de la movilidad que finalmente revierta en un claro beneficio para la sociedad".

  En 1989 Rodney Brooks construye *Genghis* en el *MIT*, un robot hexápodo capaz de caminar y de evitar la colisión con los obstáculos de su entorno, manifestando unas capacidades muy superiores a los robots implementados hasta entonces. En 1993 se construye *Dante* en la *Universidad de Carnegie Mellon*, un robot con 8 patas que desciende al volcán *Erebrus* en la Antártida, aunque la misión finalmente fracasa al no ser *Dante* capaz de operar en el difícil terreno de dicho volcán. En 1994 se construye *Dante II*, un robot más robusto que su predecesor que consigue descender satisfactoriamente al volcán *Spurr* en la Antártida, demostrando que estos nuevos robots son capaces de realizar tareas muy útiles para el ser humano.

  En 1996, y tras 10 años de investigación, *Honda* presenta su robot humanoide *P3*, un robot bípedo con unas capacidades de movilidad y flexibilidad desconocidas hasta el momento. Este robot es capaz de subir y bajar escaleras, de caminar lateralmente y de mantener su estabilidad en superficies no uniformes o incluso sobre una única pata. En el año 2000

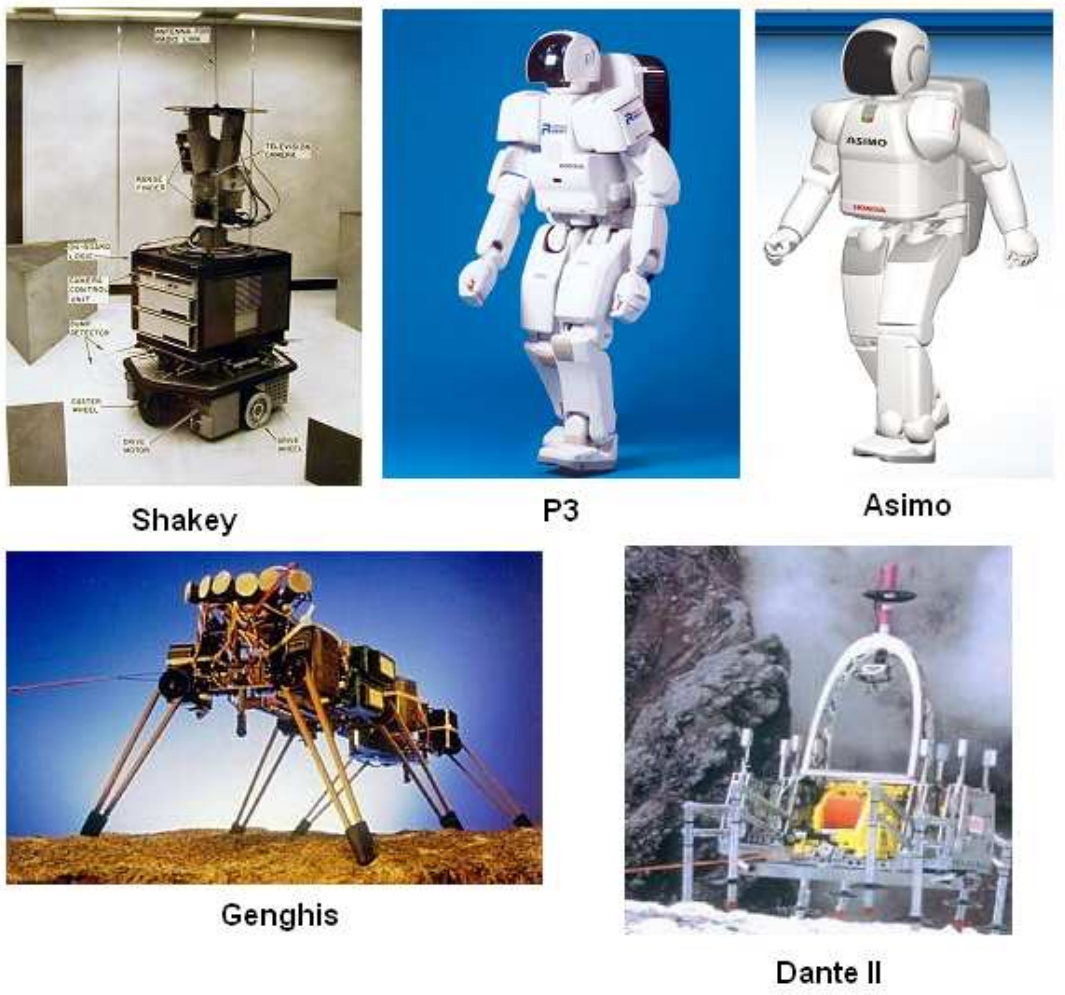

Figura 1.3: Agentes autónomos móviles.

Honda presenta su nuevo prototipo de robot humanoide *Asimo*, mejorando y perfeccionando la movilidad conseguida con el robot *P3*. Los robots de *Honda* constituyen un hito muy importante en el desarrollo de la Robótica, pues demuestran que se puede resolver el problema de la estabilidad en la implementación de androides semejantes en aspecto a los seres humanos, acercándonos al sueño de disponer de seres artificiales capaces de interactuar con las personas en su ambiente natural.

- **Exploración planetaria:** (figura 1.4)

  Sin ningún lugar a dudas, uno de los campos de aplicación más importantes para el uso de agentes autónomos móviles inteligentes se encuentra en las misiones de exploración planetaria, donde no es posible enviar seres humanos. Como fruto de los primeros éxitos cosechados por robots como *Shakey*, en los años 70 la *NASA* (*National Aeronautics and Space Administration*) inicia un programa de cooperación con el *JPL* (*Jet Propulsion Laboratory*) para desarrollar plataformas capaces de explorar terrenos hostiles.

  En 1976 la *NASA* envía a Marte las sondas *Viking 1* y *Viking 2*, con el objetivo de explorar y recoger datos de la superficie del planeta. Entre el equipamiento utilizado por

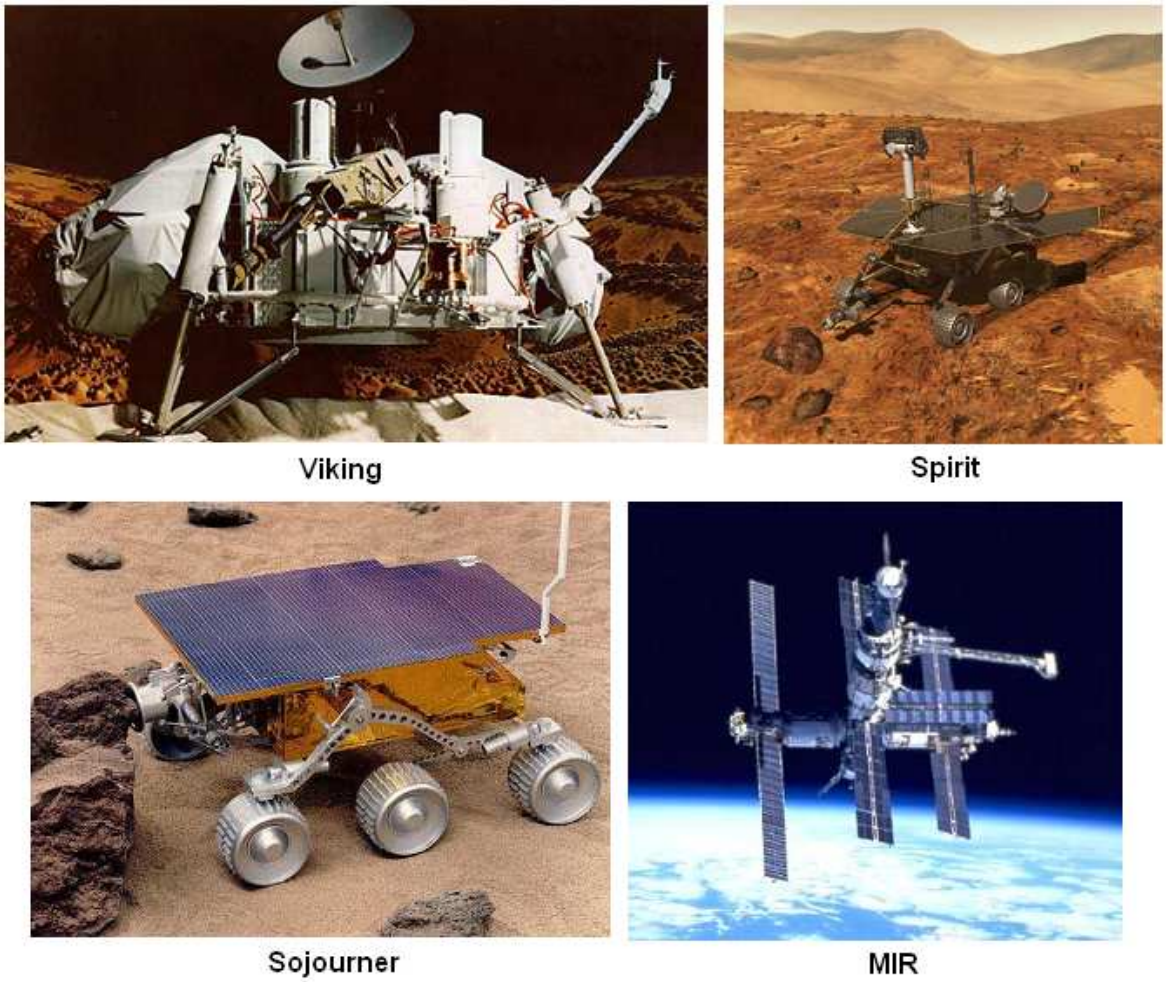

Figura 1.4: Exploración planetaria.

estas sondas se encuentra un brazo robótico para recoger muestras. En 1997 comienza a construirse la estación espacial internacional *MIR*, a la que posteriormente se le añadirá un brazo robótico para realizar tareas de mantenimiento y reparación.

Finalmente, en 1997 la *NASA* realiza con éxito la misión *Pathfinder*, que consigue enviar el robot *Sojourner* a Marte. Dicho robot es capaz de navegar durante meses por suelo marciano, recogiendo y enviando datos de la superficie del planeta. Sin duda alguna, esta misión constituye uno de los hitos más importantes de la Robótica. Esta hazaña la vuelve a repetir la *NASA* en 2003 con la misión *Mars Exploration Rover*, colocando esta vez los robots *Spirit* y *Opportunity* en la superficie de Marte.

- **Mascotas domésticas:** (figura 1.5)

  El desarrollo de la Robótica también ha encontrado su campo de aplicación en las mascotas domésticas, pequeños robots capaces de mostrar comportamientos más o menos inteligentes. La evolución de este tipo de robots ha sido realmente significativa en los últimos tiempos.

  En 1998 *Tiger Electronics* introduce *Furby* en el mercado doméstico, la primera mascota

Figura 1.5: Mascotas domésticas.

electrónica dotada de numerosos sensores que le permiten reaccionar a los estímulos del entorno y comunicarse con los seres humanos. Ese mismo año *LEGO* presenta la serie *Mindstorms*, dedicada a la implementación de pequeños robots dotados de diversos sensores y de cierta capacidad de procesamiento mediante piezas básicas de *LEGO*. En 1999 *Sony* introduce *Aibo*, una mascota doméstica con forma de perro. Las capacidades mostradas por *Aibo* son realmente avanzadas, hasta el punto de que actualmente se utiliza no sólo como mascota doméstica sino también como robot básico en investigación. Entre otras características, *Aibo* es capaz de reconocer objetos, caras y sonidos.

1.4 La Robótica actual

Es cierto que aun no estamos en situación de desarrollar robots inteligentes que puedan razonar y sustituir completamente al ser humano en la realización de tareas complejas, pero es inevitable reconocer la creciente influencia que la Robótica tiene sobre nuestra sociedad actual. De hecho, la definición actual de robot según la *Real Academia Española* es: "máquina o ingenio electrónico

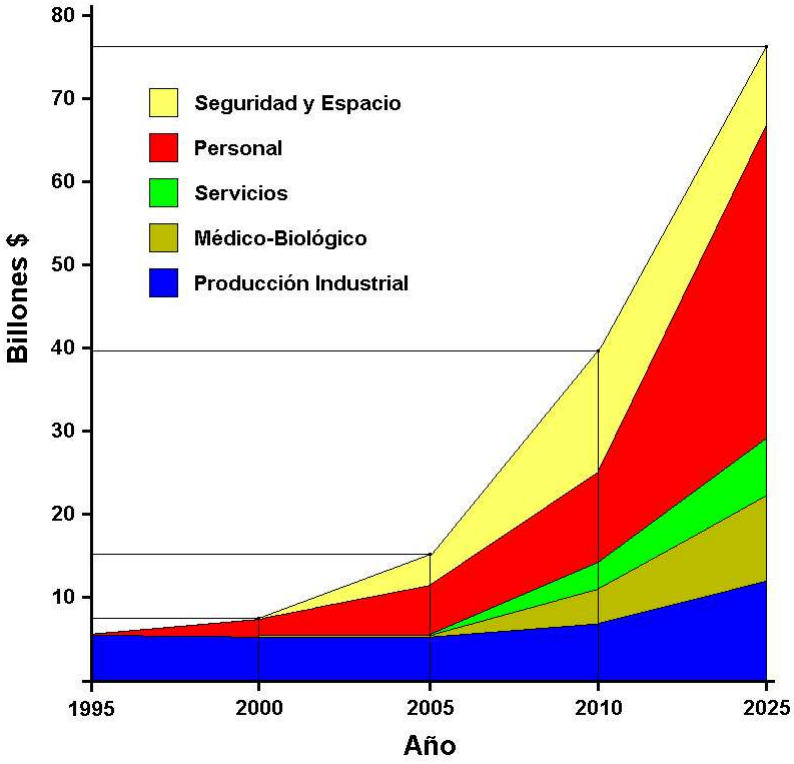

Figura 1.6: Estimación del mercado mundial de Robótica [Fuente: *Japan Robotics Association*].

programable, capaz de manipular objetos y realizar operaciones antes reservadas solo a las personas".

Se puede afirmar que actualmente la mayor aplicación de la Robótica se centra en la producción industrial en entornos estructurados, aunque una nueva generación de robots capaz de interactuar con los seres humanos en su entorno natural está comenzando a emerger. Esta nueva situación está posibilitando la aparición de nuevas aplicaciones y la apertura de potenciales mercados económicos, y nos está acercando cada vez más a la meta final de desarrollar seres artificiales inteligentes que puedan cooperar plenamente con el ser humano.

1.4.1 Situación del mercado

Como muestra la figura 1.6, un simple vistazo a las previsiones de crecimiento económico del mercado mundial de Robótica en los próximos años pone de manifiesto el espectacular avance que se espera en las próximas décadas.

De hecho, la Robótica se ha convertido hoy en día en un sector estratégico clave para aumentar la competitividad y productividad de las industrias y para mejorar la calidad de vida de los ciudadanos, por lo que las distintas potencias se están posicionando para no quedar relegadas en este importante sector. La figura 1.7 muestra el estado actual de desarrollo de la Robótica en las potencias más importantes, distinguiendo los sectores más importantes presentes en el

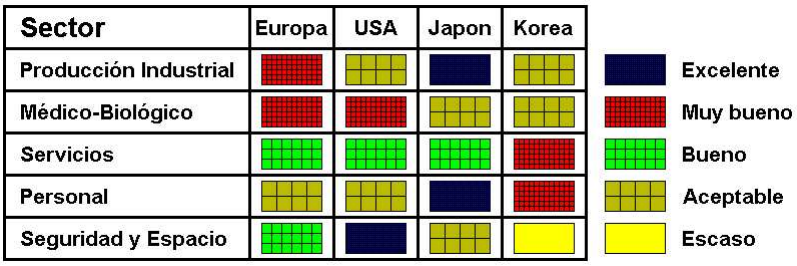

Figura 1.7: Comparativa del desarrollo actual de la Robótica [Fuente: *World Technology Evaluation Center*].

mercado.

Como se puede apreciar existe bastante igualdad entre las distintas potencias, repartiéndose entre todas ellas los distintos sectores del mercado actual de Robótica.

Dentro de este panorama, el esfuerzo que Europa está realizando por fomentar la investigación en Robótica es muy importante. Ya en el 5º Programa Marco de la Unión Europea el presupuesto destinado a investigación en Robótica era de 125 millones de euros, mientras que el en 6º Programa Marco dicho presupuesto prácticamente se ha doblado alcanzando los 200 millones de euros. En esta línea, el 7 de Octubre de 2005 se presentó la plataforma tecnológica *EUROP* (*EUropean RObotics Platform*) [1], destinada a centralizar los esfuerzos de investigación en Robótica realizados en Europa.

1.4.2 Sectores del mercado

Como indican las figuras 1.6 y 1.7 dentro de la Robótica actual están emergiendo distintos sectores, cada uno de ellos con sus propias líneas de investigación, su propio mercado de expansión y su propio campo de aplicación dentro de la sociedad actual. Los robots que seamos capaces de desarrollar en un futuro próximo serán los derivados del desarrollo de estos sectores.

Entre los sectores más importantes en los que se divide la Robótica actual se destacan:

- **Producción Industrial:** tradicionalmente la producción industrial ha sido el campo de aplicación por excelencia de la Robótica hasta nuestros días, persiguiendo la reducción de costes y el aumento en la productividad y calidad mediante la automatización de tareas repetitivas en entornos controlados. La expansión de la Robótica en este campo pretende aumentar la flexibilidad en la operación de los robots, adaptándose a la información del entorno captada por sus sensores y a las instrucciones proporcionadas por los operarios. Estos nuevos robots pretenden ser las nuevas herramientas inteligentes de los trabajadores del mañana, desarrollando las tareas repetitivas, laboriosas y peligrosas. De esta forma se pretende aumentar la competitividad y productividad de las industrias mientras se redunda

[1] `http://www.roboticsplatform.com`

en beneficios tan importantes como la reducción en los riesgos laborales, considerándose por lo tanto como un sector clave para el desarrollo económico y social.

- **Médico-Biológico:** es indudable que la esperanza y calidad de vida de los ciudadanos está directamente relacionada con los avances médicos alcanzados en la sociedad. La Robótica pretende incorporar importantes avances en el campo de la medicina: instrumental inteligente para operaciones, realización de operaciones remotas, monitorización y atención inteligente de pacientes, procesos de rehabilitación personalizados más eficientes, prótesis cibernéticas inteligentes, sensores y actuadores cibernéticos que devuelvan sensaciones y movimiento a los pacientes, ...

- **Servicios:** el bienestar social de los ciudadanos está estrechamente relacionado con los servicios que la sociedad sea capaz de proporcionarles. En este sentido, la Robótica puede contribuir con el desarrollo de nuevos robots autónomos que realicen numerosas tareas sin intervención humana: cuidado y recolección en agricultura, limpieza e inspección de edificios, transporte de mercancías y personas, vigilancia y operación forestal, construcción y demolición de edificios, operación en minas, mantenimiento de centrales nucleares, guía en museos, logística en oficinas, mantenimiento de estructuras subacuáticas, ...

- **Personal:** este sector es el que posee una mayor previsión de crecimiento en el futuro, abriendo un potencial mercado económico muy importante. Las mascotas domésticas, con el robot *Aibo* de *Sony* a la cabeza, están en constante expansión, introduciéndose en cada vez mayor número de hogares. En este sentido, robots cada vez más versátiles con funcionalidades más avanzadas son desarrollados para el consumo doméstico.

- **Seguridad y Espacio:** la exploración espacial ha sido quizás la meta más perseguida tradicionalmente por la Robótica. Si bien su desarrollo ha conseguido logros muy importantes, el perfeccionamiento de los robots implementados puede abrir nuevos campos de aplicación muy interesantes. Adicionalmente, la seguridad está empezando a cobrar una creciente importancia donde se preveen destacados avances en un futuro no muy lejano. Entre las principales aplicaciones donde pueden contribuir los robots en la seguridad y la exploración espacial se pueden destacar: vigilancia de fronteras y de infraestructuras críticas, persecución de actividades criminales, vigilancia y control marítimo y aéreo, operaciones de búsqueda y rescate, operación en situaciones críticas como catástrofes naturales, vigilancia y manipulación de materiales peligrosos, exploración y análisis de planetas lejanos, reparación y mantenimiento de estaciones espaciales y satélites, ...

Todos estos sectores, con sus perspectivas de crecimiento, servirán para desarrollar los robots que el día de mañana nos acompañarán en nuestras vidas cotidianas. El esfuerzo investigador que actualmente se está desarrollando indica que, aunque todavía lejano, quizás ya no sea tan aventurado vislumbrar un futuro no muy lejano donde el ser humano pueda coexistir con una nueva generación de robots inteligentes.

2 Navegación: definición y clasificación

Se puede afirmar que la navegación es un concepto común a cualquier tipo de móvil, y que se distingue de otras tipos de movimiento en la existencia de un objetivo final, que es el de alcanzar un destino determinado de forma segura. El concepto original de la palabra **navegar** [2] estaba asociado a la ciencia náutica, y hacía referencia al proceso de dirigir un barco hacia un destino mediante la aplicación iterativa de tres pasos:

1. Determinar la posición del barco de la forma más exacta posible empleando cartas náuticas.
2. Establecer mediante las cartas náuticas la relación existente entre la posición del barco, los puntos de referencia, los posibles peligros y el destino a alcanzar.
3. Fijar el nuevo rumbo del barco para alcanzar el destino siguiendo una ruta determinada.

La posición del barco se estimaba inicialmente mediante la técnica conocida como *dead reckoning*. Esta técnica consistía en medir la distancia y la ruta seguida por el barco a partir de un punto conocido, empleando para ello cartas náuticas y algún tipo de instrumento de medida. En general, el rumbo seguido por el barco se medía mediante una brújula magnética, instrumento utilizado en Europa desde finales del siglo XII. La distancia recorrida se calculaba multiplicando el tiempo de viaje por la velocidad del barco, la cual se estimaba arrojando una pieza de flotación a un lado del barco y midiendo el tiempo que tardaba en cruzar dos marcas consecutivas ubicadas en el propio barco. El tiempo utilizado en atravesar estas marcas se medía mediante el canto de una canción determinada, la cual comenzaba el timonel al cruzar la pieza de flotación la primera marca y detenía al cruzar la segunda. La última sílaba cantada se utilizaba para consultar una tabla de velocidades. Esta velocidad tenía que medirse cada hora y anotarse en un tablero denominado *toleta*, con el fin de actualizar la posición del barco de forma continua.

Este sistema de navegación era evidentemente bastante impreciso, por lo que en muchos casos obligaba a navegar cerca de la costa para divisar marcar significativas que permitiese corregir la posición de la nave de forma absoluta. A finales del siglo XV se introducen instrumentos como el *astrolabio* y posteriormente el *sextante*, que permitían una navegación marítima mucho más precisa basada en la orientación de las estrellas. Esta técnica constituye la base de los modernos sistemas de localización global por *GPS*, actualmente utilizados por un elevado número de sistemas de locomoción.

2.1 La navegación en la Robótica

La capacidad de navegación constituye una funcionalidad básica para cualquier agente autónomo móvil. Tradicionalmente, los conceptos relativos a la navegación náutica han sido aplicados con escasas modificaciones en el campo de la Robótica para la implementación de agentes autónomos

[2] Del latín: *navis*, barco; *agere*, mover.

móviles. Levitt y Lawton definen la navegación como un proceso iterativo compuesto por tres pasos básicos [LL90]:

1. Determinar la posición actual del agente.
2. Localizar otras ubicaciones respecto a la posición actual del agente.
3. Establecer la dirección para alcanzar estas otras ubicaciones desde la posición actual del agente.

Este esquema supone mantener una representación interna del entorno, dentro de la cual el agente debe conocer su posición y ser capaz de calcular una trayectoria libre de obstáculos hasta un destino determinado. Además, es necesario percibir los distintos obstáculos del entorno para evitar posibles colisiones con los mismos, ya se trate de obstáculos fijos o móviles. Esta es la estrategia de navegación básica implementada actualmente en muchos agentes autónomos móviles.

2.2 La Jerarquía de Navegación

Ningún sistema de navegación desarrollado sobre un agente autónomo móvil ha conseguido igualar la flexibilidad y eficiencia de los esquemas de navegación presentes en los organismos biológicos. Parece evidente que el escenario descrito anteriormente no parece ser el más adecuado en el reino animal, ya que muchos animales, entre los que se incluyen los seres humanos, son capaces de presentar comportamientos de navegación sin necesidad de afrontar las tres preguntas básicas de Levitt y Lawton.

Todo ello ha conducido a que numerosos investigadores traten de inspirarse en comportamientos animales a la hora de diseñar esquemas de navegación para agentes autónomos móviles, en lugar de seguir el ciclo básico propuesto por Levitt y Lawton. Esta nueva alternativa se conoce como **Navegación Biomimética**, y pretende estudiar y analizar los distintos esquemas de navegación de los organismos biológicos para poder imitarlos e implementar sistemas de navegación flexibles y eficientes.

Franz y Mallot realizaron un estudio y clasificación de los diferentes comportamientos de navegación presentes en los organismos biológicos, estableciendo una **Jerarquía de Navegación** donde se agrupan dichos comportamientos según su complejidad [FM00]. La Jerarquía de Navegación establece siete niveles de navegación diferentes distribuidos en dos grandes grupos, de forma que dentro de cada grupo los comportamientos de los niveles superiores manifiestan todos los comportamientos de los niveles inferiores. Los grupos y niveles que forman la Jerarquía de Navegación son los siguientes (figura 1.8):

1. **Navegación Local:** dentro de este grupo se engloban todos aquellos comportamientos que solo precisan identificar el destino que se desea alcanzar, sin necesidad alguna de poseer

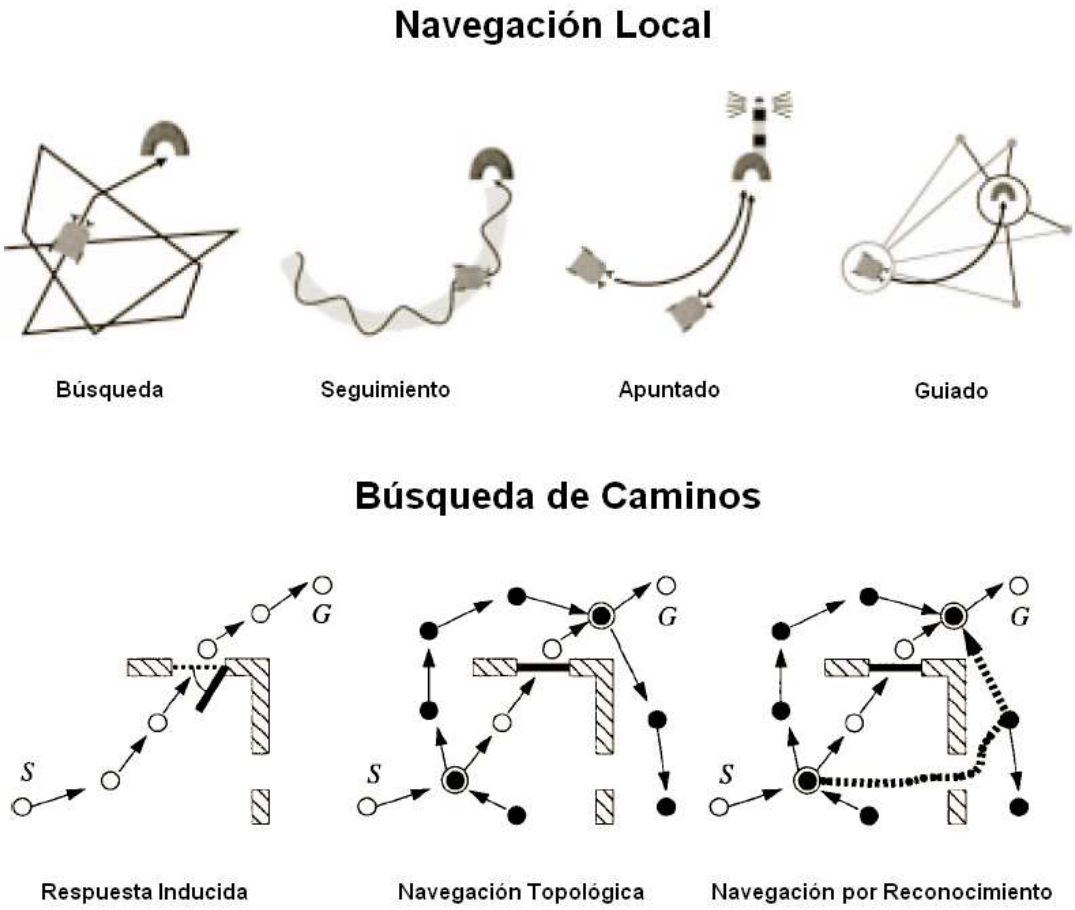

Figura 1.8: La Jerarquía de Navegación.

una representación del entorno, por lo que no realizan ningún proceso de planificación. Los niveles de navegación que pertenecen a este grupo, ordenados de menor a mayor complejidad, son los siguientes:

(a) **Búsqueda:** consiste en vagar de forma aleatoria hasta que se reconoce el destino que se desea alcanzar, sin mostrar ningún tipo de orientación hacia el mismo.

(b) **Seguimiento:** requiere alinear el rumbo actual seguido con una dirección previamente establecida. El rumbo puede ser medido a través de fuentes externas, como la ubicación relativa con las estrellas, o a través de fuentes internas, como sensores magnéticos. Si por alguna causa se pierde el rumbo que se debe seguir el destino no puede ser localizado.

(c) **Apuntado:** consiste en incorporar algún mecanismo que permita mantener el destino que se quiere alcanzar siempre al frente, identificado mediante algún tipo de marca que pueda ser detectada. Resuelve el problema del seguimiento, en tanto que se puede alcanzar el destino desde cualquier dirección y no únicamente siguiendo una previamente establecida.

(d) **Guiado:** permite alcanzar el destino basándose en su relación espacial con algunos elementos característicos del entorno. Así pues, el destino puede ser alcanzado aunque no exista ninguna señal que lo identifique.

2. **Búsqueda de Caminos:** dentro de este grupo se engloban todos aquellos comportamientos que utilizan una representación del entorno, empleando los comportamientos pertenecientes al grupo de navegación local para realizar el movimiento entre distintas localizaciones. Los niveles de navegación que pertenecen a este grupo, ordenados de menor a mayor complejidad, son los siguientes:

   (a) **Respuesta Inducida:** permite conectar dos puntos del entorno mediante comportamientos de navegación local, por lo que precisa reconocer la posición actual y la del destino a alcanzar. Una vez que se ha reconocido la posición actual, se activa un comportamiento de navegación local determinado para alcanzar el siguiente punto del entorno.

   (b) **Navegación Topológica:** realiza una representación del entorno mediante zonas características conectadas entre sí, de forma que para alcanzar un destino determinado se pueden utilizar distintos caminos, descritos por la secuencia de regiones a atravesar. Utiliza algún tipo de planificación para calcular el camino hasta el destino final, que puede ser adaptado según las circunstancias del entorno.

   (c) **Navegación por Reconocimiento:** incorpora en la representación del entorno aquellas regiones desconocidas que aun no han sido exploradas.

Se pueden encontrar numerosos ejemplos en los organismos biológicos que siguen estos comportamientos básicos de navegación, aunque el nivel más avanzado de Navegación por Reconocimiento está limitado aparentemente a los vertebrados. Generalmente, cada nivel de navegación requiere habilidades más complejas que los niveles inferiores, factor relacionado con el proceso evolutivo de las distintas especies.

3 Motivación de la Tesis

Se puede afirmar que la movilidad y la capacidad de llevar a cabo tareas de supervivencia en entornos dinámicos proporcionan la base para el desarrollo de un nivel de inteligencia superior [Mor84]. Un breve repaso por la evolución de la vida en nuestro planeta sugiere que capacidades como el lenguaje, el conocimiento, la resolución de problemas y el razonamiento en general han requerido menor tiempo para la naturaleza que dotar a los seres vivos de la capacidad de existir y reaccionar adecuadamente ante un entorno dinámico, moviéndose y captando las condiciones que le rodean y llevando a cabo las tareas básicas necesarias para asegurar el mantenimiento de la vida [Bro90].

Ante esta perspectiva, parece evidente que la resolución del problema de la movilidad constituye la base necesaria para poder desarrollar robots capaces de interactuar con el ser humano en

su ambiente natural. Con esta idea en mente, el objetivo del trabajo realizado en la presente Tesis consiste en el desarrollo de una infraestructura básica que permita incorporar la capacidad de navegación en entornos dinámicos no estructurados a un agente autónomo móvil genérico. Para ello se pretende implementar un sistema modular y flexible, capaz de integrarse con distintas plataformas robóticas y de permitir una fácil ampliación del sistema mediante nuevos comportamientos y funcionalidades.

Respecto al esquema de navegación del agente, se pretende desarrollar gradualmente la Jerarquía de Navegación propuesta por Franz y Mallot hasta implementar el nivel superior de Navegación por Reconocimiento, el cual manifiesta los comportamientos de navegación más complejos y versátiles. De esta forma, partiendo desde los niveles inferiores de la Jerarquía de Navegación se irán añadiendo capacidades cada vez más avanzadas para implementar los niveles superiores. Esta implementación gradual es quizás la aproximación más natural en la implementación de esquemas de navegación complejos, pues trata de emular la propia evolución de los distintos organismos biológicos.

Para aumentar la generalidad del sistema a desarrollar se va a suponer que el agente utilizado únicamente posee un sistema de posicionamiento basado en odometría y un sistema sensorial compuesto por sensores sonar. Este equipamiento es bastante básico y suele estar presente en un gran número de plataformas robóticas, aumentando así la posibilidad de aplicar directamente el sistema sobre un mayor número de agentes. No obstante, se pretende que el sistema a desarrollar sea lo suficientemente flexible como para incorporar otro tipo de sensores si se desea (brújula, cámaras, láser, *GPS*, ...), aumentando así las capacidades exhibidas por el mismo.

4 Estructura de la Tesis

La organización seguida en la presente Tesis es la que se describe a continuación:

- **Capítulo 2. Arquitecturas de Control:** en este capítulo se presentan las arquitecturas de control como herramienta imprescindible para simplificar el desarrollo de agentes autónomos móviles. Para ello se describe la utilidad del uso de las arquitecturas de control, realizando un recorrido histórico de su evolución y presentando algunas de las más importantes. Finalmente se analizan en profundidad las partes que integran una arquitectura de control y las prestaciones que caracterizan su funcionamiento.

- **Capítulo 3. La Arquitectura de Control *DLA*:** en este capítulo se aborda el diseño de una nueva arquitectura de control, destinada a implementar una infraestructura básica que incorpore la capacidad de navegación en entornos dinámicos no estructurados a un agente autónomo móvil genérico. Para ello se analiza la necesidad de desarrollar una nueva arquitectura de control, presentando las especificaciones básicas de la misma. Tras describir las posibles alternativas existentes a la hora de implementar la arquitectura de control, se escoge la más adecuada para satisfacer las especificaciones originales, diseñando tanto los mecanismos de interacción entre módulos que definen el estilo de la arquitectura como

la descomposición del sistema en módulos y capas que define la estructura de la misma. Finalmente, tras el proceso de diseño e implementación se describe el modo de utilización de la nueva arquitectura de control y los parámetros que caracterizan su funcionamiento.

- **Capítulo 4. Capas Física, de Representación y de Comando:** en este capítulo se describe la implementación de la Capa Física, la Capa de Representación y la Capa de Comando, necesarias para el correcto funcionamiento del sistema. Se detallan las funcionalidades básicas de cada una de ellas y se describen las decisiones escogidas en su desarrollo. Como parte del desarrollo de la Capa de Representación también se analizan las representaciones métricas del entorno.

- **Capítulo 5. Capa de Navegación:** en este capítulo se describe la implementación de la Capa de Navegación del sistema, responsable de desarrollar el comportamiento de navegación reactiva que permite alcanzar el destino final mientras se evita la colisión con los obstáculos fijos y/o móviles del entorno. Se analizan las principales alternativas existentes en la implementación de esquemas de navegación reactiva, profundizando en el paradigma del razonamiento basado en casos finalmente utilizado.

- **Capítulo 6. Capa de Planificación:** en este capítulo se describe la implementación de la Capa de Planificación del sistema, responsable de desarrollar los algoritmos de planificación adecuados que permitan aumentar la eficiencia del proceso de navegación. Tras presentar los distintos niveles de planificación incorporados en el sistema se analizan las posibles alternativas para cada uno de ellos, escogiendo las más adecuadas y procediendo al diseño y caracterización de las mismas.

- **Capítulo 7. Resultados:** en este capítulo se describen y analizan las pruebas realizadas sobre el sistema, destinadas tanto a verificar su correcto funcionamiento como a extraer las prestaciones finales que lo caracterizan.

- **Capítulo 8. Conclusiones y Líneas Futuras:** en este capítulo se presentan las principales conclusiones obtenidas con la realización de la presente Tesis, las principales contribuciones derivadas de la misma y las líneas futuras de trabajo que pueden ampliar su contenido.

- **Apéndice A. Guía de Instalación y Manual de Usuario:** en este apéndice se presenta la guía de instalación y el manual de usuario de la arquitectura de control propuesta.

Capítulo 2

Arquitecturas de Control

El desarrollo de agentes autónomos móviles constituye una labor lo suficientemente compleja como para requerir el uso de determinadas herramientas que simplifiquen dicho desarrollo. En este capítulo se describen las arquitecturas de control, las cuales constituyen la herramienta básica utilizada en la implementación de agentes autónomos móviles.

Este capítulo se ha organizado como se indica a continuación. En el apartado 1 se describe la utilidad y necesidad de uso de las arquitecturas de control. En el apartado 2 se realiza un recorrido histórico en la evolución de las arquitecturas de control, describiendo el estado del arte y analizando algunas de las arquitecturas de control existentes más destacadas. En el apartado 3 se analizan en profundidad las arquitecturas de control, describiendo las partes que las integran y los parámetros que caracterizan su funcionamiento. Por último, en el apartado 4 se describen las principales conclusiones extraídas a lo largo del presente capítulo.

1 Necesidad de las arquitecturas de control

Los agentes autónomos móviles constituyen sistemas bastante complejos de desarrollar, pues deben operar en entornos reales caracterizados por su complejidad y dinamismo. Para su correcto funcionamiento, pues, es imprescindible operar con un alto grado de incertidumbre en la distribución del entorno, reaccionar en tiempo real ante situaciones inesperadas, gestionar adecuadamente los errores en las lecturas de los sensores, mantener la integridad física del sistema evitando potenciales situaciones de riesgo de colisión, ...

Con el objetivo de simplificar en la medida de lo posible esta elevada complejidad, el diseño de agentes autónomos móviles ha sido tradicionalmente abordado siguiendo una estrategia de **descomposición**. Para ello es necesario dividir el sistema completo en unidades funcionales menores interconectadas entre sí, las cuales deben operar conjuntamente para desarrollar la funcionalidad total del sistema. Estas unidades funcionales se conocen como **módulos**, constituyendo entidades independientes de menor complejidad que el sistema completo, por lo que su diseño es más manejable y abordable. Finalmente, tras el desarrollo de todos los módulos es

necesario realizar la correcta **integración** de los mismos, de forma que se permita la operación conjunta de todos ellos para implementar el sistema total.

Dependiendo del sistema a desarrollar, tanto la correcta descomposición como la posterior integración pueden llegar a convertirse en problemas considerablemente complejos. Desde el punto de vista de la implementación de agentes autónomos móviles el problema de la descomposición suele ser relativamente abordable, pues prácticamente todos los agentes necesitan desarrollar un conjunto común de tareas básicas: localización del agente en el entorno, navegación mientras se evitan los obstáculos circundantes, modelado y representación del entorno, planificación del camino a seguir para alcanzar el destino, ...

Sin embargo, en el desarrollo de agentes autónomos móviles la integración de los distintos módulos puede llegar a constituir un problema bastante complejo si no es adecuadamente gestionado. Esta complejidad se ve reforzada por el hecho de que normalmente los distintos módulos del sistema suelen ser desarrollados en paralelo por distintos diseñadores, empleando cada uno de ellos distintas metodologías y herramientas en el desarrollo de los mismos. Así pues, la heterogeneidad resultante en el desarrollo de los distintos módulos suele dificultar más aun la integración de todos ellos en un entorno común, necesario para implementar el sistema completo.

Para manejar la evidente complejidad resultante al aplicar esta estrategia en el desarrollo de agentes autónomos móviles es prácticamente imprescindible disponer de alguna herramienta que facilite los procesos de descomposición e integración. Dicha herramienta se conoce como **arquitectura de control**, la cual permite definir la estructura de módulos del sistema y su interconexión, además de suministrar los mecanismos de interacción entre módulos adecuados para facilitar la integración de los mismos.

Las arquitecturas de control se han convertido en una parte imprescindible de cualquier agente autónomo móvil, formando el núcleo de dichos sistemas. De hecho, las funcionalidades finales que es capaz de manifestar el agente están directamente relacionadas con las prestaciones de la arquitectura de control sobre la cual se basa su desarrollo. En este sentido, el correcto diseño de la arquitectura de control sobre la que se desarrolla el sistema constituye uno de los aspectos más importantes y críticos en la implementación de agentes autónomos móviles.

2 Estado del arte

Para obtener una mayor comprensión de las arquitecturas de control resulta imprescindible realizar un recorrido a través de la evolución histórica de las mismas, pues los éxitos y los fracasos cosechados a lo largo de las últimas décadas han derivado en las características que actualmente muestran las arquitecturas de control contemporáneas.

2.1 Evolución de las arquitecturas de control

El interés en el desarrollo de arquitecturas de control para el desarrollo de agentes autónomos móviles está fuertemente influenciado, como no puede ser de otra forma, con el nacimiento y desarrollo de la Robótica. Este hecho se debe a la ineludible simbiosis existente entre las arquitecturas de control y los agentes autónomos móviles, pues debido a la complejidad de estos sistemas resulta impracticable el desarrollo de los mismos sin la utilización de alguna herramienta que simplifique su desarrollo. Así pues, la evolución histórica de las arquitecturas de control avanza paralelamente a la evolución histórica de la propia Robótica, coincidiendo con los intentos de la comunidad científica por desarrollar agentes capaces de sustituir al ser humano en la realización de determinadas tareas.

2.1.1 Arquitecturas deliberadas

Durante las décadas de los años 60 y 70 la visión dominante en la comunidad científica para el desarrollo de agentes autónomos móviles estaba fuertemente influenciada por la Inteligencia Artificial, la cual utilizaba la representación y la abstracción del conocimiento como base del razonamiento. Por aquella época, pues, se consideraba que un agente autónomo móvil debía desarrollar las siguientes tareas básicas [Nil82]:

- **Modelado:** responsable de generar un modelo del entorno a partir de los datos de entrada provenientes de los sensores.
- **Planificación:** responsable de elaborar un plan adecuado sobre el modelo del entorno disponible para llevar a cabo un objetivo determinado.
- **Ejecución:** responsable de generar las acciones necesarias para ejecutar correctamente el plan establecido.

Este paradigma se conoce como *SPA* (*Sense-Plan-Act*) [Alb91, HB96], y utiliza una representación interna del entorno sobre la que planifica la respuesta más adecuada en función de los objetivos perseguidos por el agente. Los sistemas basados en este tipo de filosofía se denominaron **sistemas deliberados**.

Las arquitecturas de control asociadas con este tipo de sistemas estaban compuestas por tres elementos funcionales básicos según una distribución horizontal, tal y como se muestra en la figura 2.1: un sistema sensorial (*Sense*) que realiza el modelado del entorno, un sistema planificador (*Plan*) que elabora planes sobre dicho modelo y un sistema ejecutor (*Act*) que lleva a cabo los planes generados. Cada uno de estos elementos podía ser descompuesto a su vez en varios módulos si era necesario. Este tipo de arquitecturas de control basadas en el modelado del entorno y en la planificación sobre dicho modelo de las acciones a desarrollar se denominaron **arquitecturas deliberadas**.

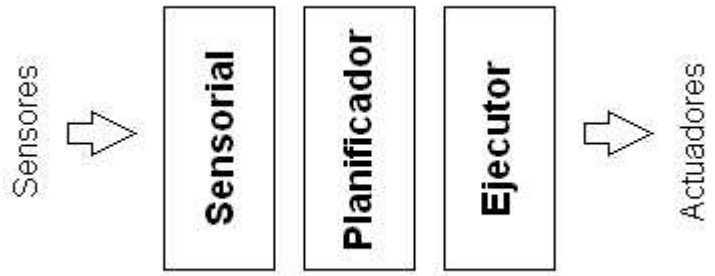

Figura 2.1: Estructura de las arquitecturas deliberadas.

Es evidente que estos sistemas seguían una filosofía descendente (*top-down*), descomponiendo el problema en una serie de etapas sucesivas de procesamiento, donde el flujo de información es lineal desde los sensores hasta los actuadores. Los esfuerzos de la comunidad científica en la implementación de agentes autónomos móviles durante esta época se centraban, pues, en la creación de modelos adecuados del entorno y en el desarrollo de algoritmos de planificación eficientes sobre dichos modelos [BAFH84, AML87]. No obstante, pronto se evidenciaron importantes limitaciones en este tipo de soluciones, entre las que se destacan [Gat98]:

- El tiempo de respuesta del sistema era considerablemente elevado, debido a que la información debía ser procesada linealmente por todos los módulos. Este hecho era el causante de que el sistema fuese incapaz de reaccionar con rapidez ante cualquier estímulo externo inesperado.
- Puesto que el tiempo de respuesta del sistema era elevado, el entorno podía cambiar considerablemente mientras se realizaba la planificación, de forma que cualquier situación imprevista no contemplada en la planificación provocaba la inviabilidad de los planes originalmente calculados.
- Todos los módulos del sistema debían estar sobredimensionados en su diseño, procesando y transfiriendo toda la información que se pudiera usar en cualquier momento en el sistema por módulos sucesivos, no solo la estrictamente necesaria para cada módulo.
- Los módulos debían preveer cualquier tipo de contingencia, pues el fallo de cualquier módulo provocaba el fallo del sistema completo al interrumpir el flujo lineal de información entre sensores y actuadores.
- La implementación gradual y depuración del sistema era compleja, pues para que el sistema comenzase a operar era necesario que todos los módulos estuviesen operativos para no interrumpir el flujo lineal del información entre sensores y actuadores.

Se han llevado a cabo algunos esfuerzos por solucionar los inconvenientes mostrados por este tipo de sistemas mediante la imposición de distintas restricciones temporales [Tso97], pero aun

Figura 2.2: Estructura de las arquitecturas reactivas.

así estos esquemas no han sido capaces de operar eficientemente en entornos dinámicos no estructurados, debido a la incertidumbre presente en entornos reales y a la consecuente dificultad de mantener sincronizados el entorno y la representación interna del mismo. Por estos motivos los sistemas deliberados se mostraron poco eficientes, siendo progresivamente descartados en la implementación de agentes autónomos móviles.

2.1.2 Arquitecturas reactivas

Durante la década de los años 80 se produjo el desuso del paradigma *SPA* basado en el modelado del entorno y en el desarrollo de complejos algoritmos de planificación sobre dicho modelo. La comunidad científica de la época comenzó a rechazar el desarrollo de modelos abstractos de representación simbólica y de complejos procesos de planificación, debido a la incapacidad que estaban mostrando estos sistemas en la resolución de problemas básicos como la navegación en entornos dinámicos no estructurados [Bro90, Bro91].

Como alternativa a la abstracción y representación del conocimiento, se comenzó a concebir el desarrollo de agentes autónomos móviles mediante el establecimiento de relaciones sencillas entre la percepción captada del entorno y la acción a desarrollar. Estas relaciones directas entre percepción y acción se conocen como **comportamientos**, definidos como acciones o respuestas reflejas a realizar a partir de la información directa recibida por los sensores [Ark98]. La combinación de varios comportamientos se traducía en lo que se conoce como **comportamiento emergente**, un resultado global más o menos complejo que, en principio, no podía predecirse con precisión ya que depende de la lectura de los sensores en un instante de tiempo determinado. Así pues, eliminando por completo la representación interna del entorno desaparece la necesidad de mantener una representación actualizada, resolviendo los problemas manifestados por los sistemas deliberados. Los sistemas basados en este tipo de filosofía se denominaron **sistemas reactivos**.

Las arquitecturas de control asociadas con este tipo de sistemas estaban compuestas por distintos elementos funcionales distribuidos verticalmente, tal y como como se muestra en la figura 2.2. Cada uno de estos elementos funcionales desarrollaban un tipo de comportamiento más o

menos complejo, y posteriormente todos los comportamientos eran integrados para desarrollar el comportamiento emergente final deseado, por lo que la arquitectura de control debía implementar algún tipo de mecanismo para permitir la adecuada cooperación entre comportamientos. Cada uno de estos elementos podía ser descompuesto a su vez en varios módulos si era necesario. Este tipo de arquitecturas de control basados en la integración de numerosos comportamientos se denominaron **arquitecturas reactivas**.

La innovación más importante de este paradigma radicaba en la utilización de una filosofía ascendente (*bottom-up*), realizando una descomposición vertical del sistema frente a la tradicional descomposición horizontal de las arquitecturas deliberadas. Esta descomposición vertical permitía que cada comportamiento tuviese una descripción completa del procesamiento a realizar sobre el flujo de información existente entre los sensores y los actuadores, en lugar de existir un único flujo lineal como ocurría en los sistemas deliberados. Esta filosofía presentaba importantes ventajas, entre las que se destacan:

- El tiempo de respuesta de cada módulo era independiente del tiempo de respuesta de los demás módulos, por lo que se podían implementar comportamientos más rápidos que respondían a situaciones críticas y comportamientos más lentos que llevasen a cabo un procesado más profundo. Así pues, el sistema podía reaccionar rápidamente a situaciones inesperadas.
- Los módulos no necesitaban estar sobredimensionados, ya que procesaban flujos completos de información entre sensores y actuadores.
- El fallo de un módulo no implicaba el fallo del sistema completo, pues no se interrumpía el flujo de información en el resto de módulos.
- El sistema podía desarrollarse y depurarse gradualmente, sin necesidad de que todos los módulos estuviesen implementados. Así pues, se podían ir añadiendo progresivamente funcionalidades más complejas basándose en las previamente implementadas.

El desarrollo de los sistemas reactivos fue posible gracias al trabajo de Rodney Brooks, cuya **arquitectura de subsunción** delegó definitivamente el uso de las arquitecturas deliberadas y estableció los importantes conceptos que permitieron una nueva filosofía en el desarrollo de agentes durante su época. Debido precisamente a esta importancia es imprescindible analizar con mayor profundidad la arquitectura de subsunción para comprender la evolución de las arquitecturas de control hasta nuestros días.

La arquitectura de subsunción

La arquitectura de subsunción propuesta por Rodney Brooks [Bro86] constituye uno de los puntos de inflexión más importantes en el desarrollo contemporáneo de la Robótica. Se trata de una arquitectura reactiva que utiliza distintos módulos para implementar comportamientos básicos que posteriormente son combinados entre sí. La figura 2.3 muestra la estructura de la arquitectura de subsunción.

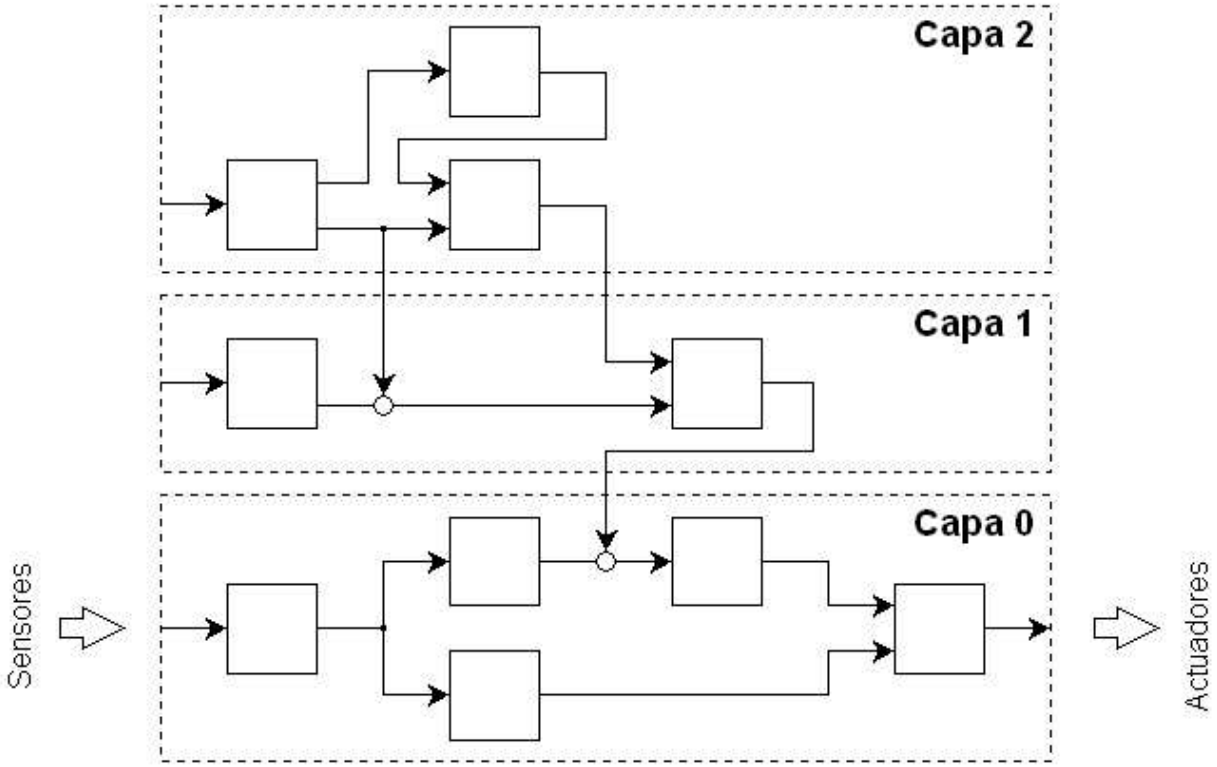

Figura 2.3: Estructura de la arquitectura de subsunción.

En la arquitectura de subsunción los distintos comportamientos básicos son desarrollados por lo que se conoce como **capa**, una entidad abstracta que agrupa a todos aquellos módulos que cooperan entre sí en el desarrollo de un determinado comportamiento. Estas capas interaccionan entre sí según una estructura jerárquica, por lo que la implementación del sistema se realiza de forma gradual: partiendo de comportamientos básicos se añaden nuevas capas que implementan nuevos comportamientos. Los mecanismos para garantizar la cooperación e interacción entre comportamientos consisten en la **supresión** y la **inhibición** temporal que las capas superiores efectúan sobre determinados flujos de información entre los módulos de las capas inferiores. Todos los flujos de información que se intercambian en el sistema son asíncronos, es decir, los nuevos datos que genera cada módulo pueden sobreescribir los antiguos si estos no fueron leídos, no realizando ninguna confirmación de los mismos por parte del módulo que los recibe. Este funcionamiento garantiza que los datos de entrada sobre los que opera cada módulo son siempre los más recientes.

En oposición a las arquitecturas deliberadas, centradas en modelar el entorno y elaborar planes adecuados a dicho modelo, la arquitectura de subsunción experimentó un temprano éxito en el área de la navegación y evitación de obstáculos, implementándose agentes capaces de navegar en entornos reales con relativa facilidad. Su mayor éxito y quizás la cima de sus posibilidades la

alcanzó con *Herbert*, un agente programado para localizar y recoger latas vacías en un entorno real de una oficina [Con89b]. Si bien el funcionamiento de *Herbert* no fue todo lo correcto que se esperaba, las capacidades desarrolladas fueron muy destacables, aunque parecen definir un límite de lo que se puede implementar mediante la arquitectura de subsunción.

No obstante, tras algunos años en los que sucesivamente se demostraba la enorme capacidad de operación de la arquitectura de subsunción, comenzaron a aparecer algunas serias limitaciones, entre las que se destacan [HP91]:

- Una descomposición jerárquica de los comportamientos no siempre es apropiada, puesto que pueden existir comportamientos complementarios que no necesariamente se relacionen de forma jerárquica, sino que se encuentren al mismo nivel. También pueden existir comportamientos independientes que operen simultáneamente, sin necesidad de que uno sea de más alto nivel que otro.
- No es lo suficientemente modular, ya que debido a los mecanismos de supresión e inhibición que las capas superiores ejercen sobre las capas inferiores estas no se pueden diseñar totalmente independientes unas de otras, aumentando la complejidad conforme aumenta la funcionalidad y el número de capas del sistema. Así pues, la modificación de una de las capas puede derivar en la necesidad de modificar otras capas.
- El mecanismo de cooperación entre comportamientos consistente en la supresión y la inhibición no es siempre el más apropiado, pues existen situaciones en las que son las capas inferiores las que deben modificar el funcionamiento de las capas superiores, como por ejemplo en aquellas situaciones críticas detectadas por las capas inferiores que implican un peligro para el sistema. Existen otras aproximaciones o alternativas quizás más eficientes a este problema de cooperación entre comportamientos, como pueden ser la **composición** y la **selección** [Ark89, Ros95].

2.1.3 Arquitecturas híbridas

Durante la década de los años 90 se experimentó un afloramiento de numerosas arquitecturas de control, algunas derivadas de la arquitectura de subsunción [RP89, GDI$^+$94] y otras desarrolladas de forma relativamente independiente [Kae86, GL87, Sol90, Ark90, Sim90]. La experiencia acumulada durante el desarrollo de estos nuevos sistemas puso de manifiesto que si bien los sistemas reactivos eran robustos y fácilmente extensibles, capaces de navegar en entornos dinámicos no estructurados, poseían algunos inconvenientes importantes, entre los que se destacan [Gat98]:

- El comportamiento emergente final del agente era bastante complejo de determinar, pues dependía de la compleja interacción entre numerosos comportamientos básicos.
- Al no existir ningún tipo de planificación el sistema podía comportarse de forma poco eficiente en determinadas circunstancias, incluso bloquearse y quedar atrapado en trampas de mínimos locales.

- Puesto que la percepción y la acción se acoplaban directamente, el sistema podía fallar si la percepción del entorno que se captaba no coincidía con el entorno debido a algún tipo de fallo en los sensores empleados. En este sentido, una representación interna del entorno podría ser útil para detectar o corregir los posibles fallos momentáneos de los sensores.

Debido a estos inconvenientes la comunidad científica de la época comenzó a percibir que si bien los sistemas deliberados basados únicamente en planificación eran incapaces de enfrentarse eficientemente a la incertidumbre asociada con el mundo real, la ausencia completa de planificación de los sistemas reactivos conducía a agentes incapaces de mostrar comportamientos avanzados y eficientes. Es por ello por lo que comenzó a emerger una nueva corriente que pretendía combinar las características mostradas por los sistemas deliberados con aquellas mostradas por los sistemas reactivos, intentando reutilizar lo mejor de cada uno. Los sistemas basados en este tipo de filosofía se denominaron **sistemas híbridos**.

Tooth [Ang89] y *Rocky III* [MDG+92] fueron unos de los primeros agentes autónomos móviles implementados con esta nueva filosofía. Ambos estaban programados para recolectar objetos, si bien *Tooth* estaba diseñado para operar en entornos interiores y *Rocky III* en entornos exteriores. Estos agentes presentaban capacidades muy parecidas a las de *Herbert*, si bien su comportamiento era mucho más fiable, siendo capaces de operar de forma autónoma durante largos períodos de tiempo. La arquitectura de control implementada con *Tooth* y *Rocky III* se basaba en la arquitectura de subsunción, pero a diferencia de ésta aceptaba el uso de la abstracción y la representación del conocimiento. Sin embargo, a pesar de encontrarse entre los primeros agentes autónomos móviles capaces de realizar tareas más complejas que la simple navegación, *Tooth* y *Rocky III* tenían una seria limitación: la arquitectura de control implementada no permitía la reprogramabilidad de los agentes, por lo que no podían modificar la tarea a realizar sin reescribir el código que los controlaba.

La búsqueda de nuevas arquitecturas de control para el desarrollo de sistemas híbridos que pudiesen realizar diversas tareas continuó durante algún tiempo, hasta que finalmente distintos investigadores que trabajaron más o menos independientemente resolvieron este problema de forma similar al mismo tiempo: Connell, Bonasso y Gat. La arquitectura de control propuesta por Connell se denominó *SSS* [Con92] y se basó en la arquitectura de subsunción de Brooks [Bro86], la propuesta por Bonasso se denominó *3T* [Bon91] y se basó en el estudio de Kaelbling [Kae88], y la propuesta por Gat se denominó *ATLANTIS* [Gat91] y se basó en el trabajo de Firby [Fir89].

Las arquitecturas de control propuestas por Connell, Bonasso y Gat estaban formadas por tres capas básicas, tal y como se muestra en la figura 2.4: una capa reactiva de bajo nivel, una capa deliberada de alto nivel y una capa intermedia que actúa de interfaz entre ambas. Cada una de estas capas podía ser descompuesta a su vez en varios módulos si era necesario. Este tipo de arquitecturas de control basadas en la combinación de las filosofías reactiva y deliberada se denominaron **arquitecturas híbridas**.

Las tres arquitecturas de control propuestas por Connell, Bonasso y Gat eran muy semejantes en su concepción, y se descomponían principalmente en tres componentes, por lo que se denom-

Figura 2.4: Estructura de las arquitecturas híbridas.

inaron arquitecturas *Three-Layer*. La importante mejora presentada por estas arquitecturas radicaba en su capacidad de combinar eficientemente las respuestas rápidas de los sistemas reactivos con los procesos de planificación de los sistemas deliberados, por lo que se convirtieron probablemente en el ejemplo más representativo de arquitecturas híbridas, siendo utilizadas como referencia en numerosos trabajos [Gat98].

2.2 Arquitecturas de control actuales

Tras los buenos resultados obtenidos con las arquitecturas híbridas *Three-Layer* la comunidad científica reconoció que una filosofía híbrida constituía la mejor aproximación para abordar la implementación de agentes autónomos móviles, pues permite combinar las filosofías deliberadas y reactivas en una arquitectura heterogénea, facilitando el diseño de comportamientos reactivos a bajo nivel conectados con un razonamiento deliberado de alto nivel. El interfaz entre ambas filosofías puede ser complejo de realizar, y debe ser cuidadosamente diseñado para conseguir la fusión adecuada de ambos paradigmas.

Actualmente el desarrollo de agentes autónomos móviles se basa en la utilización de arquitecturas de control híbridas. Sin embargo, a pesar del evidente interés generado en las arquitecturas de control no se ha desarrollado todavía una metodología estructurada para abordar el diseño de las mismas. Tampoco se han definido formalmente esquemas que permitan medir cuantitativamente las prestaciones ofrecidas por una arquitectura de control, por lo que resulta muy complejo establecer comparaciones entre ellas [CMS00, OC03]. Este escenario ha originado la aparición en los últimos tiempos de una gran cantidad de soluciones particulares y adaptadas al problema concreto que cada una pretende resolver, dificultando enormemente su reutilización en nuevos sistemas. Coste-Manière presenta una excelente visión de la situación en el diseño de arquitecturas de control en la última década [CMS00]. Otros trabajos importantes en el estudio de las arquitecturas de control son los de Gat [Gat98] y Orebäck [OC03].

Algunas de las arquitecturas de control más importantes desarrolladas en los últimos tiempos son: *AuRA* [Ark90], *CIRCA* [MDS93], *TCA* [Sim94], *DAMN* [Ros95], *Saphira* [KM96], *Remote Agent* [MNPW97], *3T* [BFG+97], *ControlShell* [SCPCW98], *OrCCAD* [BCME+98],

LAAS [ACF+00] y *BERRA* [LOC00].

Aunque parece evidente que todos los sistemas deben incorporar comportamientos reactivos, algunos imponen restricciones más estrictas en términos de ejecución en tiempo real. Algunas arquitecturas como *CIRCA* [MDS93] facilitan el control de la frecuencia de ejecución de los distintos comportamientos, mientras que muchas otras arquitecturas como la de subsunción [Bro86] y *3T* [BFG+97] operan sin ninguna restricción en tiempo real, simplemente llevando a cabo una ejecución tan rápida como sea posible, dependiendo por supuesto de las características y necesidades de la aplicación que esta alternativa sea suficiente. Muchas arquitecturas de control facilitan la concurrencia en la ejecución de los distintos módulos, algunas como *ControlShell* [SCPCW98] en un único procesador y otras como *TCA* [Sim94] en un entorno distribuido multiprocesador. Muchas arquitecturas de control incluyen algunas facilidades para arbitrar entre comportamientos, bien bloqueando algunos de ellos mientras se ejecutan otros, como es el caso de las arquitecturas de subsunción [Bro86] y *3T* [BFG+97], bien combinando la salida de múltiples comportamientos para formar un comportamiento emergente mayor, como es el caso de las arquitecturas *AuRA* [Ark89] o *DAMN* [Ros95]. El bloqueo de comportamientos normalmente produce resultados más predecibles, si bien la combinación de los mismos tiende a comportamientos más flexibles y potentes. Debido a la naturaleza modular de los agentes autónomos móviles es muy importante disponer de facilidades que permitan verificar los distintos componentes antes de que el sistema completo esté operativo, así como capacidades de monitorización que permitan analizar la evolución del sistema en tiempo real sin alterar el funcionamiento del mismo. Estas son características incorporadas en un gran número de arquitecturas.

La principal conclusión que se puede extraer de esta gran diversidad es que no existe una arquitectura de control por excelencia que satisfaga todas las necesidades existentes en la implementación de agentes autónomos móviles. Cada una de las arquitecturas existentes propone sus propias facilidades para simplificar el desarrollo de agentes autónomos móviles, por lo que la elección de la arquitectura más idónea depende claramente de la aplicación final que se desee implementar. En muchos casos , además, diferentes arquitecturas pueden ser empleadas para implementar sistemas que realicen tareas similares, como por ejemplo robots recolectores de objetos [Con89a, SLF90] y robots capaces de caminar [Bro89, SK91]. Las principales diferencias entre la utilización de una arquitectura u otra radican en la facilidad de implementación del sistema y en la eficiencia del mismo en el desarrollo de las tareas para las que fue diseñado.

Para profundizar en el conocimiento de las arquitecturas de control contemporáneas y analizar su estructura con mayor detalle se van a presentar a continuación algunas de las arquitecturas más importantes que se han desarrollado en los últimos tiempos.

2.2.1 Arquitectura *3T*

La arquitectura *3T* [Bon91] es una de las arquitecturas *Three-Layer* originales, por lo que sigue una filosofía muy flexible y ha conducido al exitoso desarrollo de numerosas aplicaciones [BFG+97]. Posee las tres capas de este tipo de arquitecturas, los cuales deben cooperar concurrentemente y de forma asíncrona. Normalmente son procesos distintos que se pueden imple-

mentar en distintos procesadores o en un sistema multiproceso. En el caso de la arquitectura *3T* estas tres capas son:

- **Reactiva:** contiene un conjunto de comportamientos reactivos (denominados *skills*), que posibilitan la interacción con el mundo, realizando el procesamiento necesario de la percepción del entorno para generar los comandos adecuados enviados al agente. Existe un gestor de dichos comportamientos (denominado *skill manager*) que permite la comunicación entre comportamientos. Además, el gestor presenta un interfaz común en el uso de cualquier comportamiento, facilitando la activación y desactivación de los mismos mediante la correcta gestión de eventos asíncronos, necesarios para coordinar los comportamientos.

  Los comportamientos son complejos de implementar, puesto que necesitan cumplir ciertas restricciones para poder ser integrados en la arquitectura. En este sentido, todo comportamiento debe poseer una especificación de los datos de entrada y salida, un algoritmo que procese los datos de entrada y genere los datos de salida, una rutina de inicialización cuando el comportamiento se activa por primera vez, una función de activación y una función de desactivación. Para aliviar la complejidad en la creación de comportamientos se han desarrollado herramientas que generan la estructura básica que debe tener todo comportamiento, debiendo implementar únicamente la funcionalidad particular que se quiere que desarrolle cada uno de ellos.

- **Secuenciador:** encargado de activar y desactivar el conjunto de comportamientos adecuados en el orden correcto para ejecutar una acción determinada. Para describir la correcta descomposición de una determinada acción en un conjunto de pasos discretos se emplea una estructura conocida como *RAP* (*Reactive Action Package*) [Fir89]. El secuenciador de la arquitectura, pues, consiste en el intérprete de *RAPs*. Es importante notar que cualquier *RAP* debe describir la reacción del sistema ante el conjunto de circunstancias externas que se quieran detectar, puesto que la forma en que se ejecuta la acción mediante la activación y desactivación de los comportamientos adecuados depende del estado del entorno. El secuenciador, pues, consiste en una librería de *RAPs*, cada uno de ellos dedicados a ejecutar una determinada acción mediante el uso de un conjunto de comportamientos en un orden adecuado.

- **Planificador:** responsable de generar los planes necesarios para alcanzar la consecución de determinados objetivos, contemplando aspectos como uso de recursos y restricciones temporales. El secuenciador por sí mismo no está estructurado para ejecutar tareas complejas y generar acciones alternativas ante la no consecución de determinados objetivos. El planificador se encarga de organizar la secuencia de acciones a desarrollar para ejecutar una tarea global y más compleja, considerando la implicación que tiene la ejecución de diversas acciones en el desarrollo de dicha tarea global. El planificador empleado es el *AP* (*Adversarial Planner*) [ES94], el cual fue concebido como un planificador multiagente.

2.2.2 Arquitectura *TCA*

La arquitectura *TCA* (*Task Control Architecture*) [Sim94] es una arquitectura orientada a tareas bastante eficiente. Su filosofía consiste en implementar una base deliberada para tratar las situaciones normales sobre la que se añaden progresivamente nuevos comportamientos reactivos para manejar las situaciones excepcionales. Esta división permite una implementación progresiva del sistema, incrementando la funcionalidad del mismo al añadir nuevos comportamientos sin modificar los anteriores.

La arquitectura *TCA* no implementa ningún comportamiento en particular, sino que proporciona aquellos mecanismos y herramientas software que se necesitan para desarrollar e integrar los comportamientos deseados. Está concebida como un sistema operativo para sistemas robóticos, ofreciendo numerosas prestaciones para la gestión y coordinación de las diferentes tareas que pueda requerir el sistema: comunicación entre módulos distribuidos, descomposición de tareas para elaboración de planes, manejo adecuado de recursos compartidos, monitorización del progreso del sistema, gestión de excepciones y eventos, ...

La arquitectura *TCA* está formada por un elemento central de control y un número variable de módulos que se ejecutan en distintos procesadores. Los módulos están implementados en lenguaje *C* o *Lisp*, y se conectan únicamente al elemento central de control mediante un proceso de registro donde cada módulo indica su ubicación. La comunicación entre módulos se realiza mediante un sistema de mensajes que son adecuadamente encaminados por el elemento central hacia sus destinatarios. Este esquema de comunicación ofrece importantes ventajas, pues la información necesaria para comunicar entre sí los distintos módulos se establece de forma dinámica y no hay que conocerla a priori. De hecho ningún módulo conoce exactamente la ubicación de los demás módulos del sistema, por lo que es posible activar y desactivar selectivamente los módulos sin necesidad de reestructurar al resto. Los mensajes que se intercambian los módulos entre sí se caracterizan por un nombre, el cual es conocido por cualquier módulo independientemente de su ubicación.

Los módulos se comunican con el elemento central de control mediante una librería implementada y facilitada a tal efecto, y tras registrar el nombre de los mensajes que desean procesar pasan a un estado de espera. El elemento central recibe los distintos mensajes que se intercambian en el sistema, y con la información registrada en el mismo envía cada uno de los mensajes a todos aquellos módulos que están interesados en su procesamiento. Cuando un módulo recibe un mensaje por parte del elemento central de control, se activa y comienza el procesamiento del mismo. Los mensajes se guardan en colas en el elemento central de control, por lo que no se pierden y son correctamente procesados en orden. Indicar, por último, que existen distintos tipos de mensajes, que permiten implementar con mayor flexibilidad la cooperación entre distintas tareas.

2.2.3 Arquitectura *LAAS*

La arquitectura *LAAS* [ACF+98] fue desarrollada en los laboratorios *LAAS (Laboratoire d'Analyse et d'Architecture des Systèmes)* en 1998. Es una arquitectura muy robusta y potente, cuya versatilidad ha quedado demostrada en diversos proyectos [ACF+00]. Es muy similar en su concepción a las arquitecturas *Three-Layer*, distinguiéndose tres grandes capas en la misma:

- **Nivel Funcional:** contiene una serie de comportamientos que implementan las acciones básicas de percepción y acción que será capaz de desarrollar el sistema, es decir, implementa el sistema reactivo. Estos comportamientos se encapsulan en *módulos*, que son entidades de software capaces de desarrollar un número de funciones específicas mediante el procesamiento de sus datos de entrada.

  Todo módulo integra diferentes *servicios*, cada uno de los cuales es un algoritmo de procesamiento que implementa una función determinada. Los distintos servicios de un módulo se pueden activar mediante un protocolo cliente/servidor asíncrono (es decir, no bloqueante), en el que una petición por parte de un cliente inicia la actividad del servicio solicitado. Al finalizar la ejecución del servicio se devuelve un informe al cliente que solicitó su ejecución. Este modo de operación garantiza por un lado que el sistema sea distribuido, al poder ejecutarse los módulos en distintos procesadores, y por otro lado que cualquier módulo pueda ser cliente de otro, por lo que todos los módulos se pueden activar entre sí, permitiendo la agrupación de comportamientos sencillos en otros más complejos. Es importante destacar que las relaciones entre módulos no se conocen a priori, y son establecidas dinámicamente. Esto dota al sistema de una gran flexibilidad, puesto que los módulos se pueden ubicar en cualquier procesador y se pueden incorporar o eliminar del sistema en tiempo de ejecución.

  Los módulos intercambian datos entre sí mediante un esquema de memoria compartida, en una estructura de datos conocida como *poster*, la cual contiene los datos producidos por un módulo mediante el procesamiento que realiza. Aunque cualquier módulo puede tener acceso a cualquier poster, sólo el propietario del mismo puede modificar su contenido, evitando así problemas de consistencia.

  Los módulos son entidades complejas en la arquitectura *LAAS*. Tienen que poseer una estructura determinada para poder ser integrados correctamente en la arquitectura, soportando peticiones cliente/servidor asíncronas, gestionando eventos para la activación y desactivación de las actividades, compartiendo datos mediante posters, ... Puesto que la implementación de estos módulos es muy compleja, se ha desarrollado un lenguaje específico que con la ayuda del correspondiente compilador genera el esqueleto de la estructura necesaria para encapsular los algoritmos de procesamiento adecuados. Este compilador se ha llamado *GenoM*, y acepta código en *C* para generar módulos que pueden ser ejecutados en *Linux*.

- **Nivel de Ejecución:** controla y coordina la ejecución de los distintos comportamientos del Nivel Funcional según la acción a ejecutar. No posee capacidad alguna de planificación, simplemente recibe una secuencia de acciones a ejecutar, y en función del estado

del sistema se encarga de seleccionar, parametrizar y sincronizar dinámicamente las actividades adecuadas del Nivel Funcional que se deben activar. Por lo tanto, es el responsable de mantener un estado actualizado del sistema, en función del resultado obtenido en el procesamiento de los datos que realizan las distintas actividades de los módulos.

El Nivel de Ejecución posee un conjunto de reglas que asigna a cada acción los distintos comportamientos a ejecutar, creando informes del proceso de las tareas y resolviendo los posibles conflictos que puedan ocurrir entre distintos módulos en el uso de los recursos. Para ello puede asignar prioridades, interrumpir módulos o dejar peticiones pendientes. Para desarrollar el Nivel de Ejecución es preciso establecer el conjunto de reglas lógicas que coordina la ejecución de los comportamientos, lo cual puede ser una tarea muy compleja. Para facilitar su implementación, se ha definido un lenguaje formal y un entorno de programación, conocido como *Kheops*, que genera automáticamente el conjunto de reglas necesarias para implementar el Nivel de Ejecución a partir de una sencilla descripción en forma de proposiciones lógicas.

- **Nivel de Decisión:** descompone la tarea a desarrollar en una serie de acciones adecuadas para ejecutarla, que serán posteriormente enviadas al Nivel de Ejecución. Engloba las capacidades deliberadas del sistema, por lo que incluye un *planificador* que produce la secuencia de acciones necesarias para realizar una tarea determinada y un *supervisor* que monitoriza y controla la ejecución del plan desarrollado y reacciona ante posibles eventos, tomando las decisiones adecuadas en función de los informes de consecución de una acción desarrollados por el Nivel de Ejecución.

  El planificador implementado en el Nivel de Decisión se conoce como *IxTeT*, y se basa en formalismos lógicos, mientras que el supervisor se crea mediante la herramienta *TransGen*. Ambos se programan mediante *PRS* (*Procedural Reasoning System*), que es un lenguaje de programación que define un conjunto de herramientas y métodos para representar y ejecutar planes.

2.2.4 Arquitectura *BERRA*

La arquitectura *BERRA* (*BEhavior-based Robot Research Architecture*) [LOC00] es muy similar en su concepción a la arquitectura *3T*, y ha sido diseñada con el objetivo básico de proporcionar una gran flexibilidad en el desarrollo de agentes autónomos. Utiliza el lenguaje *C++* y se basa en el paquete *ACE* (*Adaptive Communication Environment*) [Sch94], por lo que puede ser utilizada en todos aquellos sistemas operativos donde se encuentre desarrollado dicho paquete, como *Linux*. Es una arquitectura distribuida que ha sido diseñada de forma que cualquier proceso pueda ser transparentemente ubicado en cualquier máquina de la red.

La arquitectura *BERRA* posee, al igual que la arquitectura *3T*, tres capas:

- **Deliberada:** encargada de interaccionar con el operador humano mediante el interfaz *HRI* (*Human Robot Interface*) y de interpretar y generar los planes de alto nivel adecuados mediante un planificador.

- **Ejecución:** encargada de mantener actualizada continuamente la posición del agente mediante un localizador y de gestionar la correcta ejecución de los comportamientos reactivos mediante el supervisor *TES* (*Task Execution Supervisor*), el cual traduce los planes recibidos por el planificador de la capa deliberada al conjunto de comportamientos necesarios para ser llevada a cabo.
- **Reactiva:** encargada de recoger la información de los sensores mediante el acceso a los recursos del agente, ejecutar los algoritmos reactivos almacenados en los comportamientos y enviar los comandos adecuados a los actuadores del propio agente.

Por último, para manejar la correcta interacción entre todos los procesos del sistema de la forma más transparente posible se ha desarrollado un módulo especializado en dicha labor, el gestor *PM* (*Process Manager*). Debe existir un gestor *PM* en cada una de las máquinas que se usen para distribuir el sistema, y todos ellos son los responsables de mantener actualizada en cada momento la ubicación de cada uno de los procesos del sistema a través de un mecanismo de registro. Esto facilita la libre distribución de los procesos en la red mientras que se garantiza un mecanismo para ser localizados por cualquier otro proceso del sistema.

3 Descripción de las arquitecturas de control

Una arquitectura de control constituye una herramienta bastante compleja, cuyo uso permite incorporar importantes propiedades en el sistema final con el objetivo de simplificar su desarrollo. En este apartado se analiza en profundidad la configuración interna de las arquitecturas de control, describiendo posteriormente los principales parámetros que caracterizan su funcionamiento.

3.1 Niveles de descripción

El objetivo básico de toda arquitectura de control es la de reducir la complejidad existente en el desarrollo del sistema. Para ello, debe ser capaz de proporcionar los mecanismos adecuados que simplifiquen los correspondientes procesos de descomposición e integración del sistema. En este sentido, cualquier arquitectura de control se puede dividir en dos **niveles** [CMS00]:

- **Estructura:** aborda el problema de la descomposición del sistema completo en módulos, definiendo la funcionalidad de cada uno de estos módulos así como las interacciones entre los mismos.
- **Estilo:** aborda el problema de la integración de los distintos módulos del sistema, definiendo los mecanismos computacionales que permiten el correcto intercambio de información entre módulos.

Ambos niveles son complementarios, de forma que los requisitos y prestaciones de uno delimitan los requisitos y las prestaciones del otro.

3.2 Parámetros de descripción

Toda arquitectura de control posee una serie de parámetros que describen su funcionamiento. Entre los parámetros más importantes que suelen poseer las arquitecturas de control se pueden destacar los siguientes [OC03]:

- **Sencillez:** un característica muy importante para el éxito de una arquitectura de control es la facilidad de uso. Idealmente debería operar en un nivel lo más transparente posible al usuario que desarrolla el sistema, ya que una arquitectura de control muy potente pero excesivamente compleja está destinada a ser utilizada por un reducido número de personas especializadas, imponiendo una carga considerable de aprendizaje en su uso.
- **Escalabilidad:** se entiende por escalabilidad la capacidad de incrementar la potencia del sistema conforme las necesidades del mismo van aumentando, sin detrimento en las prestaciones exhibidas. Una de las técnicas más extendidas para garantizar la escalabilidad del sistema se basa en posibilitar su distribución entre varios procesadores, para lo cual es necesario que se suministren mecanismos que permitan la comunicación distribuida entre módulos. También es recomendable utilizar otras técnicas para optimizar el uso de los recursos disponibles, como puede ser la capacidad de activar y desactivar módulos dinámicamente en tiempo de ejecución, según sean o no necesarios en el procesamiento de los datos de entrada para llevar a cabo una tarea determinada.
- **Extensibilidad:** por extensibilidad se entiende la capacidad de sustituir o incluir fácilmente nuevos componentes en el sistema. Esta es una característica muy importante en el diseño de agentes autónomos móviles, puesto que dichos sistemas en el entorno de la investigación tienden a evolucionar y ampliar paulatinamente sus capacidades. Una arquitectura de control extensible debería manifestar esta característica en dos niveles:
  - **Hardware:** referido a la incorporación de nuevos sensores y actuadores en el el agente para aumentar sus capacidades de interacción con el entorno.
  - **Software:** referido a la incorporación de nuevos comportamientos en el agente para aumentar su funcionalidad.
- **Portabilidad:** puesto que el hardware y el software están generalmente sujetos a cambios, la portabilidad es un objetivo altamente deseable en el diseño de agentes autónomos móviles. Una arquitectura de control portable debería manifestar esta característica en dos niveles:
  - **Hardware:** referido a las distintas plataformas robóticas existentes, cada cual con una serie de sensores y actuadores. Aunque los fabricantes de plataformas robóticas normalmente ofrecen distintas soluciones y sistemas, las características básicas de todas ellas son semejantes, utilizando motores y sensores básicos muy similares. El propio fabricante suele proporcionar una librería software que permite el uso de la plataforma robótica mediante comandos básicos de alto nivel, como lectura de sensores y comandos de movimiento. La arquitectura de control debería encapsular

los comandos específicos de las distintas plataformas robóticas en un conjunto estandarizado de comandos, manteniendo un interfaz de comandos genéricos independientemente de la plataforma robótica empleada.

  - **Software:** referido a las distintas plataformas y lenguajes de programación que puedan soportar el uso de la arquitectura de control. Para ello, la arquitectura debería facilitar los mecanismos adecuados para ampliar al máximo posible el uso de la misma en diferentes plataformas y lenguajes, para lo que se suele proporcionar un conjunto adecuado de librerías software para el uso de la arquitectura en dichas plataformas y lenguajes.

- **Flexibilidad:** se entiende por flexibilidad la capacidad de adaptarse a los requisitos y necesidades de la aplicación particular desarrollada, minimizando las restricciones impuestas sobre los distintos módulos a la hora de desarrollar el sistema. A medida que se aumenta el número de restricciones impuestas por una arquitectura de control para su uso se suele disminuir la gama de aplicaciones donde puede ser empleada.

- **Depuración:** puesto que los agentes autónomos móviles constituyen sistemas complejos que suelen constar de numerosos módulos que interaccionan entre sí, es altamente deseable disponer de facilidades que permitan depurar dichos sistemas. En este sentido, suele ser recomendable poder verificar aisladamente el correcto funcionamiento de los distintos módulos antes de integrarlos en el sistema, y también disponer de herramientas de monitorización en tiempo real del estado del sistema para controlar la evolución del mismo.

- **Eficiencia:** es indispensable que la sobrecarga que introduce en el sistema el uso de la arquitectura de control sea lo suficientemente reducida como para no alterar el funcionamiento del propio sistema. En este sentido, se suelen emplear como parámetros de sobrecarga el uso de la memoria y de los procesadores que la arquitectura de control utiliza de las máquinas existentes en el sistema.

4 Conclusiones

En el presente capítulo se han descrito las numerosas ventajas obtenidas mediante el uso de una arquitectura de control en la implementación de agentes autónomos móviles. Estos sistemas son lo suficientemente complejos como para emplear algún tipo de estrategia que permita simplificar su desarrollo. Tradicionalmente, el diseño de agentes autónomos móviles se ha abordado mediante la **descomposición** en módulos de menor complejidad, requiriendo posteriormente una correcta **integración** de todos los módulos una vez implementados.

Si bien el proceso de descomposición suele ser bastante abordable en el diseño de agentes autónomos móviles, el proceso de integración es un problema que crece en complejidad a medida que aumenta el número de módulos implicados en el desarrollo del sistema. Si la complejidad derivada de dicha integración no es abordada eficientemente se puede llegar a convertir en un problema inmanejable. En este sentido, una **arquitectura de control** constituye una

herramienta muy potente para simplificar el desarrollo de sistemas complejos, al proporcionar mecanismos adecuados para facilitar los procesos de descomposición y de integración.

Un recorrido histórico a lo largo de la evolución de las arquitecturas de control muestra que ha habido distintas propuestas en el desarrollo de arquitecturas de control, en consonancia con la propia evolución de la Robótica en las últimas décadas: **arquitecturas deliberadas**, **arquitecturas reactivas** y **arquitecturas híbridas**. Las arquitecturas híbridas que combinan respuestas rápidas y procesos de planificación constituyen la alternativa más eficiente que presenta mejores prestaciones, por lo que se han adoptado como la solución utilizada hoy en día en el desarrollo de agentes autónomos móviles.

Desgraciadamente, a pesar de su importancia no se poseen actualmente ni metodologías estructuradas de diseño ni métricas adecuadas para las arquitecturas de control, por lo que es difícil abordar un diseño sistemático y realizar una comparativa entre ellas. Es por ello por lo que se han desarrollado numerosas arquitecturas de control, no existiendo una definitiva que manifieste todas las prestaciones que se cabría esperar de un sistema de este tipo. Más bien, cada una de estas arquitecturas de control suele resolver los problemas particulares de la aplicación para la cual fue desarrollada, dificultando su reutilización en nuevos sistemas.

Una arquitectura de control se descompone en dos grandes niveles: una **estructura** y un **estilo**. La estructura hace referencia a la descomposición del sistema en módulos, mientras que el estilo hace referencia a los mecanismos computacionales empleados para facilitar la integración de los distintos módulos. Ambos niveles son complementarios, de forma que los requisitos y prestaciones de uno delimitan los requisitos y las prestaciones del otro. Entre los principales parámetros que caracterizan el funcionamiento de una arquitectura de control se destacan: **sencillez**, **escalabilidad**, **extensibilidad**, **portabilidad**, **flexibilidad**, facilidad de **depuración** o **eficiencia**.

Capítulo 3

La Arquitectura de Control *DLA*

Una vez analizada la necesidad de utilizar una arquitectura de control para facilitar el desarrollo de agentes autónomos móviles se aborda en el presente capítulo la propuesta de una nueva arquitectura. Esta nueva arquitectura permite la implementación de cualquier sistema distribuido, y se ha denominado ***DLA*** (***D****istributed and* ***L****ayered* ***A****rchitecture*). Mediante la utilización de esta arquitectura se pretende implementar la infraestructura básica que incorpore la capacidad de navegación en entornos dinámicos no estructurados a un agente autónomo móvil.

Este capítulo se ha organizado como se indica a continuación. En el apartado 1 se analiza la necesidad de implementar una nueva arquitectura de control en lugar de utilizar alguna de las ya existentes. En el apartado 2 se presentan las especificaciones básicas de partida para la implementación de la nueva arquitectura de control. En el apartado 3 se describe el proceso de diseño del estilo de la arquitectura de control, implementando los mecanismos de interacción adecuados entre los distintos módulos. En el apartado 4 se aborda el diseño de la estructura de la arquitectura de control, describiendo la descomposición realizada del sistema en módulos y capas. En el apartado 5 se resume la correcta utilización de la arquitectura de control propuesta y de los elementos básicos que la componen. En el apartado 6 se analizan los parámetros que caracterizan la operación de la arquitectura de control. Por último, en el apartado 7 se describen las principales conclusiones extraídas a lo largo del presente capítulo.

1 Justificación de la arquitectura *DLA*

Como ya se ha comentado en el apartado 1 del capítulo 2, el desarrollo de agentes autónomos móviles constituye una tarea bastante compleja que se suele abordar mediante la descomposición del sistema en módulos menores, los cuales deben ser posteriormente integrados para posibilitar su operación conjunta. En el caso particular del desarrollo de agentes autónomos móviles el proceso de descomposición suele ser relativamente abordable, definiéndose con cierta facilidad un conjunto de módulos que implementen los comportamientos básicos necesarios durante el proceso de navegación: localización, evitación de obstáculos, representación del entorno, planificación de caminos, ...

Esta estrategia de descomposición es muy adecuada no sólo por reducir la complejidad en el desarrollo del agente autónomo móvil, sino por posibilitar además la división del trabajo entre varios diseñadores, los cuales pueden trabajar de forma relativamente independiente en cada uno de los módulos del sistema. Durante el desarrollo de estos módulos, es altamente deseable que cada diseñador emplee las herramientas que considere más apropiadas, pues de esta forma se puede optimizar el desarrollo de los mismos según sus necesidades particulares. Así pues, en función de los conocimientos de cada diseñador y de las necesidades de cada módulo se deberían poder utilizar distintas herramientas (*JBuilder*, *VisualC*, ...), distintos lenguajes (*C*, *C++*, *JAVA*, *Matlab*, ...) y/o distintas plataformas (*Linux*, *Windows*, *Unix*, ...) en su desarrollo.

Con esta filosofía de trabajo donde el diseñador posee una total libertad para elegir las herramientas de trabajo más adecuadas se facilita la división del trabajo y la optimización en el desarrollo de los distintos módulos, reduciendo en consecuencia el tiempo de desarrollo del sistema. Sin embargo, la falta de metodología común para el desarrollo de los distintos módulos suele derivar en una alta **heterogeneidad** de los módulos resultantes, cada uno de los cuales puede haber sido desarrollado en un lenguaje y en una plataforma determinados. Este escenario, si bien es deseado desde el punto de vista del desarrollo de los módulos, tiende a convertir el proceso de integración del trabajo realizado por los distintos diseñadores en un problema muy complejo.

Si se quiere mantener esta filosofía donde cada diseñador posea la libertad de escoger las herramientas más adecuadas para realizar su trabajo, se hace prácticamente imprescindible la utilización de una arquitectura de control apropiada que reduzca la complejidad en el proceso de integración de todos estos módulos heterogéneos. Puesto que se considera que este esquema de trabajo es quizás el más apropiado para el desarrollo de agentes autónomos móviles, esta es la estrategia seguida en el trabajo realizado en la presente Tesis.

Como ya se ha comentado en el apartado 2.2 del capítulo 2, la falta de metodologías de diseño estructuradas y de métricas adecuadas para diseñar y comparar las arquitecturas de control ha originado la aparición de numerosas arquitecturas de control en los últimos años. Cada una de estas arquitecturas constituye una solución particular adaptada al problema concreto que pretende resolver, no existiendo una arquitectura por excelencia que satisfaga todas las necesidades. Es por ello por lo que se debe escoger cuidadosamente la arquitectura de control utilizada para resolver el problema particular que se presenta en esta Tesis: la integración de los distintos módulos heterogéneos que componen el agente autónomo móvil.

Puesto que el número de diseñadores implicados en el desarrollo de un agente autónomo móvil puede llegar a ser relativamente elevado, se considera un objetivo primordial para facilitar este proceso de integración que la arquitectura de control utilizada presente un alto grado de **transparencia** en su uso, concretando esta pretendida transparencia en dos aspectos básicos:

- **Sencillez:** puesto que el número de diseñadores implicados en la utilización de la arquitectura de control puede ser elevado, es deseable minimizar la carga de trabajo necesaria en su aprendizaje. Una arquitectura de control muy potente pero relativamente compleja está destinada a ser utilizada por un reducido número de personas expertas.

- **Portabilidad:** puesto que se posee una considerable heterogeneidad en los módulos desarrollados, es deseable permitir el uso del mayor número posible de lenguajes de programación y plataformas de desarrollo, con el fin de optimizar el desarrollo de los módulos del sistema.

Para escoger la arquitectura de control deseada que posibilite la filosofía de trabajo anteriormente mencionada se han analizado las arquitecturas de control híbridas más significativas. El análisis detallado de estas revela que cada una de ellas es una solución particular que plantea sus propias restricciones y prestaciones para el desarrollo de agentes autónomos móviles. Muchas de ellas son el fruto de un gran trabajo de investigación con un coste temporal y humano bastante considerable, por lo que presentan numerosas funcionalidades que permiten usarlas en una gran cantidad de sistemas distintos: pase de mensajes, sistemas de prioridades, intercambio de datos asíncronos, esquemas de sincronización, manejo de excepciones, ...

Sin embargo, ninguna de las arquitecturas analizadas satisface plenamente el objetivo perseguido de la transparencia en su uso, pues son relativamente complejas de utilizar y, sobre todo, muestran una limitada portabilidad al imponer restricciones bastante importantes en los lenguajes y plataformas de uso. Esta circunstancia dificulta claramente la filosofía de trabajo que se pretende utilizar en el desarrollo de agentes autónomos móviles, por lo que la alternativa de utilizar una arquitectura de control existente ha sido descartada.

Así pues, con el objetivo de resolver exactamente las necesidades específicas de la filosofía de trabajo que se pretende utilizar en el desarrollo de agentes autónomos móviles se va a desarrollar una nueva arquitectura de control. Esta alternativa permite centrarse en la transparencia como factor determinante en el desarrollo de la arquitectura, buscando siempre la sencillez de uso y la portabilidad deseada mediante la minimización de las restricciones impuestas en el desarrollo de los distintos módulos.

La nueva arquitectura de control se ha denominado ***DLA*** (***D****istributed and* ***L****ayered* ***A****rchitecture*), y permite implementar sistemas cooperativos mediante la interacción de procesos libremente distribuidos. Esta arquitectura ha sido utilizada para implementar una infraestructura básica que incorpore la capacidad de navegación en entornos dinámicos no estructurados a un agente autónomo móvil de cualquier tipo, aunque es posible utilizarla en otro tipo de sistemas cooperativos incluso fuera del ámbito de la Robótica.

Antes de comenzar con el diseño de la nueva arquitectura, se presenta el análisis realizado sobre las principales arquitecturas de control híbridas existentes, así como las conclusiones obtenidas de dicho análisis.

1.1 Arquitectura *3T*

La arquitectura *3T* presenta una alta flexibilidad en su uso. Su clásica descomposición en tres capas permite integrar rápidamente comportamientos reactivos y deliberados en un entorno distribuido, incorporando además un secuenciador y un planificador que pueden ser programados

para la elaboración de tareas complejas.

Sin embargo, esta arquitectura plantea también algunos inconvenientes, entre los que cabe destacar:

- Impone restricciones demasiado importantes en la generación de los módulos, ya que es necesario utilizar herramientas específicas para su desarrollo y se deben definir a priori las interacciones entre los mismos, no permitiendo una interacción dinámica según las necesidades puntuales del sistema.

- Si bien es posible utilizar distintos lenguajes de programación en el desarrollo de los módulos (*C*, *C++*, *Pascal* y *LISP*) el lenguaje natural para la programación de los mismos es *LISP*, por lo que el empleo de los demás lenguajes implica un proceso de adaptación del código. No se permite el empleo de otros lenguajes de programación quizás más apropiados para algunas tareas, como por ejemplo *JAVA* o *Matlab*.

- Se ha desarrollado sobre una plataforma *Macintosh*, por lo que no es posible utilizar otras plataformas como *Linux* o *Windows*, que pueden ser más idóneas para algún tipo particular de módulos.

- Los módulos desarrollados tienden a ser bastante dependientes de la plataforma robótica utilizada, por lo que la migración a otra plataforma robótica distinta puede suponer una reescritura de una parte considerable de los comportamientos implementados.

1.2 Arquitectura *TCA*

La arquitectura *TCA* es muy eficiente en su operación. Ha sido concebida como un sistema operativo para sistemas robóticos, presentando un elevado número de funcionalidades distintas como comunicación distribuida, gestión de eventos y excepciones, elaboración de planes, ...

Aun siendo la más sencilla de utilizar de todas las analizadas, esta arquitectura plantea también algunos importantes inconvenientes, fundamentalmente centrados en el aspecto de la portabilidad:

- Los módulos sólo se pueden programar en *C* o en *LISP*, imposibilitando el empleo de otros lenguajes de programación quizás más apropiados para algunas tareas, como por ejemplo *JAVA* o *Matlab*.

- Su portabilidad a distintas plataformas está limitada, pues si bien es posible utilizarla a través de varios distribuciones del sistema operativo *Linux*, no es posible emplear otras plataformas como *Windows*, que pueden ser más idóneas para algún tipo particular de módulos.

1.3 Arquitectura *LAAS*

La arquitectura *LAAS* es muy robusta y potente. Su descomposición en tres capas permite desarrollar numerosos comportamientos en un entorno distribuido, incorporando además un gestor de comportamientos y un planificador que pueden ser programados para la elaboración de tareas complejas.

A pesar de haber sido utilizada exitosamente en numerosos proyectos esta arquitectura también plantea algunos inconvenientes:

- Es muy compleja de utilizar, pues hay que utilizar numerosas herramientas adicionales en el desarrollo del sistema, como el lenguaje de programación y el uso del compilador de módulos *GenoM*, el lenguaje y el entorno de programación *Kheops*, el planificador *IxTeT*, la herramienta de implementación *TransGen* y el lenguaje de programación *PRS*.
- Impone restricciones demasiado severas en la generación de los distintos módulos, puesto que es necesario usar el compilador específico *GenoM* y el lenguaje *C* para programarlos. No se permite el empleo de otros lenguajes de programación quizás más apropiados para algunas tareas, como por ejemplo *JAVA* o *Matlab*.
- Su portabilidad a distintas plataformas está limitada, pues si bien es posible utilizarla a través de varios distribuciones del sistema operativo *Linux*, no es posible emplear otras plataformas como *Windows*, que pueden ser más idóneas para algún tipo particular de módulos.

1.4 Arquitectura *BERRA*

La arquitectura *BERRA* muestra una elevada flexibilidad en su uso. Está concebida para permitir una distribución prácticamente transparente de los módulos del sistema, permitiendo una muy sencilla expansión del mismo.

No obstante, esta arquitectura también plantea algunos inconvenientes, entre los que se pueden destacar:

- Es una arquitectura bastante compleja, con un elevado número de directorios y archivos. Esto la convierte en un sistema difícil de comprender y utilizar.
- Sólo admite como lenguaje de programación *C++*, no aceptando otros lenguajes de programación quizás más apropiados para algunas tareas, como por ejemplo *JAVA* o *Matlab*.
- Aunque es posible utilizarla a través de varios sistemas operativos como *Linux*, no es posible utilizarla bajo la plataforma *Windows*, que puede ser más adecuada para algún tipo particular de módulos.
- No implementa facilidades de depuración, lo cual complica el desarrollo del sistema al consistir en numerosos módulos ejecutándose concurrentemente.

2 Especificaciones de la arquitectura *DLA*

El primer paso en la implementación de una nueva arquitectura de control es definir claramente las especificaciones de partida que se persiguen en el desarrollo de la misma. En los siguientes apartados se describen estas especificaciones, derivadas de las necesidades que se intentan resolver con la utilización de la arquitectura.

2.1 Niveles de descripción

Toda arquitectura de control se compone de dos grandes bloques básicos: una estructura y un estilo. Ambos bloques están estrechamente relacionados, pues las distintas necesidades de interacción que presenten los módulos entre sí marcarán las distintas funcionalidades de comunicación que el estilo de la arquitectura deberá suministrar. Así pues, resulta imposible realizar el diseño de cualquiera de ellos sin considerar las necesidades del otro.

Analizando detenidamente las necesidades que se pretenden cubrir con la arquitectura de control se pueden determinar las prestaciones básicas que deben manifestar la estructura y el estilo de la misma. Esta prestaciones son las que se presentan a continuación.

2.1.1 Estructura

La estructura de la arquitectura de control hace referencia a la descomposición realizada del sistema en módulos. Como se ha comentado en el apartado 2.1.3 del capítulo 2 una arquitectura de control híbrida constituye la alternativa más eficiente en el desarrollo de agentes autónomos móviles, por lo que la nueva arquitectura de control seguirá dicha filosofía. El aspecto fundamental de una arquitectura de control híbrida radica en la adecuada integración de comportamientos reactivos con comportamientos deliberados. Mientras los comportamientos reactivos deben ser simples en su operación para poder reaccionar con rapidez ante los distintos estímulos captados del entorno, los comportamientos deliberados deben realizar un procesamiento más profundo por lo general bastante lento. La característica clave, pues, es que existen distintas escalas temporales en la operación del sistema, siendo algunos módulos más rápidos que otros.

Ante una perspectiva de este tipo, no parece razonable implementar un esquema síncrono que opere bajo demanda, pues esto podría imponer restricciones muy severas en el funcionamiento del sistema al tener que esperar unos módulos quizás más rápidos la respuesta de otros módulos quizás bastante más lentos. La mejor filosofía ante una situación como la descrita donde existen distintas escalas temporales es seguir un esquema **asíncrono**, en el que cada módulo opera a la velocidad que necesite e intercambia los datos con el resto de módulos cuando lo requiera, evitando que ningún módulo deba esperar la llegada de datos de ningún otro módulo [And91]. Una estrategia asíncrona constituye uno de los mecanismos de comunicación entre procesos más eficiente y flexible, puesto que es un mecanismo de comunicación muy flexible, siendo el mecanismo normalmente implementado en la mayoría de las funciones de comunicación de los

distintos sistemas operativos. De hecho, este es el esquema originalmente propuesto por Rodney Brooks en la arquitectura de subsunción [Bro86].

Por lo tanto, la filosofía de operación de la nueva arquitectura de control seguirá un esquema asíncrono, por lo que el sistema deberá ser convenientemente descompuesto en distintos módulos que operen asíncronamente entre sí.

2.1.2 Estilo

El estilo de la arquitectura de control hace referencia a los mecanismos de interacción implementados para permitir la adecuada integración de los distintos módulos. Las especificaciones de partida para el desarrollo adecuado de estos mecanismos de interacción en relación con las objetivos perseguidos son las siguientes:

- **No bloqueantes:** puesto que el sistema debe ser descompuesto en una serie de módulos con una filosofía de operación asíncrona, los distintos mecanismos de interacción entre módulos deben garantizar una operación no bloqueante, para que los módulos que los utilicen no queden a la espera de recibir la respuesta solicitada del resto de módulos.
- **Tipos de datos:** puesto que se considera como aspecto fundamental de la arquitectura de control la transparencia en su uso, estos mecanismos de interacción deben minimizar las posibles restricciones impuestas a los módulos que los utilicen, permitiendo el intercambio de toda clase de información sin imponer ningún tipo predefinido de estructura de datos a intercambiar.
- **Portabilidad:** puesto que se desean integrar diversos módulos heterogéneos desarrollados con diferentes lenguajes y sobre diversas plataformas, es necesario garantizar la portabilidad de los mecanismos de interacción implementados entre dichos lenguajes y plataformas.
- **Distribuidos:** puesto que se pretenden utilizar distintos módulos implementados en diferentes plataformas, se debe posibilitar la distribución de los diversos módulos del sistema entre estas plataformas, por lo que los mecanismos de interacción implementados entre módulos deben ser capaces de operar en sistemas distribuidos.

2.2 Parámetros de descripción

El funcionamiento de una arquitectura de control queda caracterizado por un conjunto de parámetros que permiten determinar las prestaciones de la misma. Los parámetros perseguidos en la implementación de la nueva arquitectura de control son los que se describen a continuación:

- **Sencillez:** constituye un objetivo prioritario implementar una arquitectura fácil de utilizar. Para ello debe imponer las mínimas restricciones posibles en el diseño de los módulos y constituir una herramienta transparente que cualquier diseñador pueda utilizar sin conocer ningún detalle interno de implementación.

- **Escalabilidad:** se pretende que la arquitectura de control permita la ampliación de la capacidad de proceso del sistema conforme se incrementan las necesidades del mismo.

- **Extensibilidad:** se considera indispensable posibilitar una sencilla ampliación del sistema mediante la incorporación de nuevos componentes que garanticen el progresivo crecimiento del mismo. Se pretenden desarrollar esta característica en dos niveles:

  - **Hardware:** incorporando nuevo equipamiento en el agente que permita mejorar su interacción con el entorno.
  - **Software:** incorporando nuevos comportamientos que permitan desarrollar tareas más complejas.

- **Portabilidad:** constituye un objetivo prioritario implementar una arquitectura capaz de ser utilizada en distintos lenguajes y plataformas. En este sentido se busca la portabilidad en dos niveles:

  - **Hardware:** para garantizar el éxito en el uso de la arquitectura de control, es necesario que la misma pueda operar sobre distintas plataformas robóticas con los mínimos cambios posibles en los distintos módulos del sistema. Idealmente debería abstraer los detalles de implementación de cada plataforma robótica y presentar siempre un interfaz común al resto de módulos del sistema.
  - **Software:** para proporcionar la transparencia necesaria en el desarrollo de los distintos módulos del sistema, es necesario que la arquitectura de control admita distintos lenguajes de programación (*C*, *C++*, *JAVA*, *Matlab*, ...) y distintas plataformas (*Linux*, *Windows*, *Unix*, ...), es decir, que sea una arquitectura de control heterogénea.

- **Flexibilidad:** se persigue la minimización de las restricciones en el uso de la arquitectura de control, permitiendo la distribución transparente de los distintos módulos del sistema y el intercambio de cualquier tipo de información entre dichos módulos.

- **Depuración:** es deseable proporcionar algún tipo de herramienta que simplifique el proceso de depuración del sistema, para reducir el tiempo de desarrollo necesario en cada uno de los módulos del mismo.

- **Eficiencia:** se pretende que la sobrecarga que la arquitectura de control introduce en el sistema sea lo más reducida posible, para no alterar el funcionamiento de los distintos módulos diseñados aisladamente.

3 Estilo de la arquitectura *DLA*

En este apartado se va a diseñar el estilo de la arquitectura de control *DLA* propuesta, analizando las distintas alternativas existentes a la hora de interconectar distintos módulos entre sí y escogiendo aquella que satisfaga las especificaciones originales descritas en el apartado 2.

3.1 Análisis del estilo

El estilo de la arquitectura de control hace referencia a todos aquellos mecanismos de intercambio de información que permiten la correcta interacción entre los distintos módulos del sistema [CMS00]. Existen obviamente distintas alternativas a la hora de implementar estos mecanismos de intercambio de información, cada una con unas características determinadas y más eficientemente aplicable a cierto tipo de problemas. Un excelente estudio de las alternativas más importantes que se han desarrollado a lo largo de las últimas décadas es el realizado por Mary Shaw y Paul Clements [SC96]. En dicho estudio se clasifican los estilos más utilizados en la implementación de sistemas cooperativos, distinguiéndose los siguientes tipos:

- **Flujo de datos:** estilos centrados en la existencia de un flujo lineal de datos a través del sistema, los cuales van atravesando distintos módulos que los van procesando convenientemente.
- **Llamada a procedimientos:** estilos enfocados a un orden de procesamiento síncrono, donde un módulo realiza una llamada a otro módulo y espera la respuesta del mismo para continuar su procesamiento.
- **Interacción entre procesos:** estilos caracterizados por la comunicación entre distintos módulos independientes que se ejecutan concurrentemente.
- **Agentes centralizados:** estilos gestionados por la existencia de un agente central encargado de almacenar los datos del sistema, sobre los cuales los distintos módulos realizan un procesamiento selectivo coordinado por el propio agente central.
- **Datos compartidos:** estilos dirigidos a almacenar el conjunto de datos que deben ser compartidos por los distintos módulos del sistema.
- **Estructura jerárquica:** estilos dominados por realizar una división jerárquica entre módulos, limitando las interacciones posibles a las existentes entre un módulo y los correspondiente módulos del nivel inferior.

Estas alternativas se pueden combinar entre sí, y algunas de ellas comparten determinadas características comunes. De entre todas ellas, algunas no son apropiadas para la arquitectura de control que se pretende implementar. El flujo de datos realiza un procesamiento lineal de los datos por parte de los módulos del sistema, circunstancia que no tiene por qué adecuarse a la comunicación entre módulos que se pretende desarrollar. La llamada a procedimientos queda directamente descartada por requerir un esquema de comunicación síncrono. La estructura jerárquica imposibilita la interacción directa entre módulos de igual nivel, restricción demasiado severa para ser válida en todos los sistemas.

Las alternativas restantes (interacción entre procesos, agentes centralizados y datos compartidos) sí son válidas para la implementación de la nueva arquitectura de control, pues ninguna impone restricciones adicionales en la comunicación entre módulos ni en el orden de procesamiento de los

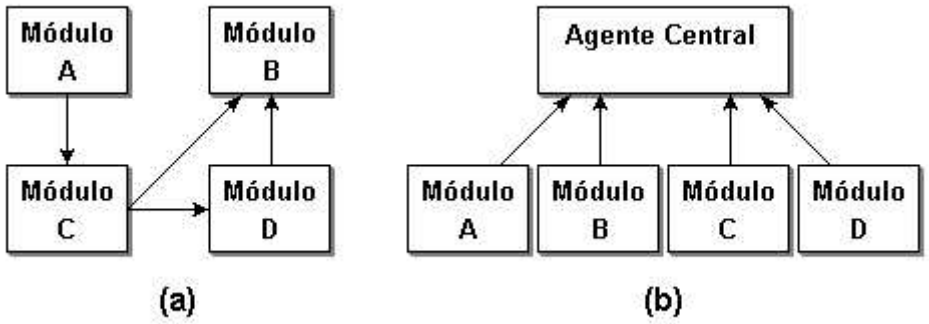

Figura 3.1: Esquemas de interconexión entre módulos: a) interconexión directa; b) interconexión indirecta.

datos. En el caso particular de la arquitectura a implementar se persigue una alta flexibilidad en la interacción entre módulos, para permitir la sencilla reestructuración de dichas interacciones a medida que el sistema se modifica o se amplía. Ante esta necesidad de alto grado de flexibilidad en el sistema la alternativa más eficiente para implementar sistemas cooperativos la constituye la **interacción entre procesos** [SC96], que consiste en implementar los distintos módulos del sistema de forma que operen independiente y concurrentemente, intercambiando entre sí los datos necesarios para la correcta operación de cada uno de ellos.

Ahora bien, existen diferentes estrategias a la hora de interconectar distintos módulos entre sí. Tradicionalmente la cooperación entre módulos se ha realizado mediante la **interconexión directa** entre los mismos según el flujo de datos necesarios entre ellos [Cor03], como se aprecia en la figura 3.1.a. En este esquema punto a punto las conexiones entre módulos están predeterminadas a priori, siendo una aproximación eficiente si el conjunto de módulos así como su conexionado es estático. Los esquemas de interconexión directa son más sencillos de implementar, pero no facilitan características tan importantes como la extensibilidad y la escalabilidad del sistema, pues un cambio en cualquier módulo implica la reestructuración de todos los módulos que interaccionan con el mismo.

En sistemas complejos que requieren un desarrollo incremental y donde el conjunto de módulos está sujeto a continuos cambios, como es el caso de los agentes autónomos móviles, la aproximación de interconexión directa es realmente ineficiente. Una aproximación mucho más eficiente en este tipo de sistemas sometidos a constantes modificaciones consiste en la **interconexión indirecta** entre ellos mediante algún tipo de agente central que gestione la interacción entre los módulos [Cor03], como se aprecia en la figura 3.1.b. Este esquema incrementa considerablemente la eficiencia en el desarrollo del sistema, pues permite conexiones dinámicas y anónimas entre módulos, ya que cada uno está conectado únicamente al agente central y desconoce la existencia y ubicación del resto. Así pues, la modificación de cualquier módulo no implica la modificación del resto.

Los esquemas de interconexión indirecta son más complejos de implementar, pero resultan mucho más eficientes en el desarrollo de sistemas complejos. Entre otras características, facilitan con-

siderablemente la extensibilidad y la escalabilidad del sistema, y aumentando la funcionalidad del agente central permiten añadir herramientas de monitorización y depuración para simplificar el desarrollo del sistema. Es interesante notar que algunas de las arquitecturas de control híbridas más importantes siguen este esquema de interconexión indirecta (*3T*, *TCA* y *BERRA*).

Por último, independientemente de la distribución y conectividad de los módulos del sistema es necesario escoger el mecanismo de intercambio de datos adecuado entre los mismos. Entre los **mecanismos** más importantes que existen para el intercambio de datos entre módulos se pueden destacar [Ste90]:

- **Ficheros:** permiten el intercambio de datos mediante operaciones de lectura y escritura en disco. Están restringidos a un ámbito local y son lentos por requerir acceso al disco.
- **Memoria compartida:** permiten el intercambio de datos mediante operaciones de lectura y escritura en memoria. Están restringidos a un ámbito local y son los más rápidos en realizar las operaciones de lectura y escritura.
- **Llamadas a procedimientos remotos:** permiten el intercambio de datos mediante la ejecución síncrona de funciones y procedimientos. Pueden usarse en un ámbito local o distribuido.
- ***Sockets*:** permiten el intercambio directo de datos a través de las funciones de red. Pueden usarse en un ámbito local o distribuido.

La memoria compartida constituye la alternativa más eficiente en cuanto a prestaciones de velocidad, pero no es directamente aplicable pues está restringida a un entorno local no distribuido. Las llamadas a procedimientos remotos permiten un intercambio de datos en un entorno distribuido, pero están especialmente concebidas para un uso síncrono, por lo que el módulo que realiza la llamada queda en espera hasta que recibe la respuesta del módulo que ejecuta la función. Debido al carácter distribuido y asíncrono de la arquitectura de control que se pretende desarrollar la alternativa más apropiada es el empleo de ***sockets***.

Así pues, la solución finalmente adoptada para la implementación del estilo de la nueva arquitectura de control se basa en la alternativa de **interacción entre procesos**, siguiendo un esquema de **interconexión indirecta** mediante un agente central y utilizando como mecanismo de intercambio de información los ***sockets***.

3.2 Diseño del estilo

Una vez seleccionada la alternativa más eficiente para el estilo de la arquitectura de control propuesta se va a proceder a su implementación. Para ello se partirá de un modelo básico que será progresivamente mejorado hasta obtener las prestaciones finales deseadas en la nueva arquitectura de control.

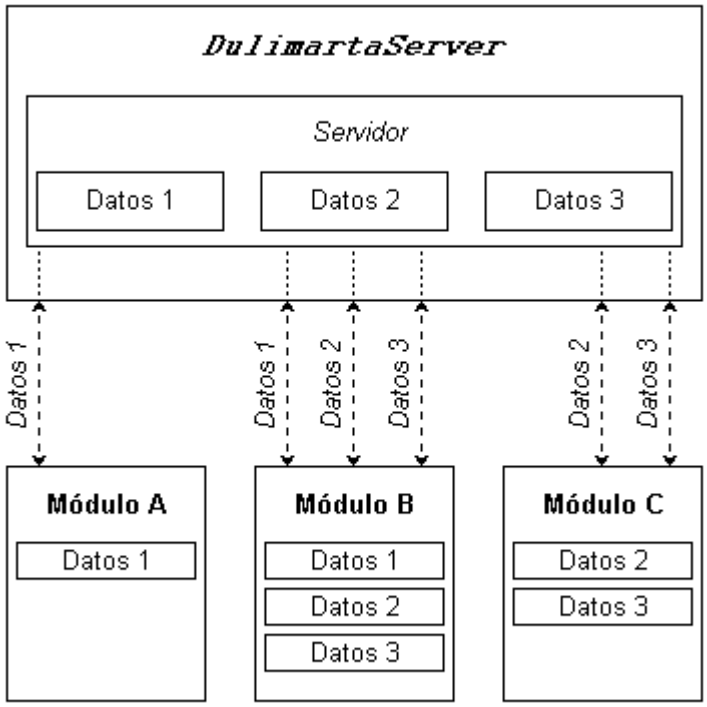

Figura 3.2: Estilo propuesto por Dulimarta.

3.2.1 Esquema distribuido básico

Dulimarta implementó un esquema cliente-servidor que permitía la interconexión indirecta entre distintos módulos distribuidos siguiendo un modo de operación asíncrono [DJ96], tal y como se representa en la figura 3.2. Este esquema desarrolla las características necesarias para el estilo de la arquitectura de control buscada (interacción entre procesos mediante interconexión indirecta utilizando *sockets*), por lo que va a constituir el punto de partida de la nueva arquitectura de control, adaptándolo convenientemente y mejorando sus prestaciones donde sea posible.

El núcleo de la arquitectura lo forma un servidor central, que es el encargado de coordinar el intercambio de información entre los distintos módulos del sistema. Este esquema implementa un sistema de **memoria compartida distribuida**, en el cual el servidor central se encarga de almacenar todos los datos que el resto de módulos quieren compartir entre sí. Así pues, cualquier módulo puede conectarse al servidor central y solicitar el uso de una zona de memoria para leer o escribir los datos almacenados en la misma, zonas de memoria conocidas como **conexiones**. Distintos módulos pueden enlazarse a la misma zona de memoria, por lo que la escritura de nuevos datos en una conexión es directamente accesible por los demás módulos enlazados a la misma. El servidor central es el encargado de gestionar las distintas conexiones creadas en el sistema, enlazando a cada una de ellas los módulos que lo soliciten y controlando el acceso exclusivo a sus respectivas zonas de memoria para evitar lecturas y escrituras simultáneas. Mediante este esquema la operación de los distintos módulos es totalmente asíncrona entre sí,

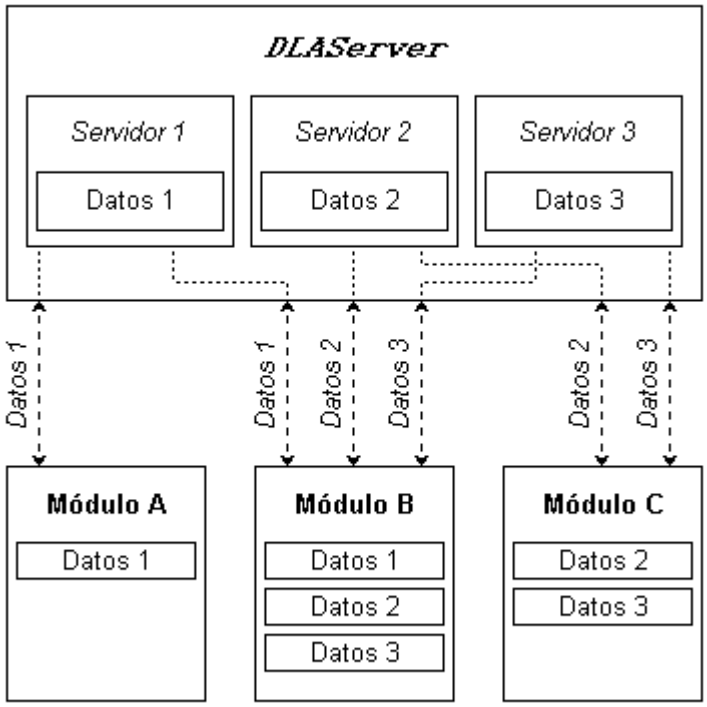

Figura 3.3: Estilo propuesto para la arquitectura de control *DLA*.

ya que son servidos en sus operaciones de intercambio de datos inmediatamente en cuanto lo soliciten, independientemente de la operación del resto de módulos.

El principal inconveniente en el esquema cliente-servidor propuesto por Dulimarta radica en el hecho de que todo el intercambio de información está gestionado por un único servidor. Así pues, si este intercambio de información se incrementa considerablemente el sistema puede llegar a congestionarse, aumentando excesivamente el tiempo de servicio ante las sucesivas peticiones de acceso a los datos. En un sistema donde coexisten conexiones grandes con un número elevado de datos y conexiones pequeñas con un número reducido de datos son las conexiones pequeñas las más perjudicadas, pues aunque son rápidas de procesar deben esperar largos períodos de tiempo mientras el servidor central está atendiendo las peticiones generadas por las conexiones grandes que están situadas delante en la cola del servidor. Este problema puede llegar a ser especialmente grave en los sistemas híbridos, puesto que las conexiones pequeñas están asociadas normalmente con los módulos reactivos, las cuales deben ser atendidas con bastante frecuencia y necesitan tiempos de servicio bajos. Por lo tanto, el resultado final es que un módulo deliberado puede bloquear la respuesta a un módulo reactivo durante una cantidad de tiempo considerable.

Para resolver este problema se ha propuesto un estilo de arquitectura alternativo basado en el esquema cliente-servidor de la figura 3.3. La diferencia principal con el esquema propuesto por Dulimarta radica en la incorporación de varios servidores específicos para la gestión del intercambio de información entre módulos en lugar de utilizar únicamente uno. Este nuevo esquema

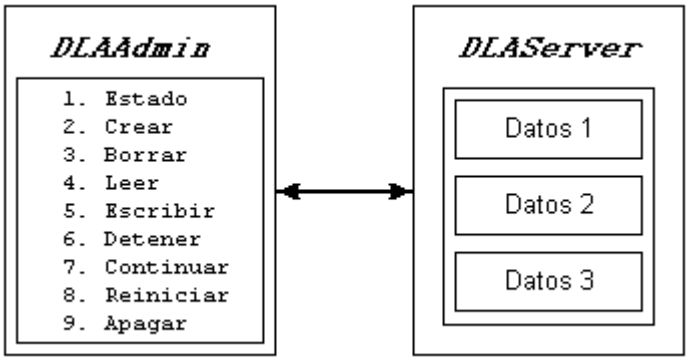

Figura 3.4: Administrador remoto *DLAAdmin* de la arquitectura *DLA*.

se ha denominado **esquema distribuido básico**, y posee un servidor central *DLAServer* que se encarga de gestionar de manera transparente la creación y destrucción de estos servidores específicos.

Cada uno de estos servidores específicos está destinado a atender exclusivamente a una de las conexiones que existen en el sistema, gestionando todos los accesos de los distintos módulos a los datos almacenados en dicha conexión. Así pues, en lugar de existir una cola general en el sistema para atender todas las peticiones de los módulos existen varias en función del tipo de información a intercambiar. Al estar en este nuevo esquema las conexiones grandes separadas de las conexiones pequeñas estas pueden ser atendidas simultáneamente sin que se corra el riesgo de que unas peticiones bloqueen a otras.

Este esquema es bastante más complejo de implementar, pues hay que gestionar la creación y destrucción de varios servidores específicos en el sistema, pero posee la evidente ventaja de que se reduce considerablemente el tiempo medio de servicio de las conexiones pequeñas. La arquitectura se ha diseñado para optimizar automáticamente el uso de los recursos disponibles, minimizando la memoria empleada y el uso del procesador. En este sentido, los servidores específicos detectan automáticamente la desconexión de los módulos conectados a la conexión asociada, destruyendo la conexión cuando no existen módulos conectados. El servidor central, por su parte, destruye automáticamente los servidores específicos cuando la conexión asociada es destruida y ya no son necesarios en el sistema.

Para facilitar la labor de desarrollo y depuración de los distintos módulos en el sistema se le ha añadido al servidor central *DLAServer* la posibilidad de monitorizar y modificar en tiempo real su operación. Para ello se ha desarrollado un administrador remoto denominado *DLAAdmin* [1], tal y como se muestra en la figura 3.4. Este administrador remoto permite acceder a todos los datos internos almacenados en el servidor central así como realizar determinadas tareas de administración. Además, se le ha dotado con la capacidad de crear conexiones permanentes en el sistema que no sean destruidas aunque no tengan ningún módulo conectado. Este tipo de conexiones permanentes se pueden utilizar para mantener la persistencia de algunos datos

[1] Una descripción detallada del uso del administrador remoto *DLAAdmin* se puede encontrar en el apartado 3.1.3 del apéndice A.

importantes en el sistema, como por ejemplo el mapa del entorno sobre el que el agente autónomo móvil desarrolla su actividad.

Por último, se han realizado algunas pruebas para comparar las prestaciones obtenidas por el esquema distribuido básico de la arquitectura *DLA* frente a las prestaciones obtenidas por el esquema propuesto por Dulimarta. Para ello se ha medido el tiempo medio de servicio de la arquitectura para una conexión pequeña ante distintas cargas de tráfico en el sistema. Puesto que el tráfico generado en el sistema es difícil de caracterizar a priori, se ha modelado el mismo usando algunas de las distribuciones clásicas y se han repetido las pruebas para cada una de ellas. En este sentido se han utilizado la distribución uniforme, la exponencial y la gaussiana, con un tiempo medio de 100 *ms* para todas las distribuciones y con una desviación estándar de 12.5 *ms* para la distribución gaussiana.

La tabla 3.1 muestra el tiempo medio de respuesta (medido en milisegundos) que emplea el sistema en atender una petición de intercambio de información de una conexión de 16 bytes, según el tráfico generado en el sistema (medido en bytes por segundo) se haya modelado como uniforme, exponencial o gaussiano. Los errores se han calculado sobre un intervalo de confianza del 95%. Según la tabla 3.1 el comportamiento del sistema es muy similar independientemente del tipo de distribución usado para modelar el tráfico, como era de esperar.

La prueba 1 se corresponde con el escenario donde únicamente existe una conexión de 16 bytes en el sistema, los cuales son periódicamente transmitidos según la distribución de tráfico usada en cada momento. Como indica la tabla 3.1 las diferencias entre la arquitectura *DLA* y el esquema propuesto por Dulimarta son prácticamente inapreciables. Esta situación es totalmente lógica, puesto que en estas circunstancias únicamente existe una conexión en el sistema y ambos esquemas son totalmente equivalentes.

La prueba 2 añade a la conexión básica de 16 bytes 4 conexiones más también de 16 bytes. En este caso comienzan a aparecer pequeñas diferencias entre ambos esquemas, si bien todavía no son muy apreciables debido a que el tráfico del sistema es aun bastante reducido.

Las pruebas 3 a la 6 muestran la respuesta de ambos esquemas cuando el tráfico generado aumenta considerablemente, añadiendo a la conexión básica de 16 bytes de 1 a 4 conexiones de 64 Kbytes respectivamente [2]. Se puede observar que las prestaciones del esquema distribuido básico de la arquitectura *DLA* son claramente superiores a las del esquema propuesto por Dulimarta cuando la carga de tráfico en el sistema es elevada, debido al diseño concurrente de la arquitectura *DLA*.

La figura 3.5 representa los resultados de las pruebas 3 a la 6 para las distintas distribuciones de tráfico, tanto para la arquitectura *DLA* como para el esquema propuesto por Dulimarta. Las pruebas 1 y 2 no han sido representadas debido a que las diferencias entre ambos esquemas no son muy significativas cuando la carga de tráfico del sistema no es muy elevada. El retardo medio se mide en milisegundos y el tráfico generado en megabytes por segundo. Según la figura

[2] Este tamaño de conexión ha sido escogido intencionadamente porque coincide con el tamaño de la conexión del mapa métrico usado en el sistema, por lo que se corresponde con una situación real.

Tráfico Uniforme	***DLA*** *Retardo* [*Tráfico*]	**Dulimarta** *Retardo* [*Tráfico*]
Prueba 1	16.52 ± 0.17 *ms* [275 Bps]	16.27 ± 0.19 *ms* [275 Bps]
Prueba 2	16.3 ± 0.4 *ms* [1.34 KBps]	20.5 ± 0.8 *ms* [1.30 KBps]
Prueba 3	24 ± 3 *ms* [0.89 MBps]	28 ± 4 *ms* [0.83 MBps]
Prueba 4	32 ± 5 *ms* [1.35 MBps]	52 ± 6 *ms* [1.19 MBps]
Prueba 5	40 ± 6 *ms* [1.71 MBps]	78 ± 9 *ms* [1.25 MBps]
Prueba 6	48 ± 6 *ms* [1.89 MBps]	122 ± 13 *ms* [1.38 MBps]
Tráfico Exponencial	***DLA*** *Retardo* [*Tráfico*]	**Dulimarta** *Retardo* [*Tráfico*]
Prueba 1	16.05 ± 0.17 *ms* [276 Bps]	16.68 ± 0.25 *ms* [274 Bps]
Prueba 2	17.6 ± 1.0 *ms* [1.33 KBps]	19.8 ± 0.7 *ms* [1.30 KBps]
Prueba 3	21.8 ± 2.3 *ms* [0.85 MBps]	25 ± 3 *ms* [0.84 MBps]
Prueba 4	32 ± 4 *ms* [1.34 MBps]	46 ± 5 *ms* [1.18 MBps]
Prueba 5	37 ± 6 *ms* [1.62 MBps]	72 ± 8 *ms* [1.28 MBps]
Prueba 6	51 ± 7 *ms* [1.85 MBps]	126 ± 13 *ms* [1.42 MBps]
Tráfico Gaussiano	***DLA*** *Retardo* [*Tráfico*]	**Dulimarta** *Retardo* [*Tráfico*]
Prueba 1	16.19 ± 0.17 *ms* [275 Bps]	16.29 ± 0.19 *ms* [275 Bps]
Prueba 2	16.4 ± 0.4 *ms* [1.34 KBps]	20.4 ± 0.7 *ms* [1.30 KBps]
Prueba 3	26 ± 3 *ms* [0.86 MBps]	24.0 ± 2.2 *ms* [0.88 MBps]
Prueba 4	33 ± 4 *ms* [1.39 MBps]	48 ± 5 *ms* [1.17 MBps]
Prueba 5	43 ± 6 *ms* [1.69 MBps]	78 ± 9 *ms* [1.26 MBps]
Prueba 6	50 ± 6 *ms* [1.90 MBps]	117 ± 13 *ms* [1.40 MBps]

Tabla 3.1: Comparativa entre el esquema distribuido básico de la arquitectura *DLA* y el esquema propuesto por Dulimarta.

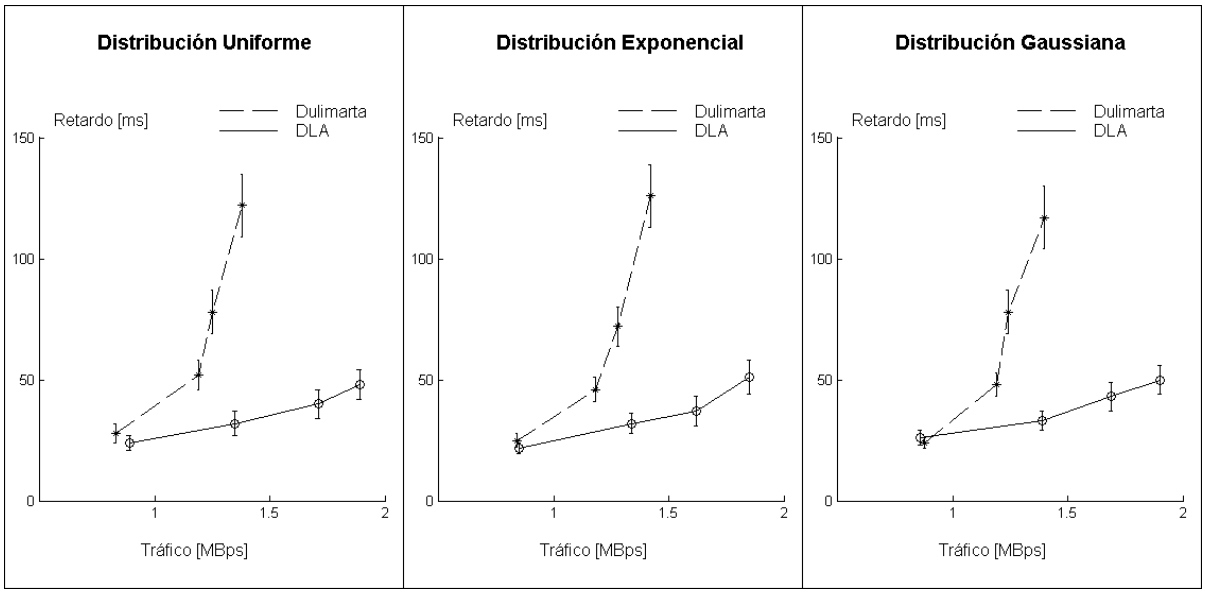

Figura 3.5: Comparativa entre el esquema distribuido básico de la arquitectura *DLA* y el esquema propuesto por Dulimarta.

3.5 la arquitectura *DLA* presenta unas prestaciones claramente superiores al esquema propuesto por Dulimarta.

3.2.2 Esquema distribuido síncrono

Una vez desarrollado el esquema distribuido básico de la arquitectura *DLA* aparece una nueva necesidad para optimizar el funcionamiento del sistema: la **sincronización**. Si bien es cierto que el modo de operación de la arquitectura de control *DLA* es asíncrono como se analizó en el apartado 2, es conveniente poseer algún mecanismo de sincronización entre módulos, puesto que en determinadas circunstancias este mecanismo puede ser necesario para optimizar el funcionamiento del sistema. De hecho, el esquema originalmente propuesto por Dulimarta posee un mecanismo de sincronización distribuido: los semáforos. El principal inconveniente de los semáforos radica en la dificultad de uso de los mismos, pues son mecanismos de bajo nivel que necesitan un análisis detallado de su uso para no correr el riesgo de caer en estados de bloqueo, conocidos como *deadlocks* [Ste90].

Para incorporar esta nueva funcionalidad en la arquitectura *DLA* se ha implementando un nuevo esquema denominado **esquema distribuido síncrono**, el cual posee un mecanismo de sincronización abstracto muy sencillo de utilizar: los **buzones** (*mailboxes*). Este mecanismo está basado en el uso de los semáforos, pero encapsula la dificultad de uso de los mismos en una estructura de alto nivel distribuida que permite realizar operaciones de sincronización entre módulos de forma muy sencilla, disminuyendo considerablemente la necesidad de un análisis detallado para evitar los posibles estados de bloqueo en el sistema.

Básicamente, un buzón es un contenedor de datos que puede almacenar cualquier valor simple de tipo entero. Cualquier módulo puede leer y escribir en cualquier instante sobre cualquier buzón que se defina en el sistema, manteniendo la capacidad de acceso distribuido que poseen las conexiones. La principal diferencia entre las conexiones y los buzones, además del tamaño prefijado de los datos que tienen los buzones, radica en que un módulo puede bloquearse en un buzón hasta que lleguen datos nuevos al mismo. En este estado de bloqueo, el módulo pasa a un estado inactivo, por lo que no consume tiempo de ejecución del procesador asignado ni genera tráfico a gestionar por el servidor de la arquitectura *DLAServer*, no consumiendo recursos del sistema y no interfiriendo pues con la ejecución del resto de módulos que estén procesando datos. Cuando cualquier módulo escribe datos en un buzón todos los módulos que estaban en el estado de bloqueo sobre dicho buzón a la espera de nuevos datos son automáticamente activados de forma transparente, por lo que pueden continuar con el procesamiento de datos para el que fueron diseñados.

La utilización de los buzones mejora la funcionalidad y las prestaciones de la arquitectura de control *DLA*, pues permite llevar a cabo ciertas operaciones muy interesantes desde el punto de vista de la eficiencia del sistema:

- **Sincronización entre módulos:** expresamente diseñados con tal fin, los buzones permiten la sincronización entre módulos para la realización conjunta de algún tipo de proce-

samiento. Para realizar tal operación, basta con que los distintos módulos compartan uno o más buzones entre sí, de forma que algunos módulos se bloqueen en algún buzón común hasta que otros módulos escriban en el mismo. Esto permite, por ejemplo, que módulos que necesitan los datos de otros módulos se bloqueen hasta que estos datos estén disponibles.

- **Activación y desactivación de módulos:** los buzones permiten que únicamente estén activos en el sistema aquellos módulos que necesiten realizar un procesamiento de los datos de entrada, pasando el resto de módulos a un estado inactivo. Para realizar tal operación, basta con que cada módulo se bloquee en un buzón, el cual será activado únicamente cuando se necesite que el módulo en cuestión realize el procesamiento para el cual fue diseñado. Esto permite, por ejemplo, optimizar los recursos del sistema de forma que los módulos que no necesiten procesar datos no consuman recursos innecesariamente.

- **Distribución dinámica de módulos:** los buzones permiten la distribución dinámica de módulos a través de las máquinas destinadas a la implementación del sistema, con el fin de optimizar la utilización de los recursos disponibles. Para realizar tal operación, basta con lanzar varias copias del mismo módulo en distintas máquinas, bloqueando cada una de estas copias en un buzón. Con la escritura en el buzón adecuado se puede activar selectivamente el módulo deseado, que puede estar ubicado en cualquier máquina. Si la carga de dicha máquina aumenta considerablemente se puede bloquear dicho módulo y escribir en otro buzón que active selectivamente el mismo módulo residente en otra máquina. Gracias al carácter distribuido de la arquitectura *DLA* no es necesario realizar ningún cambio en ningún módulo para reubicarlo en otra máquina, pues el proceso es totalmente transparente para el usuario. Esto permite, por ejemplo, implementar un sistema de control automático de la carga del sistema para distribuir dinámicamente los módulos a través de los recursos disponibles en cada momento, de forma que se aumente la eficiencia en el sistema.

Sin el esquema distribuido síncrono, donde no es posible la activación y desactivación selectiva de módulos, todos los módulos deben estar activos ininterrumpidamente. Esto implica que si en un instante determinado un módulo no debe realizar ningún tipo de procesamiento, debe entrar en un bucle cerrado a la espera de que su procesamiento sea de utilidad para el sistema. Estos bucles cerrados obviamente consumen recursos, ya que el procesador en el cual esté asignado dicho módulo deberá reservar tiempo de ejecución para el mismo, y además el módulo realiza continuamente peticiones de lectura de distintas conexiones que aumentan el tráfico del sistema. Esta filosofía es claramente ineficiente, por lo que un esquema síncrono es mucho más eficiente aun en el caso de que no sea necesaria la sincronización de los módulos entre sí. Es importante notar que la funcionalidad síncrona es una facilidad que se suministra con la arquitectura *DLA*, no una imposición a ser utilizada, persiguiendo siempre como objetivo prioritario en su desarrollo la flexibilidad y la disminución de las posibles restricciones en su uso.

Para la gestión de todos los buzones del sistema se ha creado un nuevo servidor específico, el cual es automáticamente creado y destruido por el servidor central de la arquitectura *DLAServer* cuando es necesario. No se ha implementado un servidor específico para cada buzón como ocurre con las conexiones debido a que el tamaño de los datos a intercambiar mediante el uso de los buzones ha sido predefinido al tipo simple entero, que normalmente posee 4 bytes. Así pues, la

gestión de un buzón implica un muy reducido intercambio de datos con el servidor, por lo que el tráfico generado por los distintos buzones del sistema es reducido y puede ser perfectamente gestionado mediante un único servidor sin aparecer problemas de congestión.

Por último, indicar que se ha aumentado la funcionalidad del administrador remoto *DLAAdmin* para la correcta gestión y manipulación de los datos almacenados en los buzones, así como la creación de buzones permanentes en el sistema que no sean destruidos aunque no tengan ningún módulo conectado, posibilitando la persistencia de algunos datos en el sistema.

3.2.3 Esquema local síncrono

Los esquemas distribuidos que se han desarrollado para el estilo de la arquitectura de control *DLA* posibilitan la libre distribución de los módulos entre varias máquinas. Sin embargo, bajo determinadas circunstancias estos esquemas distribuidos pueden resultar claramente ineficientes en la operación del sistema, debido a que los recursos de los que se dispone suelen ser limitados. De hecho, lo más usual es que exista un mayor número de módulos que de máquinas disponibles, por lo que distintos módulos suelen residir en una misma máquina. Si además algunos módulos de los que residen en la misma máquina sólo interaccionan entre ellos mismos, no intercambiando datos con ningún otro módulo que resida fuera de la máquina en cuestión, es claramente ineficiente interconectarlos entre sí a través de los esquemas distribuidos, que requieren la conexión mediante *sockets* con el agente central externo para intercambiar datos.

En estas circunstancias donde los módulos que tienen que intercambiar datos entre sí residen en la misma máquina es mucho más eficiente utilizar otro tipo de mecanismo para el intercambio de datos entre procesos no basado en *sockets*. Según se analizó en el apartado 3.1, la alternativa más eficiente en este caso la constituye el uso de la **memoria compartida** [Ste90]. De hecho, la arquitectura de control *DLA* implementa un modelo de memoria compartida distribuida, por lo que es más que evidente que si los módulos que necesitan intercambiar datos entre sí no están distribuidos sino ubicados en la misma máquina este modelo de memoria compartida distribuida se puede reducir a un simple modelo de memoria compartida local.

Al igual que se realizó con los buzones, se ha implementado otra funcionalidad que permite optimizar más aun el funcionamiento del sistema bajo estas circunstancias. Este nuevo esquema se ha denominado **esquema local síncrono**, pues hace referencia a la situación donde distintos módulos se ubican localmente en la misma máquina en lugar de distribuirse entre varias de ellas. En este caso no es necesario la utilización del servidor central de la arquitectura *DLAServer*, sino que los módulos residentes en la misma máquina se conectan directamente entre sí a través de memoria compartida. Esta es una optimización importante que no se ha encontrado en ninguna de las arquitecturas de control analizadas, incluyendo por supuesto el esquema originalmente propuesto por Dulimarta. Este mecanismo local se ha implementado tanto para las conexiones como para los buzones, por lo que también es posible utilizar mecanismos de sincronización con este nuevo esquema. Por supuesto, al igual que ocurre con los buzones, la funcionalidad local es una facilidad que se suministra con la arquitectura *DLA*, no una imposición a ser usada, persiguiendo siempre como objetivo prioritario la flexibilidad y la disminución de las restricciones

en el uso de la misma.

El uso de la memoria compartida es bastante complejo, pues implica la utilización de otro tipo de mecanismos como los semáforos para garantizar la exclusión mutua en el acceso a los datos por parte de los distintos módulos. Todo ello se ha desarrollado de forma totalmente transparente al usuario, el cual únicamente debe indicar a priori si desea que una conexión o un buzón determinado utilice un esquema distribuido o un esquema local, encargándose el sistema de gestionarlos adecuadamente. Obviamente se pueden mezclar conexiones y buzones distribuidos y locales en el mismo sistema, permitiendo una optimización total en el uso de los recursos del mismo.

La utilización del esquema local síncrono permite optimizar la eficiencia en el uso de la arquitectura de control *DLA*, manifestando importantes características como:

- **Uso de recursos:** expresamente diseñado con tal fin, el esquema local permite utilizar los recursos más óptimos y eficientes donde sea posible. Además de la considerable reducción en el tiempo de acceso a los datos que implica el esquema local frente al distribuido, el uso del esquema local reduce el tráfico generado en el sistema, por lo que el servidor central de la arquitectura *DLAServer* disminuirá el tiempo de servicio hacia el resto de módulos, redundando en un beneficio global para todos los intercambios de datos en el sistema y aumentando considerablemente la eficiencia del mismo.
- **Implementación totalmente local:** en determinadas circunstancias todos los módulos del sistema deben ser ejecutados en la misma máquina, por lo que un esquema distribuido resulta claramente ineficiente. Bajo estas condiciones el esquema local síncrono es el mejor que puede ser implementado, derivando en un aumento de prestaciones en el sistema claramente significativo. Este escenario es muy común cuando se posee un agente autónomo en exteriores, donde suele ser complejo disponer de una infraestructura de red con varias máquinas. En estas circunstancias todos los módulos suelen ser ubicados en la máquina del propio agente autónomo, por lo que la alternativa local es la más óptima que se puede utilizar.

Por último, indicar que se ha aumentado la funcionalidad del administrador remoto *DLAAdmin* para la correcta gestión y manipulación de los datos internos almacenados en las conexiones y buzones locales, así como la creación de conexiones y buzones locales permanentes en el sistema que no sean destruidos aunque no tengan ningún módulo conectado, posibilitando la persistencia de algunos datos en el sistema aunque se desactiven los módulos del mismo.

4 Estructura de la arquitectura *DLA*

Una vez concluido el diseño del estilo de la arquitectura de control *DLA* es necesario abordar el diseño de su estructura, es decir, la descomposición del sistema en módulos que interaccionen entre sí para implementar el comportamiento final deseado. Esta interacción entre módulos se

llevará a cabo a través de los mecanismos de comunicación suministrados por el estilo de la arquitectura de control, que como se ha analizado en el apartado 3 consisten en el empleo de conexiones y buzones para el intercambio de datos y la sincronización entre los distintos módulos, respectivamente.

Es importante resaltar que si bien el estilo de la arquitectura *DLA* constituye un esquema genérico que puede ser aplicado a cualquier sistema cooperativo mediante la interacción de procesos libremente distribuidos, su estructura depende obviamente de la aplicación final que se quiera implementar, que será la que determine el conjunto de módulos necesarios a tal fin. Recuérdese que tal y como se analizó en el apartado 3 del capítulo 1, el objetivo del trabajo realizado en la presente Tesis consiste en el desarrollo de una infraestructura básica que permita incorporar la capacidad de navegación en entornos dinámicos no estructurados a un autónomo móvil. Para ello, se pretende implementar gradualmente distintos niveles de la Jerarquía de Navegación, hasta alcanzar los niveles superiores capaces de manifiestar los comportamientos de navegación más complejos y versátiles.

Así pues, la descomposición que se pretende realizar del sistema estará formada por el conjunto de módulos necesarios para implementar dicha infraestructura básica, dotada con la capacidad de navegación exhibida por el nivel superior de la Jerarquía de Navegación.

4.1 Niveles de navegación

La mejor aproximación para implementar los niveles superiores de la Jerarquía de Navegación consiste en un desarrollo gradual, por lo que partiendo de niveles de navegación más básicos y sencillos se van a añadir nuevas capacidades en el sistema para conseguir niveles de navegación cada vez más complejos. Esta estrategia presenta importantes ventajas desde el punto de vista funcional:

- El agente puede mostrar en cada momento las capacidades de navegación más adecuadas en función de las circunstancias particulares del entorno y de los recursos disponibles en el sistema. Así por ejemplo, se pueden desarrollar comportamientos básicos sin planificación si se carece de representaciones del entorno.
- Se independiza el proceso de diseño de los distintos niveles, permitiendo la ampliación o modificación de cualquier nivel sin necesidad de modificar el resto. Así por ejemplo, se pueden alterar los comportamientos de planificación sin modificar los comportamientos básicos de navegación.

Con esta perspectiva de implementación gradual del sistema, los niveles de navegación que se van a desarrollar y su correspondencia con los niveles descritos en la Jerarquía de Navegación propuesta por Franz y Mallot son los que se describen a continuación (figura 1.8 del capítulo 1):

- **Navegación Reactiva:** es el nivel de navegación más básico, y pretende alcanzar el destino final evitando las posibles colisiones con los obstáculos fijos y/o móviles del entorno.

No utiliza ningún tipo de representación del entorno, por lo que no realiza ningún proceso de planificación. Al no utilizar planificación posee tiempos de respuesta reducidos, por lo que constituye una alternativa idónea para implementar algoritmos rápidos de evitación de obstáculos. Sin embargo, la ausencia de planificación puede derivar en recorridos no eficientes, e incluso no permitir que se alcance el destino final al quedar el agente atrapado en trampas de mínimos locales. Este nivel de navegación se corresponde con el grupo de Navegación Local de la Jerarquía de Navegación, el cual desarrolla los niveles más básicos de navegación: Búsqueda, Seguimiento, Apuntado y Guiado.

- **Navegación Planificada:** sobre el nivel básico de Navegación Reactiva se incorpora una representación del entorno y un proceso de planificación del camino a recorrer en dicho entorno. Gracias al proceso de planificación se añade un mayor grado de eficiencia al optimizar bajo algún criterio el camino a seguir por el agente, permitiendo evitar las trampas de mínimos locales mientras el agente sigue evitando las colisiones con los obstáculos fijos y/o móviles del entorno gracias al nivel de Navegación Reactiva. Este nivel de navegación se corresponde con el nivel de Respuesta Inducida por Reconocimiento del grupo de Búsqueda de Caminos de la Jerarquía de Navegación.

- **Navegación Topológica:** sobre el nivel de Navegación Planificada se incorpora una nueva representación del entorno y un proceso de planificación de la ruta a seguir en dicho entorno, determinando para ello las principales regiones a atravesar para alcanzar el destino final. Este tipo de planificación es apropiada para grandes entornos, y permite ampliar las capacidades de navegación al posibilitar el establecimiento de rutas que atraviesen determinadas regiones. Una vez determinada la ruta para alcanzar el destino final se utiliza el nivel de Navegación Planificada para calcular el camino a seguir entre regiones sucesivas, el cual emplea a su vez el nivel de Navegación Reactiva para navegar evitando la colisión con los obstáculos fijos y/o móviles del entorno. Es importante remarcar que en la nueva representación utilizada se deber representar tanto las regiones exploradas como las regiones no exploradas del entorno, para posibilitar la planificación de rutas a través de zonas desconocidas del entorno. Este nivel de navegación se corresponde con los niveles de Navegación Topológica y Navegación por Reconocimiento del grupo de Búsqueda de Caminos de la Jerarquía de Navegación.

4.2 Descomposición del sistema

Al abordar la descomposición del sistema se suelen agrupar los distintos módulos resultantes en lo que se conoce como **capas**. Una capa es una entidad abstracta responsable de desarrollar un comportamiento determinado, y está compuesta por todos aquellos módulos del sistema que contribuyen en la implementación de dicho comportamiento. Esta descomposición en capas añade una mayor modularidad en el desarrollo del sistema, ya que se pueden ir incorporando nuevas capas progresivamente para añadir nuevos comportamientos sobre los ya existentes.

Siguiendo esta filosofía, los distintos módulos obtenidos durante la descomposición del sistema se van a agrupar en capas, de forma que se facilite la implementación gradual del mismo.

4.2.1 Módulos y capas

Como se analizó en el apartado 2.1 del capítulo 2 una descomposición híbrida que combine comportamientos reactivos y deliberados es la aproximación más eficiente hoy en día para el desarrollo de agentes autónomos móviles, por lo que será el modelo de descomposición propuesto por la arquitectura de control que se está desarrollando. En todo esquema híbrido se pueden identificar los siguientes tipos de comportamientos:

- **Reactivos:** comportamientos caracterizados por la ausencia de planificación y por proporcionar respuestas rápidas ante los distintos estímulos del entorno, contituyendo una alternativa idónea para implementar esquemas de evitación de obstáculos.
- **Deliberados:** comportamientos temporalmente costosos caracterizados por llevar a cabo un proceso de planificación sobre la representación del entorno que posee el agente, aumentando la eficiencia en la operación del sistema.

Así pues, la descomposición realizada del sistema debe incorporar los módulos necesarios que integren estos comportamientos, más todos aquellos módulos adicionales que se necesiten para su correcta operación. Con tal fin se ha descompuesto el sistema en las siguientes **capas** y **módulos** (figura 3.6):

- **Capa Física:** desarrolla el comportamiento de control que permite acceder a la plataforma robótica para enviarle comandos de movimiento y recibir la información captada del entorno por la misma. Está integrada por los siguientes módulos:
  - ***IFRobot*:** accede físicamente a la plataforma robótcia, compartiendo la información recibida de la misma con el resto de módulos del sistema y enviándole los comandos de movimiento adecuados. Además implementa funciones importantes como una adecuada traducción del sistema de coordenadas que use la plataforma robótica al sistema de coordenadas que usen los módulos del sistema, y un sistema de alarma que detiene la plataforma robótica inmediatamente ante potenciales situaciones de colisión.
- **Capa de Navegación:** desarrolla el comportamiento de navegación reactiva que permite navegar en entornos dinámicos no estructurados sin colisionar con los posibles obstáculos fijos y/o móviles. Está compuesta por los siguientes módulos:
  - ***Navigation*:** implementa los algoritmos de evitación de obstáculos que permiten al agente la navegación por el entorno. Se pueden implementar distintos esquemas de navegación, de forma que se escoja en cada situación aquel más adecuado a las características de navegación deseadas. En concreto este módulo tiene implementados un esquema de navegación reactiva basado en el paradigma de los Campos Potenciales y un esquema de navegación reactiva basado en el paradigma del razonamiento basado en casos, siendo posible alternar entre uno y otro según se desee.

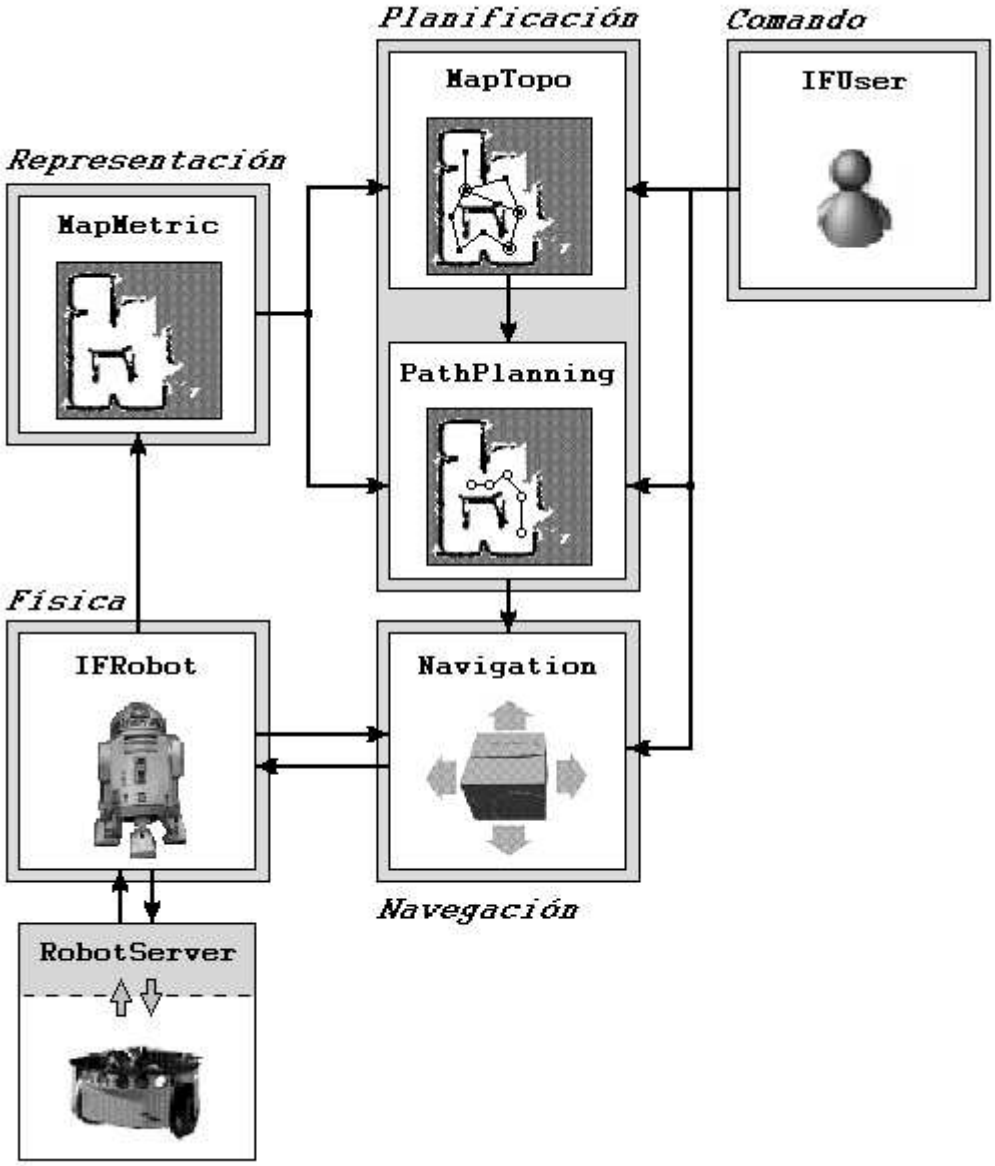

Figura 3.6: Descomposición propuesta del sistema en capas y módulos.

- **Capa de Representación:** desarrolla el comportamiento de modelado del entorno por el que navega el agente autónomo móvil, necesario para llevar a cabo los posteriores procesos de planificación. La fidelidad del modelo respecto a la situación actual del entorno delimita la validez del proceso de planificación, por lo que el modelo debe ser continuamente actualizado si se desea obtener una planificación adecuada. Está formada por los siguientes módulos:

  - ***MapMetric*:** implementa el algoritmo de construcción de una representación métrica del entorno por el que navega el agente. En concreto este módulo genera una representación métrica del entorno basada en mapas probabilísticos.

- **Capa de Planificación:** desarrolla el comportamiento de planificación de caminos y rutas que el agente autónomo móvil debe seguir para alcanzar el destino final en el entorno donde navega. Está integrada por los siguientes módulos:

  - ***PathPlanning*:** implementa los algoritmos para realizar el cálculo de caminos libres de obstáculos que el agente debe seguir para alcanzar el destino final, así como la descomposición del camino propuesto en una serie de destinos intermedios al que se irá dirigiendo sucesivamente el sistema reactivo. Se pueden implementar distintos esquemas de planificación, de forma que se escoja en cada situación aquel más adecuado a las características de navegación deseadas. En concreto este módulo tiene implementado un esquema de cálculo de caminos basado en el paradigma de los Campos Potenciales.
  - ***MapTopo*:** implementa los algoritmos de construcción de una representación topológica a partir del mapa métrico del entorno por el que navega el agente, así como el cálculo de la ruta de las sucesivas regiones que tiene que ir atravesando el agente para alcanzar el destino final. Se pueden implementar distintos esquemas de planificación, de forma que se escoja en cada situación aquel más adecuado a las características del entorno. En concreto este módulo genera una representación topológica de tipo jerárquico y tiene implementado un esquema de planificación basado en la algoritmo de Dijkstra.

- **Capa de Comando:** desarrolla el comportamiento de control que permite enviar comandos de navegación al agente para dirigirlo hacia determinadas localizaciones según el nivel de navegación deseado. Está integrada por los siguientes módulos:

  - ***IFUser*:** implementa el interfaz de usuario para controlar el agente, de forma que este pueda enviar comandos al robot y recibir la información del mismo. Actualmente se ha desarrollado una aplicación visual que permite monitorizar el estado del robot y enviarle comandos de navegación de forma sencilla.

Es importante notar que aunque en la figura 3.6 se muestre el conexionado directo entre los distintos módulos del sistema, esta interacción se lleva a cabo mediante la arquitectura de control *DLA* a través del servidor central *DLAServer*.

La Capa de Navegación desarrolla los comportamientos reactivos propios de los sistemas híbridos, implementando el nivel de Navegación Reactiva descrito en el apartado 4.1. Por su parte, la Capa de Planificación desarrolla los comportamientos deliberados de planificación del sistema, implementando los niveles de Navegación Planificada y Navegación Topológica descritos en el apartado 4.1. En la descomposición realizada no se ha implementado explícitamente la capa intermedia propia de los sistemas híbridos que actúa de interfaz entre ambas capas, pues la necesaria adaptación entre planificación y actuación se ha distribuido a lo largo de las mismas. El resto de capas que aparecen en el proceso de descomposición son necesarias para la correcta operación del sistema.

4.2.2 Servidor hardware

En la figura 3.6 se aprecia la existencia en el agente de un servidor denominado *RobotServer*. Este servidor controla y gestiona el acceso al hardware de la plataforma robótica, estando también presente en la arquitectura propuesta por Dulimarta [DJ96]. Se pueden desarrollar distintos servidores hardware para distintas plataformas robóticas, presentando todos ellos un mismo conjunto de comandos estándar para su control. De este modo, independientemente de la plataforma robótica que se esté utilizando, los comandos que la controlan son siempre los mismos.

El conjunto de comandos estándar que se ha definido para controlar cualquier plataforma robótica mediante el correspondiente servidor hardware *RobotServer* se ha implementado en una librería conocida como *RobotLibrary*, que pueden utilizar todos aquellos módulos que deseen conectarse con la plataforma robótica [3]. Esta librería ha sido desarrollada en *C* para *Linux* y para *Windows*, por lo que cualquier módulo que utilice dicho lenguaje o plataformas puede conectarse con la plataforma robótica utilizada.

El servidor hardware *RobotServer*, pues, es el encargado de traducir este conjunto de comandos estándar a los comandos específicos de la plataforma robótica utilizada. Gracias a su utilización se puede ampliar fácilmente el equipamiento básico de la plataforma robótica, incorporando nuevos sensores o actuadores que serán adecuadamente gestionados por los comandos estándar definidos, sin necesidad de modificar los módulos que utilicen la plataforma robótica. Es necesario indicar que, obviamente, no todas los comandos estándar definidos pueden estar disponibles en todas las plataformas robóticas, en cuyo caso el comando correspondiente no tendrá ningún efecto sobre la plataforma robótica particular que se esté utilizando.

Por último, indicar que únicamente se permite la conexión de un módulo con cualquier servidor hardware *RobotServer*, garantizando así el control de acceso exclusivo a los recursos físicos del agente. Sin embargo, se pueden implementar distintos servidores hardware para distintos recursos, como por ejemplo un servidor hardware para controlar un sistema de cámaras independientemente del control que se realice sobre la plataforma robótica. Así pues, se pueden desarrollar módulos especializados en controlar los distintos recursos hardware de forma concurrente a través de los correspondientes servidores hardware.

4.3 Operación del sistema

En este apartado se va a analizar desde un punto de vista cualitativo la operación del sistema según la descomposición realizada del mismo. Para ello se va a describir la funcionalidad de cada uno de los módulos y su interacción con el resto de módulos del sistema.

El objetivo final del sistema es alcanzar el destino final de forma segura y eficiente, según el nivel de navegación deseado: Navegación Reactiva, Navegación Planificada o Navegación Topológica.

[3] Una descripción detallada del uso de las funciones contenidas en la librería *RobotLibrary* se puede encontrar en el apartado 3.2.2 del apéndice A.

El nivel de Navegación Topológica se encarga de calcular una ruta hasta el destino final, el nivel de Navegación Planificada se encarga de calcular un camino libre de obstáculos para alcanzar la siguiente región de la ruta calculada, y el nivel de Navegación Reactiva se encarga de seguir el camino calculado evitando los posibles obstáculos fijos y/o móviles del entorno.

El camino calculado por el nivel de Navegación Planificada, sin embargo, rara vez puede ejecutarse exactamente, ya que el entorno es dinámico y suelen aparecer nuevos obstáculos a lo largo del mismo. Mejor que recalcular un nuevo camino cada vez que el agente se desvía del original, el camino propuesto es seguido de forma parcial. Para ello, se descompone el camino a seguir en una serie de destinos intermedios que marcan los puntos de inflexión donde se produce un cambio significativo en la curvatura del camino. Uniendo estos destinos intermedios, pues, se puede recorrer aproximadamente el camino propuesto, por lo que la Navegación Reactiva tratará de alcanzar en línea recta cada uno de estos destinos intermedios consecutivamente, evitando los posibles obstáculos que aparezcan en el entorno. Así pues, el efecto producido es que el agente es atraído por el camino propuesto y repelido de los obstáculos circundantes, aunque no es imprescindible que recorra el camino tal y como ha sido propuesto. Recorrer el camino propuesto exactamente no debe ser un objetivo, pues el camino únicamente sirve para alcanzar el destino final evitando las trampas de mínimos locales.

En algunos casos, no obstante, la aparición de numerosos obstáculos o la mayor exploración del entorno pueden modificar considerablemente la estructura del mismo, provocando que la ruta y/o el camino calculados no sean adecuados. Así pues, cuando se detectan cambios significativos en la representación del entorno mientras se navega se vuelven a recalcular automáticamente una nueva ruta y/o camino para alcanzar el destino final. Mientras se está procediendo a calcular la nueva ruta y/o el nuevo camino, el agente sigue moviéndose reactivamente para alcanzar el siguiente destino intermedio del antiguo camino calculado, el cual, aunque anticuado y no muy eficiente, suele dirigir al agente hacia el destino final. Así pues, el agente no se detiene durante el tiempo en el cual se procesa la nueva ruta y/o camino.

Siguiendo esta filosofía se puede observar que el agente es capaz de reaccionar rápidamente ante la aparición de obstáculos dinámicos e inesperados en la trayectoria gracias a la Navegación Reactiva, mientras que la ruta y el camino calculados proporcionan un mecanismo eficiente y libre de mínimos locales para alcanzar el destino final, gracias a la Navegación Topológica y a la Navegación Planificada.

Así pues, según la filosofía de operación anteriormente descrita el comportamiento de todos y cada uno de los módulos del sistema es la que se describe a continuación:

1. ***IFUser*:** todos los módulos del sistema se encuentran inactivos excepto el módulo *IFUser* que implementa el interfaz de usuario, que se encuentra a la espera de la recepción de nuevos comandos de navegación por parte del usuario. Es interesante recordar que existen distintos comandos de navegación, según se quieran activar los distintos niveles de navegación. Cuando se recibe un nuevo comando de navegación mediante la especificación de un destino final, se activan todos los módulos necesarios para desarrollar dicho comando, y se comparte el destino final con el resto de módulos del sistema.

2. ***IFRobot*:** el módulo *IFRobot* es activado, accediendo continuamente a la información sensorial de la plataforma robótica, consistente principalmente en las lecturas de los sensores sonar y en la posición odométrica interna de la plataforma. Esta información es compartida con el resto de módulos del sistema.

3. ***MapMetric*:** el módulo *MapMetric* es activado, accediendo continuamente a la información sensorial de la plataforma robótica compartida por el módulo *IFRobot* y generando un mapa que contenga una representación métrica del entorno. Esta representación métrica es así mismo compartida con el resto de módulos del sistema. Si la representación métrica del entorno cambia significativamente, además, este módulo se encarga de activar los módulos de la Capa de Planificación adecuados para recalcular la ruta y/o el camino necesarios para alcanzar el destino final.

4. ***MapTopo*:**

   - **Navegación Reactiva:** si el comando enviado por el usuario corresponde a un nivel de Navegación Reactiva, el módulo *MapTopo* queda inactivo.
   - **Navegación Planificada:** si el comando enviado por el usuario corresponde a un nivel de Navegación Planificada, el módulo *MapTopo* queda inactivo.
   - **Navegación Topológica:** si el comando enviado por el usuario corresponde a un nivel de Navegación Topológica, el módulo *MapTopo* es activado, accediendo al mapa métrico del entorno compartido por el módulo *MapMetric* y generando una representación topológica del entorno con las principales regiones del mismo. Sobre la representación topológica generada se calculan las regiones de paso necesarias a atravesar para alcanzar el destino final compartido por el módulo *IFUser* de forma eficiente. La siguiente región de la ruta calculada es compartida con el resto de módulos del sistema. Tras realizar el cálculo de la ruta el módulo se desactiva a la espera de tener que recalcular otra ruta.

5. ***PathPlanning*:**

   - **Navegación Reactiva:** si el comando enviado por el usuario corresponde a un nivel de Navegación Reactiva, el módulo *PathPlanning* queda inactivo.
   - **Navegación Planificada:** si el comando enviado por el usuario corresponde a un nivel de Navegación Planificada, el módulo *PathPlanning* es activado, accediendo al destino final especificado por el usuario compartido por el módulo *IFUser* y calculando el camino libre de obstáculos a seguir para alcanzar dicho destino. En este cálculo del camino se utiliza el mapa métrico completo compartido por el módulo *MapMetric*. Una vez calculado el camino se extraen los puntos de inflexión del mismo y se obtienen los destinos intermedios necesarios para recorrerlo. El siguiente destino intermedio del camino es compartido con el resto de módulos del sistema. Tras realizar el cálculo del camino el módulo se desactiva a la espera de tener que recalcular otro camino.
   - **Navegación Topológica:** si el comando enviado por el usuario corresponde a un nivel de Navegación Topológica, el módulo *PathPlanning* es activado, accediendo a la

siguiente región de la ruta calculada a visitar compartida por el módulo *Maptopo* y calculando el camino libre de obstáculos a seguir para alcanzar dicha región. En este cálculo se utiliza únicamente la porción del mapa métrico compartido por el módulo *MapMetric* que contiene a la región actual y a la siguiente región de la ruta, con el fin de acelerar la velocidad en el cálculo del camino. Una vez calculado el camino se extraen los puntos de inflexión del mismo y se obtienen los destinos intermedios necesarios para recorrerlo. El siguiente destino intermedio del camino es compartido con el resto de módulos del sistema. Tras realizar el cálculo del camino el módulo se desactiva a la espera de tener que recalcular otro camino.

6. ***Navigation*:**

   - **Navegación Reactiva:** si el comando enviado por el usuario corresponde a un nivel de Navegación Reactiva, el módulo *Navigation* es activado, accediendo al destino final compartido por el módulo *IFUser* y calculando los comandos de movimiento adecuados para alcanzar dicho destino final. En el cálculo de los comandos de movimiento adecuados el sistema siempre trata de alcanzar el destino en línea recta, y cuando los obstáculos del entorno se encuentran demasiado cercanos al agente se activa el esquema de evitación de obstáculos implementado. Los comandos de movimiento para la plataforma robótica son compartidos con el resto de módulos.
   - **Navegación Planificada:** si el comando enviado por el usuario corresponde a un nivel de Navegación Planificada, el módulo *Navigation* es activado, accediendo al destino intermedio compartido por el módulo *PathPlanning* y calculando los comandos de movimiento adecuados para alcanzar dicho destino intermedio. En el cálculo de los comandos de movimiento adecuados el sistema siempre trata de alcanzar el destino en línea recta, y cuando los obstáculos del entorno se encuentran demasiado cercanos al agente se activa el esquema de evitación de obstáculos implementado. Los comandos de movimiento para la plataforma robótica son compartidos con el resto de módulos.
   - **Navegación Topológica:** si el comando enviado por el usuario corresponde a un nivel de Navegación Topológica, el módulo *Navigation* es activado, accediendo al destino intermedio compartido por el módulo *PathPlanning* y calculando los comandos de movimiento adecuados para alcanzar dicho destino intermedio. En el cálculo de los comandos de movimiento adecuados el sistema siempre trata de alcanzar el destino en línea recta, y cuando los obstáculos del entorno se encuentran demasiado cercanos al agente se activa el esquema de evitación de obstáculos implementado. Los comandos de movimiento para la plataforma robótica son compartidos con el resto de módulos.

7. ***IFRobot*:** el módulo *IFRobot* accede continuamente a los comandos de movimiento compartidos por el módulo *Navigation*, enviándolos a la plataforma robótica y accediendo de nuevo a la información sensorial de la misma, comenzando de nuevo el ciclo de operación.

Una descripción más detallada de la implementación y funcionamiento de todas y cada una de las capas y módulos del sistema se realizará en los siguientes capítulos.

4.4 Desarrollo del sistema

La descomposición del sistema en capas permite llevar a cabo un desarrollo progresivo del mismo, pues se pueden incorporar nuevos comportamientos en el sistema añadiendo nuevas capas sobre las ya existentes. En este sentido, el sistema propuesto va a ser desarrollado incrementalmente, añadiendo progresivamente nuevos comportamientos más complejos mediante el desarrollo de nuevas capas. La relación existente entre los comportamientos incorporados en el sistema y las capas que los desarrollan son los siguientes:

- **Control básico:** mediante la implementación de la Capa Física el sistema es capaz de enviar comandos de movimiento a la plataforma robótica y acceder a la información interna de la misma, compuesta básicamente por las lecturas de los sensores sonar y por la posición odométrica.
- **Navegación reactiva:** al incorporar la Capa de Navegación el sistema es capaz de navegar reactivamente evitando los obstáculos fijos y/o móviles del entorno.
- **Modelado del entorno:** al añadir la Capa de Representación se consiguió que el sistema es capaz de construir un modelo del entorno por el cual navega.
- **Cálculo de rutas y caminos:** añadiendo la Capa de Planificación el sistema es capaz de calcular rutas y caminos libres de obstáculos para alcanzar cualquier destino final especificado.
- **Comandos:** con la incorporación de la Capa de Comando se posibilitó el envío de comandos de navegación al agente para alcanzar un destino final según bajo un determinado nivel de navegación.

4.5 Ampliación del sistema

La descomposición obtenida del sistema es extraordinariamente modular, posibilitando una sencilla incorporación de nuevos comportamientos y funcionalidades, siendo este uno de los objetivos básicos perseguidos mediante la implementación de la arquitectura de control. Entre algunas de las futuras ampliaciones que se pueden incorporar en el sistema se destacan:

- **Navegación en exteriores:** añadiendo un sensor *GPS* en la plataforma robótica se puede desarrollar un sistema de localización para la navegación de agentes en entornos exteriores [VMPU+06].
- **Localización:** mediante el desarrollo de nuevos algoritmos de localización en el módulo *IFRobot* se puede mejorar el sistema de localización del agente utilizando algoritmos más complejos y eficientes [PVN+04], o se puede implementar un sistema de localización global mediante cadenas de Markov que permita posicionar al agente en cualquier ubicación del entorno conocido [PUdTS04].

- **Seguimiento de personas:** mediante la implementación de un módulo *Follow* en la Capa de Navegación es posible desarrollar comportamientos para seguir personas.
- **Exploración completa:** integrando un módulo *Exploration* en la Capa de Planificación se puede dotar al agente con un sistema de exploración inteligente del entorno [PPB+02].
- **Entornos virtuales:** desarrollando un módulo *3DMap* en la Capa de Representación se puede generar una representación virtual 3D del entorno donde el agente navega, incorporando información de texturas captadas con un sistema de cámaras [LMP+01].
- **Visión:** incorporando un módulo *Vision* en la Capa de Comando es posible implementar un sistema de reconocimiento que pueda dirigir al agente en la realización de comportamientos más complejos como la recolección de objetos del entorno [Rod01, Val03, dT04].

5 Utilización de la arquitectura *DLA*

Una vez diseñada la nueva arquitectura de control, se va a presentar a modo de resumen una visión genérica de los distintos elementos que se han desarrollado, describiendo la correcta utilización de los mismos y los distintos lenguajes y plataformas soportados en su uso.

5.1 Elementos básicos de la arquitectura *DLA*

La figura 3.7 muestra los distintos elementos que componen la arquitectura de control *DLA*, cuya utilización se realiza como se describe a continuación:

- ***DLAServer*:** el servidor central *DLAServer* permite el empleo de la arquitectura de control *DLA* mediante la creación, la lectura y la escritura de las conexiones y buzones necesarios en la operación del sistema.
- ***DLALibrary*:** la librería *DLALibrary* implementa todas las funciones necesarias para que cualquier módulo se conecte con el servidor central *DLAServer* y utilice las conexiones y buzones que precise [4].
- ***DLAAdmin*:** el administrador remoto *DLAAdmin* permite monitorizar y modificar en tiempo real la operación del servidor central *DLAServer*, constituyendo una herramienta muy potente para analizar el flujo de intercambio de datos existente en el sistema y facilitar así la depuración del mismo [5].
- ***RobotServer*:** el servidor hardware *RobotServer* controla el acceso a la plataforma robótica del sistema mediante un conjunto estándar de comandos. Este conjunto de co-

[4] Una descripción detallada del uso de las funciones contenidas en la librería *DLALibrary* se puede encontrar en el apartado 3.1.2 del apéndice A.

[5] Una descripción detallada del uso del administrador remoto *DLAAdmin* se puede encontrar en el apartado 3.1.3 del apéndice A.

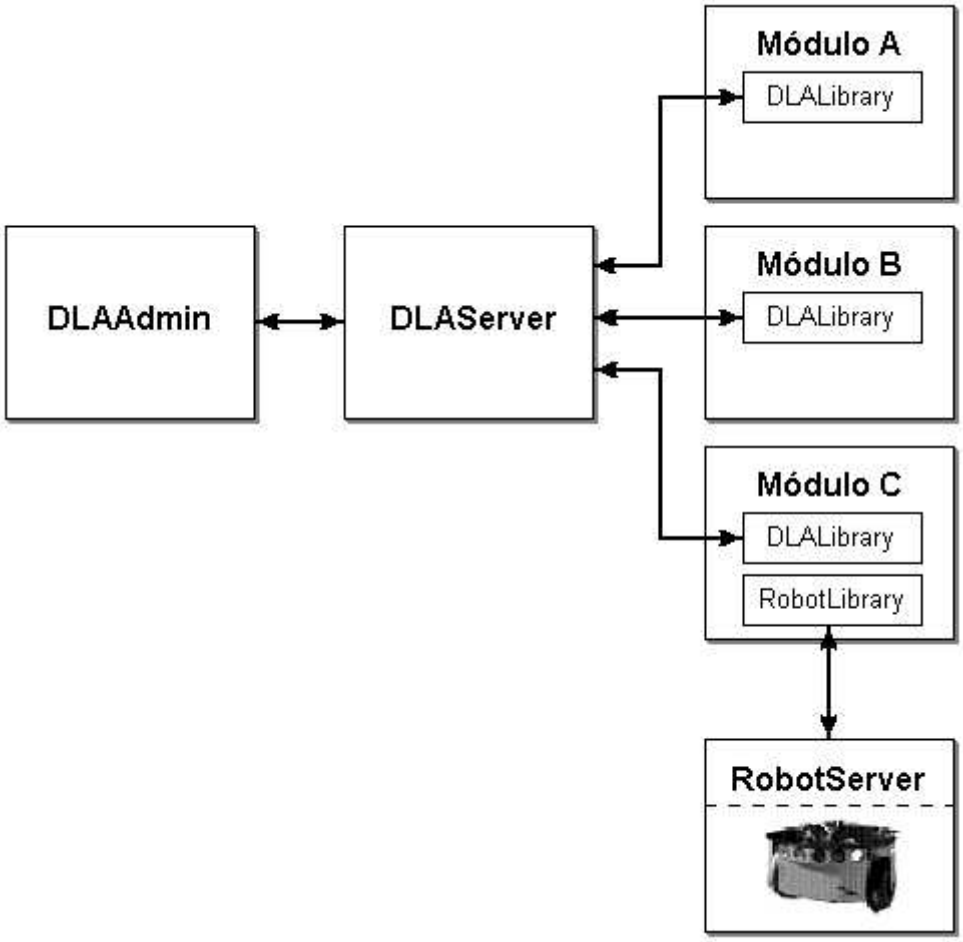

Figura 3.7: Elementos que componen la arquitectura de control *DLA*.

mandos es común para todas las plataformas robóticas que se deseen utilizar, facilitando enormemente la portabilidad hardware del sistema.

- ***RobotLibrary***: la librería *RobotLibrary* implementa el conjunto estándar de comandos para que cualquier módulo pueda controlar cualquier plataforma robótica a través del correspondiente servidor hardware [6].

5.2 Lenguajes y plataformas soportadas

Entre los objetivos básicos de la arquitectura implementada se encuentran la sencillez y la portabilidad en su uso, minimizando las restricciones impuestas a los usuarios en el desarrollo de los módulos que posteriormente integrarán el sistema. Una de las restricciones más severas radica en la imposición de determinado lenguaje de programación o de determinada plataforma para el uso de la arquitectura, por lo que se ha intentado garantizar la máxima portabilidad

[6]Una descripción detallada del uso de las funciones contenidas en la librería *RobotLibrary* se puede encontrar en el apartado 3.2.2 del apéndice A.

posible entre distintos lenguajes y plataformas.

Los lenguajes y plataformas soportados por los distintos elementos que componen la arquitectura de control *DLA* son los que se describen a continuación:

- **Servidor central *DLAServer*:** los distintos módulos del sistema son totalmente independientes del servidor central *DLAServer*, por lo que el lenguaje de programación y/o plataforma empleada en el desarrollo del mismo no impone restricción alguna sobre el lenguaje de programación y/o plataforma empleada en el desarrollo de los módulos. El servidor central *DLAServer* ha sido desarrollado en *C* sobre *Linux* y en *JAVA* [7], por lo que puede ser instalado en cualquier máquina que soporte dichos lenguajes y plataformas.
- **Librería *DLALibrary*:** para que los distintos módulos del sistema hagan uso de la arquitectura de control *DLA* deben utilizar la librería *DLALibrary*, por lo que dicha librería tendrá que soportar la portabilidad entre todos los lenguajes y plataformas que se deseen utilizar en el desarrollo de los distintos módulos. En este sentido, es importante diferenciar entre los esquemas distribuidos y el esquema local implementado por la librería, pues cada uno de ellos emplea distintos mecanismos de comunicación entre procesos, los cuales son soportados de forma diferente por cada lenguaje y/o plataforma:
  - **Esquemas distribuidos:** los esquemas distribuidos utilizan los *sockets* como mecanismo básico de comunicación entre procesos, por lo que cualquier lenguaje y/o plataforma que soporte el uso de *sockets* es susceptible de ser utilizado en el desarrollo de módulos que puedan utilizar la arquitectura. Hoy en día prácticamente cualquier lenguaje y/o plataforma soporta el empleo de *sockets*, por lo que las posibilidades de portabilidad de la vertiente distribuida de la librería *DLALibrary* es prácticamente ilimitada. En concreto, la vertiente distribuida de la librería *DLALibrary* puede ser actualmente utilizada en *C*, *JAVA* y *Matlab* [8] para *Linux* y *Windows*, aunque si se detecta la necesidad de incluir nuevos lenguajes y/o plataformas para el desarrollo e integración en la arquitectura de algún tipo de módulo específico basta con reescribir la librería *DLALibrary* en dichos lenguajes y plataformas, siempre que soporten el empleo de *sockets*.
  - **Esquema local:** el esquema local utiliza la memoria compartida y los semáforos como mecanismos básicos de comunicación entre procesos, por lo que cualquier lenguaje y plataforma que soporte estos mecanismos es susceptible de ser utilizado en el desarrollo de módulos que puedan utilizar la arquitectura. La memoria compartida y los semáforos no son mecanismos tan extendidos como los *sockets* a través de los distintos lenguajes y/o plataformas, por lo que las posibilidades de portabilidad de la vertiente local de la librería *DLALibrary* no son tan elevadas. En concreto, la vertiente local de la librería *DLALibrary* puede ser actualmente utilizada en *C* para *Linux*, aunque si se detecta la necesidad de incluir nuevos lenguajes y/o plataformas para el desarrollo e

[7] La versión *JAVA* del servidor central *DLAServer* ha sido desarrollada mediante un Proyecto Fin de Carrera realizado bajo la dirección del autor de la presente Tesis [Igl04].

[8] El desarrollo de la librería *DLALibrary* en *JAVA* y *Matlab* está siendo actualmente realizada por sendos Proyectos Fin de Carrera realizados bajo la dirección del autor de la presente Tesis.

Lenguaje	**Distribuido Básico**	**Distribuido Síncrono**	**Local Síncrono**
C	Sí	Sí	Sí
JAVA	Sí	Sí	No
Matlab	Sí	Sí	No
Plataforma	**Distribuido Básico**	**Distribuido Síncrono**	**Local Síncrono**
Linux	Sí	Sí	Sí
Windows	Sí	Sí	No

Tabla 3.2: Lenguajes y plataformas actualmente soportados por la arquitectura de control *DLA*.

integración en la arquitectura de algún tipo de módulo específico basta con reescribir la librería *DLALibrary* en dichos lenguajes y plataformas, siempre que soporten el empleo de memoria compartida y semáforos.

Cualquier módulo desarrollado en estos lenguajes y/o plataformas puede utilizar la arquitectura de control *DLA* y ser directamente integrado en el sistema, posibilitando el intercambio de datos con cualquier otro módulo desarrollado en cualquier otro lenguaje y/o plataforma soportado. En la tabla 3.2 se muestran a modo de resumen los lenguajes y plataformas actualmente soportados por los distintos esquemas implementados en la arquitectura de control *DLA*: esquema distribuido básico, esquema distribuido síncrono y esquema local síncrono [9].

Como se puede apreciar en la tabla 3.2, la librería *DLALibrary* ha sido totalmente desarrollada para *C* sobre *Linux*, permitiendo la utilización de los esquemas distribuidos o del esquema local por parte de cualquier módulo desarrollado en dicho lenguaje y plataforma. Los módulos que deseen utilizar otros lenguajes como *JAVA* o *Matlab* o bien otras plataformas como *Windows* poseen actualmente una versión reducida de la librería que únicamente implementa los esquemas distribuidos de la arquitectura.

- **Administrador remoto *DLAAdmin*:** el servidor central *DLAServer* es totalmente independiente del administrador remoto *DLAAdmin*, por lo que el lenguaje de programación y/o plataforma empleada en el desarrollo del mismo no impone restricción alguna sobre el lenguaje de programación y/o plataforma empleada en el desarrollo del servidor central *DLAServer*. El administrador remoto *DLAAdmin* ha sido desarrollado en *C* sobre *Linux*, en *JAVA* [10] y en versión *Web* para ser utilizado con cualquier navegador [11], por lo que puede ser instalado en cualquier máquina que soporte dichos lenguajes y plataformas.

- **Servidor hardware *RobotServer*:** los distintos módulos del sistema son totalmente independientes de cualquier servidor hardware *RobotServer*, por lo que el lenguaje de programación y/o plataforma empleada en el desarrollo del mismo no impone restricción

[9] Una descripción detallada de los distintos esquemas implementados en la arquitectura *DLA* se puede encontrar en el apartado 3.2.

[10] La versión *JAVA* del administrador remoto *DLAAdmin* ha sido desarrollada mediante un Proyecto Fin de Carrera realizado bajo la dirección del autor de la presente Tesis [Gon02].

[11] La versión *Web* del administrador remoto *DLAAdmin* ha sido desarrollada mediante un Proyecto Fin de Carrera realizado bajo la dirección del autor de la presente Tesis [Pod05].

alguna sobre el lenguaje de programación y/o plataforma empleada en el desarrollo de los módulos. Actualmente se dispone de servidores hardware para las plataformas robóticas *Nomad 200* y *Pioneer P2AT*, desarrollados en *C* sobre *Linux*, pues es el lenguaje y plataforma soportados por las librerías de acceso a dichas plataformas robóticas suministradas por sus fabricantes.

- **Librería *RobotLibrary*:** para que los distintos módulos del sistema hagan uso de la plataforma robótica deben utilizar la librería *RobotLibrary*, por lo que dicha librería tendrá que soportar la portabilidad entre todos los lenguajes y plataformas que se deseen utilizar en el desarrollo de los distintos módulos que accedan a la plataforma robótica. La librería *RobotLibrary* únicamente utiliza los *sockets*, por lo que cualquier lenguaje y/o plataforma que soporte el uso de *sockets* es susceptible de ser utilizado en el desarrollo de módulos que puedan utilizar la plataforma robótica. Hoy en día prácticamente cualquier lenguaje y/o plataforma soporta el empleo de *sockets*, por lo que las posibilidades de portabilidad de la librería *RobotLibrary* es prácticamente ilimitada. En concreto, la librería *RobotLibrary* puede ser actualmente utilizada en *C* para *Linux*, aunque si se detecta la necesidad de incluir nuevos lenguajes y/o plataformas para el desarrollo de algún tipo de módulo específico basta con reescribir la librería *RobotLibrary* en dichos lenguajes y/o plataformas, siempre que soporten el empleo de *sockets*.

Es de destacar que la sencillez y la portabilidad conseguidas con la arquitectura de control *DLA* es muy superior a las exhibidas por cualquiera de las arquitecturas de control híbridas más significativas analizadas en el apartado 2.2 del capítulo 2.

6 Parámetros de la arquitectura *DLA*

La arquitectura de control *DLA* cubre completamente las especificaciones descritas en el apartado 2. Un análisis detallado de la misma permite extraer los principales parámetros que caracterizan su funcionamiento:

- **Sencillez:** la arquitectura de control *DLA* es extremadamente sencilla de utilizar, requiriendo únicamente la utilización de la librería *DLALibrary*, compuesta por funciones sencillas que encapsulan la complejidad en la gestión de las conexiones y buzones para los esquemas implementados. El resto del funcionamiento de la arquitectura de control *DLA* es totalmente transparente al usuario.

- **Escalabilidad:** la arquitectura de control *DLA* es totalmente escalable en varios niveles. Gracias al empleo de *sockets* la arquitectura posibilita la implementación distribuida del sistema, técnica indispensable para permitir el progresivo aumento de capacidad de la misma. Mediante los mecanismos de sincronización implementados se permite la activación y desactivación selectiva de módulos según su necesidad, optimizando el uso de recursos empleados. Por último, si el tráfico generado en el intercambio de información entre módulo es demasiado elevado como para ser gestionado por un único servidor central *DLAServer*, se

pueden ejecutar distintos servidores centrales en diferentes máquinas con el fin de distribuir el tráfico total entre todos ellos. El proceso es inmediato y no requiere ninguna modificación en los módulos del sistema, a parte de indicar la ubicación del servidor central que se quiere emplear para acceder a una determinada conexión o buzón. Esta característica permite, por ejemplo, separar las conexiones grandes de las pequeñas en distintos servidores, por lo que se pueden mejorar más aun las prestaciones conseguidas anteriormente y prevenir posibles problemas de congestión.

- **Extensibilidad:** la arquitectura de control *DLA* permite la ampliación del sistema en varios niveles:
  - **Hardware:** la incorporación de nuevos recursos hardware en el agente es extremadamente sencilla gracias a la existencia del servidor hardware *RobotServer*, el cual independiza el sistema desarrollado del hardware específico de la plataforma robótica gracias al conjunto de comandos estándar definido. Sustituir o añadir nuevo hardware no requiere la reestructuración de ningún módulo, tan solo la modificación del servidor hardware correspondiente y la ampliación en su caso del conjunto de comandos estándar definido, por lo que es muy sencillo incorporar nuevos sensores y actuadores como sensores *GPS*, sensores láser, anillos de sensores infrarrojos, brazos articulados, cámaras ...
  - **Software:** la incorporación de nuevos módulos en el sistema que implementen nuevos comportamientos es inmediata gracias al empleo de la librería *DLALibrary*, la cual garantiza el acceso inmediato a todos los datos compartidos en el sistema por el resto de módulos, así como la posibilidad de utilizar mecanismos de sincronización entre los mismos. La incorporación de nuevos módulos en el sistema no implica ninguna modificación en el resto de módulos, más allá del acceso por parte de estos a los datos compartidos por los nuevos módulos.

- **Portabilidad:** la arquitectura de control *DLA* es altamente portable en varios niveles:
  - **Hardware:** gracias a la implementación del servidor hardware *RobotServer* se posibilita la migración directa de una plataforma robótica por otra, requiriendo únicamente el desarrollo de un servidor hardware *RobotServer* específico para la nueva plataforma robótica. Se ha desarrollado un servidor hardware genérico tanto en *Linux* como en *Windows*, por lo que se puede emplear cualquier plataforma robótica que emplee estos sistemas operativos sin la necesidad de modificar ninguno de los módulos del sistema, debiendo implementar únicamente el conjunto de comandos estándar apropiados a dicha plataforma robótica. En este sentido, se ha probado el sistema propuesto sobre una plataforma robótica *Nomad 200*, una plataforma robótica *Pioneer AT* y una plataforma robótica propia de la *Universitat Politècnica de Catalunya* denominada *Spherik*, desarrollando servidores hardware para cada una de las mismas. Implementar el sistema de navegación desarrollado en una u otra plataforma no requiere ninguna modificación de los módulos del sistema, solamente la conexión al servidor hardware *RobotServer* correspondiente a la plataforma robótica que se quiera controlar.

  - **Software:** los esquemas distribuidos de la arquitectura de control *DLA* están completamente basados en *sockets*, cuya implementación está soportada por prácticamente la totalidad de los lenguajes y plataformas modernas. La librería *DLALibrary* de uso de la arquitectura ha sido desarrollada en *JAVA*, *Matlab* y *C* tanto para *Windows* como para *Linux*. Cualquier módulo programado en estos lenguajes y plataformas puede intercambiar datos con cualquier otro módulo programado en cualquier otro lenguaje o plataforma soportados. Añadir nuevos lenguajes y plataformas es extremadamente sencillo y sólo requiere la implementación de la librería *DLALibrary* para el lenguaje y la plataforma deseados. El esquema local está basado en memoria compartida y semáforos, y actualmente la vertiente local de la librería de acceso a la arquitectura está desarrollada en *C* para *Linux*, si bien se pueden usar todos aquellos lenguajes y plataformas que soporten dichos mecanismos.

- **Flexibilidad:** la operación de la arquitectura *DLA* es muy flexible en varios niveles. En primer lugar permite el intercambio genérico de cualquier tipo de datos entre los distintos módulos, tratando a estos datos simplemente como un flujo de bytes que puede contener cualquier tipo de datos simple o abstracto. Además, puesto que un módulo sólo necesita el acceso a los datos compartidos del sistema y desconoce la existencia y ubicación del resto de módulos, la distribución de los módulos entre las máquinas disponibles en el sistema es totalmente transparente. Por este mismo motivo, cualquier módulo puede ser detenido, modificado y reiniciado de nuevo sin que el resto de módulos deba ser detenido, permitiendo una reestructurarción dinámica del sistema incluso si las condiciones de carga lo requieren.

- **Depuración:** los módulos del sistema pueden ser independientemente desarrollados y depurados, puesto que para operar sólo necesitan la existencia del servidor central de la arquitectura *DLAServer*, y en ningún caso la existencia del resto de módulos. La depuración de un módulo se puede realizar de forma sencilla inyectando en el sistema los datos de prueba que se desea que el módulo procese y visualizando los datos de salida generados por dicho módulo, lo cual es fácilmente realizable con el administrador remoto *DLAAdmin*. El administrador remoto *DLAAdmin* también permite monitorizar en tiempo real toda la información contenida en el sistema, facilitando más aun el proceso de depuración del mismo.

- **Eficiencia:** gracias a los distintos esquemas implementados es posible optimizar en gran medida el uso de los recursos disponibles en el sistema. En este sentido, los mecanismos de sincronización implementados permiten aumentar considerablemente la eficiencia del sistema bajo ciertas circunstancias. Debido además al carácter distribuido de la arquitectura de control *DLA* la sobrecarga que impone el servidor central *DLAServer* en el sistema se reduce al tiempo de servicio de las peticiones realizadas por los módulos, no imponiendo ningún tipo de carga de memoria ni de uso del procesador en las máquinas donde residen los módulos.

7 Conclusiones

En el presente capítulo se ha abordado el diseño de una nueva arquitectura de control para simplificar el desarrollo de agentes autónomos móviles, persiguiendo como objetivo prioritario la **transparencia** en su uso, concretada en la sencillez y portabilidad de la misma. Un análisis detallado de las arquitecturas de control híbridas más significativas revela que cada una constituye una solución particular que plantea sus propias restricciones y prestaciones para el desarrollo de agentes autónomos móviles, pero ninguna de ellas satisface la sencillez y portabilidad necesarias para permitir una filosofía de trabajo basada en la libre elección por parte del diseñador de las herramientas más adecuadas para implementar los distintos módulos. Es por ello por lo que finalmente se decide llevar a cabo el desarrollo completo de una nueva arquitectura de control que satisfaga estas necesidades.

La nueva arquitectura de control se ha denominado ***DLA*** (***D****istributed and* ***L****ayered* ***A****rchitecture*), y permite implementar sistemas cooperativos mediante la interacción de procesos libremente distribuidos. Esta arquitectura ha sido utilizada para implementar una infraestructura básica que incorpore la capacidad de navegación en entornos dinámicos no estructurados a un agente autónomo móvil de cualquier tipo, aunque es posible utilizarla en otro tipo de sistemas cooperativos incluso fuera del ámbito de la Robótica. Las especificaciones de esta nueva arquitectura de control han sido detalladamente analizadas, manteniendo siempre como objetivos prioritarios en el desarrollo de la misma la sencillez y la portabilidad en su uso. Además, se han perseguido otras características importantes, como escalabilidad, extensibilidad, flexibilidad, facilidad de depuración y eficiencia.

El **estilo** de la arquitectura *DLA*, referente a los mecanismos suministrados para permitir la comunicación entre los módulos, sigue un esquema de interacción entre procesos con un modelo de interconexión indirecta mediante *sockets*. Con esta filosofía se implementa un esquema de **memoria compartida distribuida**, que se adecúa perfectamente a las especificaciones originales demostrando una gran sencillez de uso y una alta portabilidad. El estilo de la arquitectura ha sido desarrollado con distintos esquemas, los cuales permiten optimizar el uso de los recursos disponibles en distintas circunstancias: esquema distribuido básico, esquema distribuido síncrono y esquema local síncrono.

La **estructura** de la arquitectura *DLA*, referente a la descomposición del sistema en módulos, sigue una filosofía **híbrida**, al constituir actualmente la alternativa más eficiente en el desarrollo de agentes autónomos móviles. Para facilitar el diseño del sistema se ha utilizado un modelo de operación asíncrono, que permite la existencia de distintas escalas temporales con módulos más rápidos y módulos más lentos. Se ha realizado la descomposición en módulos según una distribución en capas, pues dota al sistema de una mayor modularidad y flexibilidad al permitir un desarrollo incremental y progresivo del mismo. De esta forma se pueden desarrollar comportamientos básicos, y una vez depurados se pueden añadir nuevas capas que incorporen comportamientos más complejos. Siguiendo esta filosofía se han implementado gradualmente distintos niveles de navegación: Navegación Reactiva, Navegación Planificada y Navegación Topológica. Cada uno de estos niveles muestra sus propias capacidades de navegación, de acuerdo a la Jerarquía de

Navegación que se pretende implementar.

Por último, se ha descrito el uso de la arquitectura de control propuesta, enumerando los distintos elementos que la componen. También se han descrito los parámetros finales que describen el funcionamiento de la arquitectura de control, demostrando el cumplimiento de las especificaciones originales.

Capítulo 4

Capas Física, de Representación y de Comando

Tras realizar el diseño de la arquitectura de control propuesta y obtener la descomposición del sistema en capas y módulos es necesario abordar el diseño de los mismos. En este capítulo se va a describir la implementación de la Capa Física, la Capa de Representación y la Capa de Comando, necesarias para desarrollar el correcto funcionamiento de los comportamientos reactivos y deliberados del sistema.

Este capítulo se ha organizado como se indica a continuación. En el apartado 1 se describe la implementación de la Capa Física, responsable de incorporar los comportamientos de acceso y control de la plataforma robótica. En el apartado 2 se describe la implementación de la Capa de Representación, responsable de incorporar el comportamiento básico de modelado del entorno. En el apartado 3 se describe la implementación de la Capa de Comando, responsable de implementar el interfaz con el usuario que permite especificar los comandos a ejecutar y representar la información sobre el estado del agente. Por último, en el apartado 4 se describen las principales conclusiones extraídas a lo largo del presente capítulo.

1 Implementación de la Capa Física

Es evidente que los distintos algoritmos implementados en el sistema deben operar sobre los datos del entorno captados por la plataforma robótica. Además, los resultados de estos algoritmos deben ser convenientemente traducidos a comandos de movimiento para que las tareas que implementan puedan ser desarrolladas correctamente.

La Capa Física se encarga de la recogida y traducción adecuada de los datos del entorno captados por la plataforma robótica, así como de la traducción y especificación de los comandos de movimiento que la plataforma robótica debe ejecutar. Se corresponde con la capa de más bajo nivel del sistema, por lo que además de estas funciones también es responsable de implementar algunas funciones básicas de seguridad para proteger la integridad física de la propia plataforma

robótica frente a potenciales colisiones con los obstáculos fijos y/o móviles del entorno.

1.1 Módulo *IFRobot*

La Capa Física se va a implementar mediante el módulo *IFRobot*, que será el responsable de las siguientes funciones básicas:

- **Recogida de datos:** responsable de acceder a la plataforma robótica a través del servidor hardware *RobotServer*, recogiendo la información captada del entorno por medio de los sensores y la información de localización del sistema de odometría de la plataforma robótica.
- **Envío de comandos:** responsable de recoger el resultado del procesamiento realizado por el resto de módulos y traducirlos convenientemente a los comandos de movimiento adecuados para la plataforma robótica.
- **Alarma de colisión:** responsable de detener inmediatamente la plataforma robótica si se detecta proximidad a algún obstáculo por debajo de un umbral de seguridad U_{sec}, para evitar colisiones independientemente del comando de movimiento que especifiquen el resto de módulos del sistema.

La función de **recogida de datos** implica la correcta fusión de la información sensorial que es captada por la plataforma robótica, de forma que distintos tipos de sensores (sonar, infrarrojos, láser, ...) sean convenientemente combinados para proporcionar los datos adecuados que necesitan el resto de módulos del sistema. También es necesario unificar las unidades de las lecturas proporcionadas por los distintos sensores y traducir la posición de la plataforma robótica al sistema de coordenadas utilizado en el sistema.

El problema de la localización de la plataforma robótica es bastante crítico y complejo en navegación [BEF96]. En la actualidad, prácticamente cualquier plataforma robótica posee un sistema de odometría que se utiliza como ayuda para conocer la posición del mismo en el entorno, contando para ello el número de vueltas recorridas por cada una de las ruedas de la plataforma robótica. El uso de la odometría constituye una solución muy barata al problema de la localización, si bien tiene el inconveniente de que su exactitud decrece con la distancia recorrida, debido a deslizamientos (*slippage*) y a la acumulación de los errores cometidos. Por ejemplo, un error de orientación en el sistema de odometría de únicamente 1^o genera un error de traslación de unos 10 cm tras recorrer 6 m. Es por ello por lo que se suelen utilizar otra serie de técnicas complementarias a la odometría para resolver el problema de la localización.

Para intentar corregir en la medida de lo posible los errores derivados del sistema de odometría se ha utilizado en el sistema propuesto un filtro de Kalman [Kal60], por constituir una solución sencilla y barata que no requiere la utilización de sensores especiales ni de marcas en el entorno. El método implementado se basa en la combinación de una representación del entorno y de las

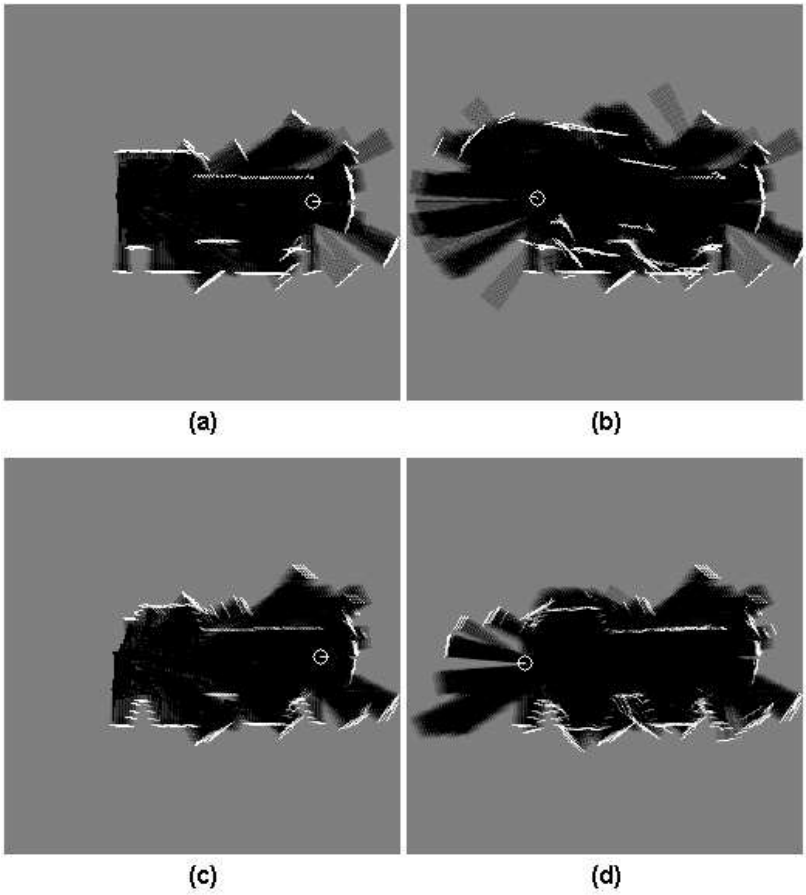

Figura 4.1: Utilización del filtrado de Kalman en el proceso de localización: a) y b) sin utilizar Kalman; c) y d) utilizando Kalman.

lecturas obtenidas de los sensores para corregir los posibles errores del sistema de odometría [CK93] [1].

Los resultados obtenidos con el método implementado se muestran en la figura 4.1. La figura 4.1.a se representa la construcción del mapa del entorno que capta el agente tras recorrer un entorno compuesto por un pasillo. Si se gira al agente sobre sí mismo al llegar al final del pasillo y se vuelve a recorrer el mismo en sentido inverso el nuevo mapa generado es el que se muestra en la figura 4.1.b, donde se pueden apreciar con total claridad los errores de orientación del sistema de odometría. Tras aplicar el filtrado de Kalman los resultados obtenidos en el mismo proceso son los que se muestran en las figuras 4.1.c y 4.1.d, donde los errores del sistema de odometría han sido sustancialmente reducidos.

Es preciso matizar que el problema de la localización es realmente complejo de resolver, constituyendo una línea de investigación bastante activa [GBFK98, GF02]. En este sentido, el trabajo realizado en la presente Tesis puede ser ampliamente mejorado con la incorporación de métodos más complejos y precisos. Mediante la utilización del método propuesto, bastante sencillo y

[1] El método de localización implementado mediante el filtro de Kalman se basa en el trabajo de un Proyecto Fin de Carrera realizado bajo la dirección del autor de la presente Tesis [Nar03].

claramente mejorable, simplemente se ha pretendido demostrar que el sistema está concebido para integrar aquellos métodos de localización que se consideren oportunos.

La función de **envío de comandos** implica el control de las operaciones de movimiento permitidas por la plataforma robótica. Por ejemplo, si se desea que la plataforma robótica únicamente pueda avanzar hacia delante, un comando de retroceso hacia atrás debe ser traducido por este módulo a un comando que haga girar al agente sobre sí mismo hasta que pueda avanzar en la dirección adecuada. También es necesario realizar la adecuada traducción de los comandos a las unidades y al sistema de coordenadas empleados por la plataforma robótica.

Por último, la función de **alarma de colisión** desarrolla un nivel de seguridad adicional en el sistema ante potenciales situaciones de colisión, anulando cualquier comando de movimiento enviado y deteniendo la plataforma robótica ante cualquier circunstancia que amenace la integridad de la misma. Una vez detectado un obstáculo por debajo del umbral de seguridad establecida la plataforma robótica se detiene y gira hasta que dicho obstáculo desaparece de la dirección de avance.

El interfaz que define los datos de entrada y salida que este módulo comparte con el resto de módulos es el siguiente:

- **Datos de entrada:**
  - **Comandos de movimiento:** generados por el módulo *Navigation*.
  - **Mapa métrico:** generado por el módulo *MapMetric* [2].
- **Datos de salida:**
  - **Posición del agente:** utilizada por los módulos *Navigation*, *MapMetric*, *MapTopo*, *PathPlanning* y *IFUser*.
  - **Lectura de los sensores:** utilizadas por los módulos *Navigation* y *MapMetric*.

2 Implementación de la Capa de Representación

Los algoritmos de planificación deliberada tratan de establecer planes eficientes para llevar a cabo determinadas tareas en el entorno donde desarrolla su actividad el agente. Para ello es necesario que se disponga de un modelo adecuado que represente el entorno, determinando la exactitud de dicho modelo la validez de los planes calculados para ser llevados a cabo. Es importante, pues, que dicho modelo sea lo más fidedigno posible en todo momento, siempre con el grado de precisión requerido por los algoritmos de planificación que lo van a procesar.

A continuación se describe el procedimiento empleado para la generación del modelo del entorno que posteriormente utilizarán los correspondientes algoritmos de planificación.

[2] Recuérdese que la representación del entorno es utilizada por el filtrado de Kalman para corregir los posibles errores del sistema de odometría.

2.1 Mapas métricos probabilísticos

Existen numerosas alternativas a la hora de generar una representación del entorno donde navega el agente [Thr02], aunque la más extendida por su sencillez y buenas prestaciones se basa en el uso de mapas métricos probabilísticos [Mor88], por lo que será la utilizada en la presente Tesis.

Este tipo de representaciones tratan de reproducir explícitamente la geometría del entorno, dividiéndolo en una rejilla de Num_Col columnas y Num_Row filas de celdas homogéneas de tamaño $Size\_Cell$ cada una. Cada una de estas celdas está asociada con una determinada posición del entorno, y el valor almacenado en la misma hace referencia a la probabilidad de que dicha posición esté ocupada por un obstáculo de acuerdo con la información presente y pasada proporcionada por los sensores del agente. El entorno completo se representa pues mediante una agrupación de tamaño $Num\_ColxNum\_Row$ celdas.

Los mapas métricos probabilísticos constituyen una alternativa muy interesante a la hora de generar una representación métrica del entorno, pues poseen importantes ventajas:

- Son generados y actualizados de una forma muy rápida y sencilla.
- Permiten una directa integración espacial y temporal de las sucesivas medidas proporcionadas por los sensores, imprescindible en aquellos sensores que poseen un considerable grado de incertidumbre en sus medidas como es el caso de los sensores sonar.
- La geometría del mapa generado se corresponde directamente con la geometría del entorno, controlando el tamaño de la celda especificado el grado de descomposición o resolución que se desea alcanzar en la representación del entorno, y el número de celdas almacenadas el tamaño del entorno que se quiere representar.

Sin embargo, los mapas métricos probabilísticos poseen algunos inconvenientes que es necesario abordar correctamente para no degradar las prestaciones finales del sistema:

- Poseen un evidente compromiso en la elección del tamaño de celda, al influir tanto en la resolución como en el tamaño final de la representación generada.
- Contiene una elevada cantidad de datos si se representan grandes entornos, aumentando el tiempo de procesamiento requerido por los posteriores algoritmos de planificación que operan sobre el mapa métrico probabilístico.

La elección correcta del tamaño de celda constituye quizás el aspecto más crítico en la implementación de un mapa métrico probabilístico. Un tamaño pequeño de celda permite obtener una alta resolución, aunque también implica un mayor número de celdas a la hora de representar el entorno. Por el contrario, un tamaño de celda grande necesita un menor número de celdas para representar el entorno, por lo que se reduce la cantidad de datos necesarios en dicha representación, aunque así también se está disminuyendo la resolución del entorno representado.

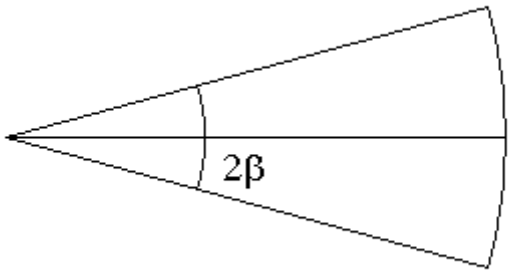

Figura 4.2: Diagrama de radiación de un sensor sonar.

Como elección de compromiso se suele elegir un tamaño de celda algo menor que el tamaño del agente que se esté utilizando.

La cantidad de datos que suele requerir una representación del entorno mediante un mapa métrico probabilístico suele ser elevada si se desea almacenar un entorno lo suficientemente grande. Por ejemplo, si se elige un tamaño de celda de 50 cm, para representar un entorno de 100x100 m^2 es necesario un mapa métrico de tamaño 200x200 celdas, es decir, 40.000 celdas. Dependiendo de la aplicación este entorno puede ser suficiente o no, por lo que si se requiere un tamaño de entorno aun mayor será necesario aumentar el número de celdas del mapa, aumentando pues la cantidad de datos asociados con el mismo. Este inconveniente puede ser resuelto combinando adecuadamente los mapas métricos probabilísticos con otro tipo de representaciones del entorno más compactas, solución adoptada en el sistema implementado y que será analizada en el apartado 3 del capítulo 6.

Existen distintas estrategias posibles a la hora de generar la representación del entorno mediante un mapa métrico probabilístico. Este tipo de representaciones se basan en el cálculo mediante técnicas estadísticas de la probabilidad de que una determinada celda esté ocupada, utilizando la adecuada fusión de las sucesivas lecturas proporcionadas por los sensores. Para generar correctamente la probabilidad de ocupación de cada celda es necesario utilizar las siguientes funciones:

- **Modelado del sensor:** función de probabilidad que representa el comportamiento del sensor utilizado, determinando la incertidumbre en la medida obtenida en función de parámetros físicos como la distancia y el ángulo entre el sensor y el obstáculo. El comportamiento de cualquier sensor sonar queda modelado mediante un diagrama de radiación con un lóbulo de anchura 2β, como se representa en la figura 4.2 [3]. Debido a la anchura del lóbulo no se puede precisar la posición exacta de un obstáculo cuando es detectado, y las distintas funciones que modelan el comportamiento del sensor pretenden determinar

[3]En realidad el diagrama de radiación de un sensor sonar suele poseer una serie de lóbulos laterales además del lóbulo principal, pero sus efectos se suelen despreciar por tener una influencia mucho menor que la del lóbulo principal.

estadísticamente la probabilidad de que un obstáculo detectado se encuentre en una determinada posición del lóbulo. Algunos de los modelos de sensores sonar más utilizados son el modelo de Oriolo [OVU95], el modelo de Pagac [PNDW98] o el modelo de Matia [MJ98].

- **Regla de actualización:** función de actualización del valor de probabilidad de ocupación de las celdas del mapa métrico en función de las medidas obtenidas de los sensores. Una vez detectado un obstáculo por un sensor es necesario incorporar dicha información en las celdas correspondientes del mapa métrico. Puesto que la posición del obstáculo se describe mediante una función de probabilidad dentro del lóbulo del sensor, el correspondiente proceso de actualización del mapa métrico debe realizarse mediante las adecuadas técnicas estadísticas. Algunas de las reglas de actualización más utilizadas son las técnicas basadas en reglas bayesianas [TBB+98] o las técnicas basadas en lógica difusa [GOU96].

Dependiendo de las funciones que se utilicen para el modelado del sensor y para la regla de actualización se implementará un modelo determinado de mapa métrico probabilístico, presentando las distintas alternativas diferentes ventajas e inconvenientes. Un estudio detallado se puede encontrar en el trabajo realizado por Urdiales [Urd99]. En dicho estudio se realiza una exhaustiva comparación entre estos modelos y se propone un nuevo modelo de mapa métrico probabilístico más sencillo y rápido de generar. Este nuevo modelo se basa en la realización de medidas frecuentes en lugar de basarse en complejos modelos de actualización, y será la alternativa escogida en la presente Tesis por ofrecer unas excelentes prestaciones.

El proceso de generación del mapa métrico probabilístico finalmente implementado utiliza las siguientes funciones [UBP+03]:

- **Modelado del sensor:** la función de probabilidad que modela el comportamiento del sensor sonar para determinar la probabilidad de que el obstáculo detectado se encuentre en una determinada posición del lóbulo es la siguiente:

$$P_{ocup}(x,\theta) = \begin{cases} \frac{1}{2d \cdot \tan\beta} & si: \quad x = d \quad ; |\theta| \leq \beta \\ \\ 0 & si: \quad 0 \leq x < d \quad ; |\theta| \leq \beta \end{cases} \tag{4.1}$$

siendo P_{ocup} la probabilidad de ocupación de una determinada posición por un obstáculo, x la distancia de la posición al sensor sonar medida sobre el eje del lóbulo, θ el ángulo formado entre la posición y el eje del lóbulo, d la distancia al obstáculo detectado y 2β la anchura del lóbulo. Esta función se representa en la figura 4.3, y simplemente especifica que todas las posiciones del lóbulo están libres hasta la distancia d y que todas las posiciones del lóbulo están ocupadas con la misma probabilidad a la distancia d. Por motivos de simplicidad y para agilizar el proceso de generación del mapa métrico probabilístico se ha considerado que a la distancia d la incertidumbre del sensor viene determinada por una línea y no por un arco.

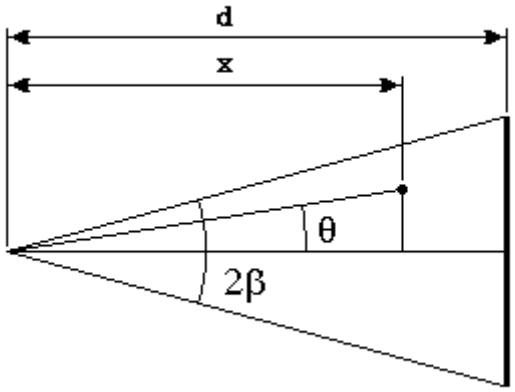

Figura 4.3: Modelado del sensor sonar para la implementación del mapa métrico probabilístico.

- **Regla de actualización:** la probabilidad de ocupación de cada celda del mapa métrico probabilístico es actualizada en función de las lecturas obtenidas de los sensores sonar. Para ello, para cada uno de los sensores sonar se actualizan las celdas contenidas en su lóbulo según la siguiente función:

$$C_i = \begin{cases} C_i + P_{occ} & si: \quad x = d \quad ; |\theta| \leq \beta \\ C_i - P_{free} & si: \quad 0 \leq x < d \quad ; |\theta| \leq \beta \end{cases} \tag{4.2}$$

  siendo C_i la probabilidad de ocupación de la celda i del mapa métrico probabilístico, x la distancia de la celda al sensor sonar medida sobre el eje del lóbulo, θ el ángulo formado entre la celda y el eje del lóbulo, d la distancia al obstáculo detectado, 2β la anchura del lóbulo y P_{occ} y P_{free} dos parámetros empíricos que determinan la variación de la probabilidad para cada celda ocupada y libre respectivamente.

Conforme el agente se va desplazando, cada nuevo conjunto de lecturas obtenidas de los sensores permite actualizar convenientemente el mapa métrico probabilístico generado, por lo que la precisión del mismo irá mejorando con las sucesivas incorporaciones de nuevas lecturas. Así pues, uniendo la función de modelado del sensor y la regla de actualización descritas anteriormente el algoritmo que permite generar adecuadamente el mapa métrico probabilístico es el siguiente:

- Todas las celdas correspondientes a la posición del agente reducen su probabilidad en P_{free}.
- Todas las celdas dentro del lóbulo del sensor que se encuentran entre el sensor sonar y el obstáculo detectado se consideran como celdas libres, por lo que reducen su probabilidad en P_{free}.
- Todas las celdas dentro del lóbulo del sensor que se encuentran a la distancia del obstáculo detectado se consideran como celdas ocupadas, por lo que aumentan su probabilidad en P_{occ}.

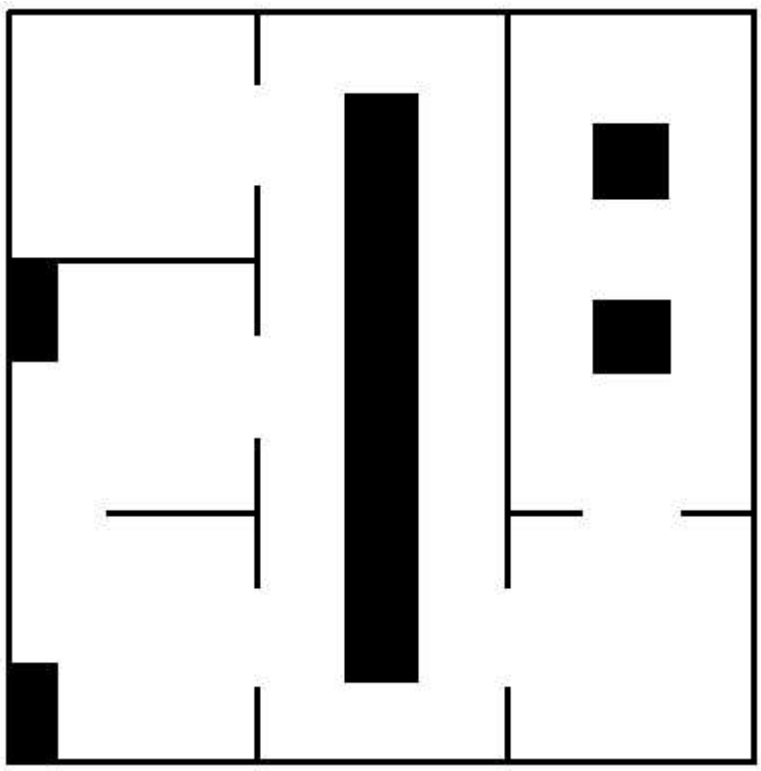

Figura 4.4: Entorno simulado para ser representado mediante un mapa métrico probabilístico.

La principal ventaja de este modelo es la sencillez y rapidez obtenida en la generación y actualización del correspondiente mapa métrico probabilístico. Su precisión es así mismo excelente, pues se basa en la continua integración de las sucesivas medidas de los sensores sonar. Es interesante notar que las celdas correspondientes a la posición del agente reducen su probabilidad en el mapa métrico probabilístico pero no lo anulan, pues aunque es cierto que el agente no puede ubicarse sobre ningún obstáculo del entorno es posible que su localización posea errores, por lo que las celdas que reducen su probabilidad pueden no corresponderse exactamente con la posición física del agente en el entorno.

La figura 4.4 muestra un entorno simulado de unos 15x15 m^2 que va a ser representado mediante un mapa métrico probabilístico de tamaño 256x256 celdas. Para ello el agente va a ir navegando por el entorno mientras integra las diferentes lecturas sonar que captan los sensores, según el método propuesto anteriormente.

La figura 4.5 muestra distintos instantes del proceso de generación del mapa métrico probabilístico con un tamaño de celda $Size\_Cell$ igual a 4 cm para el entorno de la figura 4.4. Puesto que el tamaño del entorno original es de 15x15 m^2 para representarlo completamente con una celda de tamaño 4 cm sería necesario un mapa métrico de tamaño 375x375 celdas. Puesto que el mapa métrico de que se dispone es menor no es posible representar completamente el entorno mediante el mismo.

Para poner de manifiesto la influencia del tamaño de celda $Size\_Cell$ se va a repetir el proceso de generación del mapa métrico probabilístico del entorno 4.4 modificando el tamaño de celda

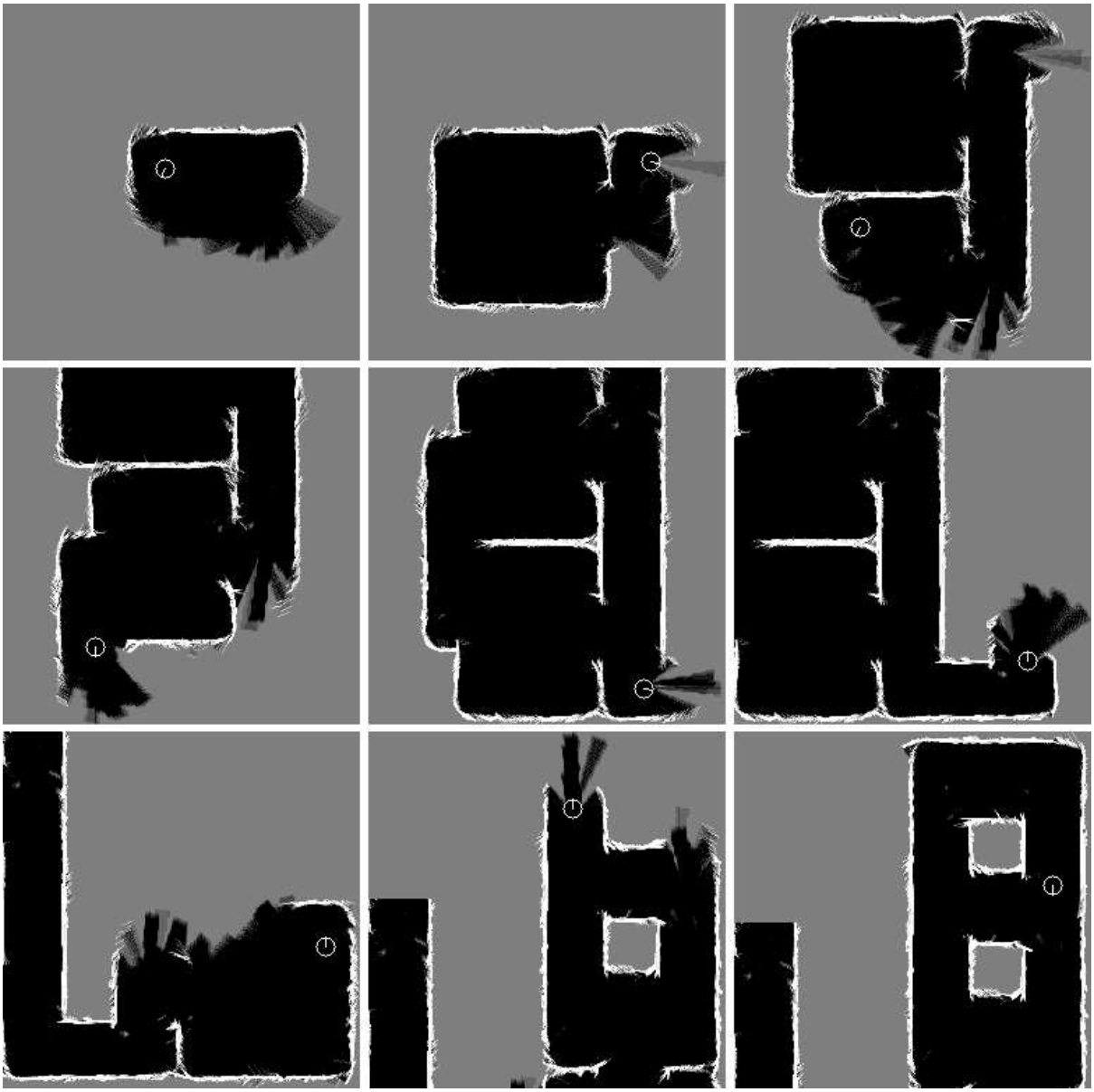

Figura 4.5: Generación del mapa métrico probabilístico mediante el método propuesto con tamaño de celda *Size_Cell* igual a 4 *cm*.

Size_Cell a 6 *cm*. La figura 4.6 muestra distintos instantes de este proceso de generación del mapa métrico probabilístico. En estas circunstancias es necesario un mapa métrico de tamaño 250x250 celdas para representar completamente un entorno de 15x15 m^2, por lo que sí puede ser representado con el mapa métrico de que se dispone.

Como indican las figuras 4.5 y 4.6 la representación de entornos mediante mapas métricos probabilísticos según el método propuesto produce resultados no sólo correctos sino además muy fidedignos al entorno original. Cuanto mayor sea el tamaño de la celda *Size_Cell* mayor es el entorno que puede ser representado mediante un mapa métrico determinado, o dicho de otra forma menor cantidad de datos es necesaria para representar un entorno. Sin embargo, aumentar el tamaño de la celda también disminuye la resolución con la que el mapa métrico es generado, pues

Figura 4.6: Generación del mapa métrico probabilístico mediante el método propuesto con tamaño de celda *Size_Cell* igual a 6 *cm*.

cada celda del mapa métrico representa un área mayor del entorno. Por lo tanto, existe un claro compromiso en la elección del tamaño de la celda de un mapa métrico probabilístico, pues hay que escoger entre la cantidad de datos generada y la resolución obtenida en la representación.

Por último, se van a analizar las prestaciones temporales del proceso de generación del mapa métrico probabilístico. Obviamente estas prestaciones dependen claramente del número de sensores sonar del agente y de los recursos de la máquina empleada para generar el mapa métrico, por lo que se ha repetido la misma prueba para distinto número de sensores sonar y para distintos tipos de máquinas: 8, 16, 24 y 32 sensores, utilizando en cada caso una máquina de altas prestaciones (Pentium 4 - 2.4 GHz - 512 MB RAM), una máquina de prestaciones medias (Pentium 3 - 1.0 GHz - 256 MB RAM) y una máquina de bajas prestaciones (Pentium 3 - 500

Pentium 4 - 2.4 GHz - 512 MB RAM				
	8 sensores	16 sensores	24 sensores	32 sensores
Celdas Libres	$2.21 \pm 0.08\ ms$	$5.02 \pm 0.08\ ms$	$7.86 \pm 0.09\ ms$	$10.66 \pm 0.09\ ms$
Celdas Ocupadas	$0.87 \pm 0.12\ \mu s$	$1.1 \pm 0.3\ \mu s$	$1.4 \pm 0.4\ \mu s$	$1.7 \pm 0.6\ \mu s$
Total	$2.21 \pm 0.08\ ms$	$5.03 \pm 0.08\ ms$	$7.86 \pm 0.09\ ms$	$10.66 \pm 0.09\ ms$
Pentium 3 - 1.0 GHz - 256 MB RAM				
	8 sensores	16 sensores	24 sensores	32 sensores
Celdas Libres	$6.67 \pm 0.19\ ms$	$15.04 \pm 0.19\ ms$	$23.41 \pm 0.22\ ms$	$31.70 \pm 0.14\ ms$
Celdas Ocupadas	$2.9 \pm 0.4\ \mu s$	$4.7 \pm 0.7\ \mu s$	$6.6 \pm 1.0\ \mu s$	$7.75 \pm 0.07\ \mu s$
Total	$6.67 \pm 0.19\ ms$	$15.05 \pm 0.19\ ms$	$23.42 \pm 0.22\ ms$	$31.71 \pm 0.14\ ms$
Pentium 3 - 500 MHz - 192 MB RAM				
	8 sensores	16 sensores	24 sensores	32 sensores
Celdas Libres	$7.23 \pm 0.08\ ms$	$16.38 \pm 0.11\ ms$	$25.50 \pm 0.13\ ms$	$34.66 \pm 0.15\ ms$
Celdas Ocupadas	$3.1 \pm 0.3\ \mu s$	$5.0 \pm 0.7\ \mu s$	$6.9 \pm 1.1\ \mu s$	$8.61 \pm 0.24\ \mu s$
Total	$7.23 \pm 0.08\ ms$	$16.38 \pm 0.11\ ms$	$25.51 \pm 0.13\ ms$	$34.67 \pm 0.15\ ms$

Tabla 4.1: Prestaciones temporales del proceso de generación del mapa métrico probabilístico.

MHz - 192 MB RAM). El tiempo medio de procesamiento empleado en cada una de las etapas del método propuesto se muestra en la tabla 4.1, donde los errores se han calculado sobre un intervalo de confianza del 95%.

Según la tabla 4.1 el proceso de generación del mapa métrico probabilístico es muy rápido, debido sin lugar a dudas a la sencillez del método propuesto. La gran diferencia que existe entre el proceso de actualización de las celdas libres y el proceso de actualización de las celdas ocupadas radica en el hecho de que el número de celdas libres es mucho mayor que el de celdas ocupadas, pues son todas las celdas que se encuentran en el lóbulo de cada sensor hasta la distancia del obstáculo detectado. Incluso en condiciones extremas con 32 sensores y utilizando una máquina de bajas prestaciones el retardo medio del proceso de generación del mapa métrico probabilístico está alrededor de los 35 ms, lo que implica que a una velocidad del agente de unos 300 mm/s se recorre aproximadamente 1 cm antes de incorporar las nuevas medidas de los sensores en el mapa métrico. En el sistema desarrollado el agente utilizado posee 8 sensores, por lo que incluso utilizando una máquina de bajas prestaciones el retardo medio está alrededor de los 7 ms, es decir, un desplazamiento del agente de unos 2 mm a una velocidad de 300 mm/s.

Estos resultados ponen de manifiesto que el proceso de generación del mapa métrico probabilístico es capaz de operar prácticamente en tiempo real, actualizando continuamente todas las lecturas de los sensores en la representación del mapa métrico.

2.2 Módulo *MapMetric*

La Capa de Representación se va a implementar mediante el módulo *MapMetric*, encargado de generar y actualizar el mapa métrico probabilístico del entorno donde se encuentra el agente según el esquema propuesto en el apartado 2.1.

La implementación del módulo *MapMetric* se ha realizado siguiendo un esquema de *buffer* circular. Esto significa que cuando la memoria reservada para la generación del mapa métrico se llene por completo el mapa seguirá generándose y actualizándose, descartando la región del entorno más antigua que se tiene representada para permitir almacenar la nueva representación que se está generando. Este modo de operación se puede observar en la figura 4.5, donde la parte más lejana del mapa métrico va desapareciendo para permitir la representación de nuevas zonas más cercanas conforme se van necesitando. Con este modo de operación se garantiza que la memoria reservada para el mapa métrico nunca se va a agotar y que la región del entorno representada en el mapa métrico es siempre aquella en la que se encuentra el agente.

El módulo *MapMetric* también es responsable de activar los procesos de planificación presentes en el sistema cuando se detectan cambios significativos en el entorno. Para ello, se computa continuamente el número de celdas del mapa métrico que pasan de libres a ocupadas o viceversa, y cuando este valor supera unos ciertos umbrales se considera que el entorno ha sido modificado sustancialmente y deben realizarse nuevos procesos de planificación sobre el mismo. Este proceso será detalladamente descrito en los apartados 2.3 y 3.4 del capítulo 6.

El interfaz que define los datos de entrada y salida que este módulo comparte con el resto de módulos es el siguiente:

- **Datos de entrada:**
  - **Posición del agente:** generada por el módulo *IFRobot*.
  - **Lectura de los sensores:** generadas por el módulo *IFRobot*.
- **Datos de salida:**
  - **Mapa métrico:** utilizado por los módulos *IFRobot*, *MapTopo*, *PathPlanning* y *IFUser*.

3 Implementación de la Capa de Comando

Para que el usuario pueda especificar las tareas a desarrollar por el agente es necesario implementar un interfaz adecuado entre ambos, que además permita mostrar algún tipo de información sobre el resultado de dichas tareas. Sería deseable que el interfaz con el usuario fuese lo más amigable y potente posible para facilitar el uso del sistema, por lo que el empleo de algún tipo de herramienta visual es altamente recomendado.

La Capa de Comando implementa este interfaz con el usuario, pero también controla el estado del sistema mediante la activación y desactivación selectiva de los módulos adecuados para que el agente desarrolle la tarea especificada. Esta activación y desactivación selectiva de los distintos módulos se realiza gracias a los mecanismos de sincronización suministrados por la arquitectura de control *DLA*. Por lo tanto, se puede decir que esta capa desarrolla también el papel de

controlador que poseen algunas de las arquitecturas de control híbridas existentes (*3T*, *TCA*, *LAAS* y *BERRA*).

3.1 Módulo *IFUser*

La Capa de Comando se va a implementar mediante el módulo *IFUser*, utilizando para ello una herramienta visual. Las funciones que debe desarrollar este módulo son las siguientes:

- **Interfaz de usuario:** responsable de capturar los comandos introducidos por el usuario y de mostrar la información del sistema.

  Los comandos que pueden ser especificados indican el nivel de navegación deseado:

  - **Navegación Reactiva:** navegación hacia el destino final mediante la evitación los obstáculos fijos y/o móviles del entorno.
  - **Navegación Planificada:** navegación hacia el destino final mediante la planificación de caminos y la evitación de los obstáculos fijos y/o móviles del entorno.
  - **Navegación Topológica:** navegación hacia el destino final mediante la planificación de rutas, la planificación de caminos y la evitación de los obstáculos fijos y/o móviles del entorno.

  La información del sistema muestra la representación del entorno que posee el agente así como los planes calculados en el desarrollo de las diferentes tareas:

  - **Posición del agente:** ubicación del agente en el entorno.
  - **Mapa métrico:** representación métrica del entorno.
  - **Mapa topológico:** representación topológica de las regiones del entorno.
  - **Región de procesamiento del mapa métrico:** región del mapa métrico utilizada para calcular el camino libre de obstáculos hasta la siguiente región de la ruta.
  - **Camino:** camino libre de obstáculos a seguir para alcanzar el destino final.
  - **Ruta:** ruta de regiones a atravesar para alcanzar el destino final.
  - **Destino final:** destino que se pretende alcanzar.
  - **Próximo destino intermedio:** próximo destino intermedio a visitar para seguir el camino.

- **Controlador:** responsable de la activación y desactivación selectiva de módulos según el comando introducido por el usuario.

  Los distintos módulos que son activados en función del comando especificado son:

  - **Navegación Reactiva:** se activan los módulos *IFRobot*, *Navigation* y *MapMetric*.
  - **Navegación Planificada:** se activan los módulos *IFRobot*, *Navigation*, *MapMetric* y *PathPlanning*.

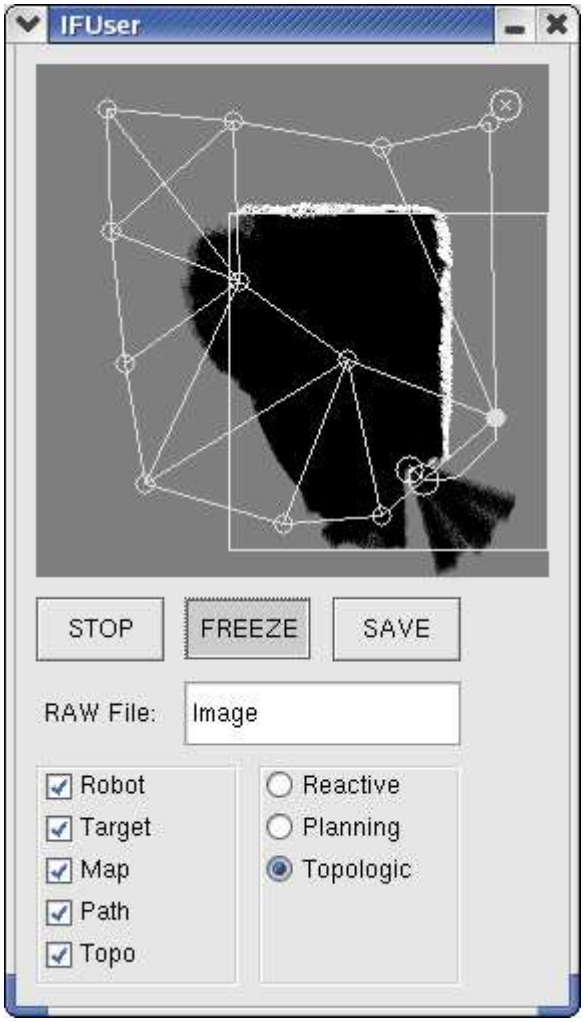

Figura 4.7: Interfaz de usuario del módulo *IFUser*.

– **Navegación Topológica:** se activan los módulos *IFRobot*, *Navigation*, *MapMetric*, *PathPlanning* y *MapTopo*.

El módulo *IFUser* se ha desarrollado con el lenguaje de programación visual *GTK* para *Linux*, aunque una implementación con cualquier otro lenguaje soportado por la arquitectura de control *DLA* es posible. La figura 4.7 muestra el aspecto del interfaz visual diseñado para el módulo *IFUser*, donde se puede apreciar la información contenida en el sistema [4]. Se ha añadido la capacidad de salvar en un archivo gráfico la información mostrada en el área de visualización del módulo *IFUser*, funcionalidad muy útil para la depuración y el análisis de los resultados obtenidos durante la operación del sistema.

El interfaz que define los datos de entrada y salida que este módulo comparte con el resto de módulos es el siguiente:

[4]Una descripción detallada del uso del interfaz de usuario *IFUser* se puede encontrar en el apartado 3.4 del apéndice A.

- **Datos de entrada:**
  - **Posición del agente:** generada por el módulo *IFRobot*.
  - **Mapa métrico:** generado por el módulo *MapMetric*.
  - **Mapa topológico:** generado por el módulo *MapTopo*.
  - **Ruta calculada:** generada por el módulo *MapTopo*.
  - **Camino calculado:** generado por el módulo *PathPlanning*.
  - **Próximo destino intermedio:** generado por el módulo *PathPlanning*.
- **Datos de salida:**
  - **Comando especificado:** utilizado por los módulos *IFRobot*, *Navigation*, *MapMetric*, *MapTopo* y *PathPlanning*.
  - **Destino final:** utilizado por los módulos *Navigation*, *MapTopo* y *PathPlanning*.

4 Conclusiones

En el presente capítulo se ha descrito la implementación de algunas de las capas necesarias en el sistema para garantizar la correcta operación del mismo.

La Capa Física es la responsable de incorporar todos aquellos comportamientos que acceden y controlan directamente la plataforma robótica. Desarrolla funciones básicas en el sistema, como pueden ser acceder a los recursos de la plataforma robótica, unificar las unidades de las lecturas obtenidas de los sensores, fusionar las distintas lecturas de los sensores, transformar el sistema de coordenadas del agente al sistema de coordenadas utilizado en el sistema, corregir los posibles errores de localización del sistema de odometría, traducir adecuadamente los comandos de movimiento generados en el sistema, o desarrollar un sistema de alarma que detenga la plataforma robótica ante cualquier riesgo de colisión inminente.

La Capa de Representación es la responsable de generar un modelo del entorno por el que el agente desarrolla su actividad, modelo que será posteriormente utilizado por los procesos de planificación del sistema. Debido a su sencillez y buenas prestaciones se ha escogido una representación mediante mapas métricos probabilísticos, utilizando en su generación un método sencillo que muestra unas excelentes características temporales.

Por último, la Capa de Comando es la responsable de capturar las distintos comandos introducidos por el usuario, a la vez que representa la información básica del estado del sistema. Se ha desarrollado mediante una aplicación visual con un sencillo interfaz capaz de especificar comandos para los distintos niveles de navegación desarrollados, a la vez que muestra información sobre la posición del agente, el modelo actual del entorno generado, el camino y la ruta calculados en el proceso de planificación y la representación topológica del entorno.

Capítulo 5

Capa de Navegación

Una vez implementadas la Capa Física, la Capa de de Representación y la Capa de Comando, necesarias para el correcto funcionamiento del sistema, se va a abordar el diseño de la Capa de Navegación, responsable de desarrollar los comportamientos reactivos de navegación con evitación de obstáculos que implementa el nivel de Navegación Reactiva del sistema.

Este capítulo se ha organizado como se indica a continuación. El apartado 1 realiza una introducción a la navegación reactiva y a los principales paradigmas existentes, describiendo las ventajas e inconvenientes de cada uno de ellos. El apartado 2 analiza en profundidad el paradigma del razonamiento basado en casos, procediendo al diseño de un sistema de navegación reactiva basado en dicho esquema. El apartado 3 describe la implementación de la Capa de Navegación del sistema. Por último, en el apartado 4 se describen las principales conclusiones extraídas a lo largo del presente capítulo.

1 Introducción

La Capa de Navegación es la responsable de implementar el nivel de Navegación Reactiva del sistema, el cual desarrolla los niveles inferiores incluidos en el grupo de Navegación Local de la Jerarquía de Navegación, tal y como se ha descrito en el apartado 4.1 del capítulo 3. La navegación reactiva implementada por la Capa de Navegación constituye el comportamiento básico que permite alcanzar un destino final evitando las colisiones con los posibles obstáculos fijos y/o móviles del entorno. Como todo comportamiento reactivo, utiliza únicamente la información instantánea de que dispone el sistema, asociando directamente las lecturas de los sensores con las acciones a desarrollar por el agente. Al realizar una asociación directa entre percepción y acción carente de cualquier planificación, los sistemas reactivos pueden proporcionar respuestas rápidas ante los distintos estímulos del entorno. A pesar de sus muy destacables ventajas, el esquema reactivo posee así mismo algunos inconvenientes, aunque estos últimos se pueden resolver empleando un esquema híbrido que integre adecuadamente respuestas reactivas y algún tipo de planificación para aumentar la eficiencia del sistema. Esta es la solución adoptada en la presente Tesis, por lo que el diseño del sistema reactivo realizado en este capítulo será combinado

posteriormente con los adecuados procesos de planificación diseñados en el capítulo 6.

Existen distintas alternativas a la hora de implementar esquemas de navegación reactivos, si bien la gran mayoría de ellos se pueden clasificar fundamentalmente en dos grandes grupos:

- **Esquemas analíticos:** incluyen todas aquellas alternativas que intentan modelar analíticamente el comportamiento de navegación. En este tipo de alternativas suele ser bastante complejo caracterizar el comportamiento de navegación completo, pues requieren de un conocimiento explícito de todas las circunstancias posibles que se puedan presentar en el entorno. Suelen ser esquemas bastante sensibles a los distintos parámetros que poseen, los cuales se pueden ajustar para algunas situaciones específicas pero son difíciles de optimizar para operar en todas las posibles circunstancias con las que se pueda encontrar el agente.
- **Esquemas basados en aprendizaje:** incluyen todas aquellas alternativas que no realizan un modelado explícito del comportamiento de navegación, sino más bien un aprendizaje de los distintos comportamientos que el sistema debe asimilar para afrontar con éxito las posibles circunstancias que pueden presentarse durante la navegación. Suelen ser sistemas más complejos de implementar que requieren de un período de entrenamiento, aunque incorporan de forma implícita las distintas particularidades de la navegación. Por estos motivos constituyen una alternativa muy flexible para implementar esquemas de navegación, al ser posible entrenar distintos comportamientos para resolver las distintas circunstancias con las que se pueda encontrar el agente.

Si bien las alternativas analíticas suelen ser bastante más sencillas de implementar, las alternativas basadas en aprendizaje pueden llegar a desarrollar esquemas de navegación más flexibles y eficientes. En los siguientes apartados se describen con mayor detalle cada uno de estos paradigmas.

1.1 Esquemas analíticos

Los esquemas analíticos de navegación reactiva pretenden determinar algún tipo de función analítica para modelar explícitamente el comportamiento de navegación a implementar. De esta forma se pueden obtener los comandos de movimiento necesarios que debe ejecutar el agente en función de la situación particular en la que se encuentre, determinada por su posición, la de los obstáculos en el entorno y la del destino final a alcanzar. Existen un gran número de esquemas analíticos de navegación reactiva, si bien todos ellos comparten los inconvenientes de este tipo de soluciones: poca flexibilidad para obtener comportamientos generales que sean válidos para un amplio rango de circunstancias particulares, así como una considerable sensibilidad del comportamiento de navegación respecto a los distintos parámetros del esquema.

Se presenta a continuación una breve descripción de algunos de los esquemas analíticos más destacados que existen.

1.1.1 Campos potenciales

El esquema de los Campos Potenciales (*Potential Fields*) [Kha86] constituye quizás el esquema analítico de navegación reactiva más conocido y empleado, debido principalmente a su sencillez y elegancia a la hora de determinar la función analítica que modela el comportamiento de navegación. Esta sencillez permite aplicar directamente este esquema al problema de la navegación reactiva en entornos dinámicos no estructurados, proporcionando respuestas rápidas ante la presencia de obstáculos. El esquema de los Campos Potenciales considera al agente como una partícula libre que se desplaza en un campo potencial determinado por la estructura del entorno. Para obtener este campo potencial se considera que los obstáculos del entorno se comportan como cargas con la misma polaridad que el propio agente, por lo que generan fuerzas repulsivas que lo alejan de los mismos, mientras que el destino final se comporta como una carga de distinta polaridad a la del agente, por lo que genera una fuerza atractiva que lo dirige hacia el mismo. De forma genérica, pues, se puede calcular el campo potencial $U(x)$ en todo punto del espacio como [Lat91]:

$$U(x) = U_{atrac}(x) + U_{rep}(x) \tag{5.1}$$

donde $U_{atrac}(x)$ es el campo potencial atractivo asociado con el destino final y $U_{rep}(x)$ es el campo potencial repulsivo asociado con los distintos obstáculos del entorno. Una vez determinado el campo potencial simplemente hay que seguir el gradiente negativo de dicho campo para alcanzar el destino final evitando los obstáculos del entorno, lo cual se realiza mediante un proceso iterativo que calcula en cada posición del espacio donde se encuentre el agente la fuerza resultante que se ejerce sobre el mismo. Esta fuerza resultante se obtiene como:

$$\vec{F}(x) = -\vec{\nabla}U(x) = -\vec{\nabla}U_{atrac}(x) - \vec{\nabla}U_{rep}(x) = \vec{F}_{atrac}(x) + \vec{F}_{rep}(x) \tag{5.2}$$

No obstante, a pesar de su extremada sencillez, el esquema de los Campos Potenciales presenta algunos inconvenientes importantes [KB91a]:

- Es muy sensible a la existencia de mínimos locales en el campo potencial calculado, posibilitando que el agente quede atrapado indefinidamente en aquellas posiciones que representen un mínimo local en el mismo. Para intentar resolver este inconveniente han aparecido nuevos esquemas de Campos Potenciales que intentan evitar la existencia de mínimos locales [GSV94, KC95] o que intentan sacar al agente de cualquier mínimo local una vez ha sido atrapado en el mismo [Ark98, MdLS00]. Es importante notar que si el esquema de los Campos Potenciales se integra dentro de un esquema híbrido el problema de los mínimos locales puede ser resuelto de forma sencilla y eficiente mediante la adecuada planificación realizada por la capa deliberada, la cual puede calcular la ruta a seguir por el agente y dirigir al sistema reactivo a lo largo de esa ruta, evitando los posibles mínimos locales existentes en el entorno.

- Posee una fuerte dependencia de los parámetros del esquema, que son difíciles de ajustar para todas las posibles circunstancias de navegación. Esta fuerte dependencia origina la aparición de comportamientos indeseados como oscilaciones en entornos con alta densidad de obstáculos o imposibilidad de atravesar obstáculos cercanos entre sí.

1.1.2 Histograma del campo vectorial

El esquema basado en el histograma del campo vectorial (*Vector Field Histogram*) [BK91] es un esquema analítico basado en la representación de la densidad de obstáculos del entorno mediante un histograma que depende de la orientación del agente. Mediante este histograma se intentan localizar las distintas regiones del entorno que presentan menor densidad de obstáculos, dirigiendo al agente hacia las mismas durante el proceso de navegación. Este mismo esquema ha sido posteriormente mejorado para tener en cuenta en el proceso de navegación el tamaño del robot y para reducir las oscilaciones del método original [UB98]. Posteriores modificaciones incluyen además un planificador que permite aumentar la eficiencia del camino recorrido y evitar los posibles mínimos locales [UB00]. El esquema basado en el histograma del campo vectorial proporciona buenas resultados incluso a altas velocidades de navegación, aunque también presenta algunos inconvenientes. En primer lugar, no es capaz de dirigir intencionadamente al agente hacia un obstáculo una vez detectado, por lo que no proporciona buenos resultados en entornos muy densos. Además, es bastante sensible a los distintos parámetros que utiliza el algoritmo, por lo que una mala elección de los mismos puede proporcionar resultados poco óptimos. Por último, el esquema precisa la construcción de un mapa del entorno para ser capaz de extraer el histograma del campo vectorial, por lo que algunos autores consideran que este esquema no se puede considerar como puramente reactivo, al depender de percepciones anteriores y no únicamente de la percepción actual del entorno.

1.1.3 Bandas elásticas

El esquema de las bandas elásticas (*Elastic Band*) [QK93] se basa en calcular una trayectoria sencilla que conecte la posición actual del agente con el destino final, y una vez calculada deformarla en función de unas fuerzas artificiales que se crean a partir de la distribución de obstáculos presentes en el entorno. Una banda elástica se puede definir como un camino deformable libre de colisiones, cuya forma inicial viene determinada por un planificador. El camino final resultante es por lo general bastante suave y eficiente, por lo que este esquema es bastante adecuado para operar con entornos parcialmente conocidos, donde se pueda obtener mediante un proceso de planificación una primera trayectoria que posteriormente se irá deformando. No obstante, los resultados proporcionados en entornos desconocidos o altamente dinámicos no son lo suficientemente apropiados, puesto que una deformación excesiva de la trayectoria original puede llegar a hacer inoperativo el camino originalmente propuesto, si bien algunos estudios posteriores han intentado mejorar el esquema en dichas circunstancias [BK00]. El principal inconveniente de este esquema radica en la sensibilidad del algoritmo con los distintos parámetros del mismo, dificultando un ajuste óptimo para obtener un buen comportamiento en distintas circunstancias.

1.1.4 Ventana dinámica

El esquema de la ventana dinámica (*Dynamic Window Approach*) [FBT97] trata de incorporar la dinámica del agente al problema de la navegación reactiva. Para ello formula el problema a resolver como una optimización en el espacio de las velocidades, imponiendo una serie de restricciones derivadas de las limitaciones de movimiento del agente. Posteriores estudios han extendido su aplicación para agentes holonómicos y han tratado de resolver el problema de los mínimos locales mediante la inclusión de un planificador [BK99]. No obstante, el principal inconveniente de este esquema vuelve a encontrarse en la dificultad de ajuste de los parámetros del sistema de forma general para garantizar un funcionamiento adecuado bajo un amplio abanico de circunstancias diferentes.

1.1.5 Diagrama de proximidad

El esquema basado en el diagrama de proximidad (*Nearness Diagram*) [App00] es muy similar en su concepción al esquema basado en el histograma del campo vectorial, utilizando una representación polar de la densidad de obstáculos en el entorno y escogiendo a partir de dicha información la mejor dirección de avance para el agente. Sus principales ventajas radican en una mejor operación en entornos muy densos, al permitir dirigirse hacia los obstáculos del entorno, y en utilizar un único parámetro en el algoritmo, aunque sigue siendo necesario ajustarlo convenientemente. Posteriores estudios han incorporado un planificador para hacer más eficiente la operación del esquema [MMSA01]. Es interesante señalar que al igual que en el esquema basado en el histograma del campo vectorial este esquema precisa la construcción de un mapa del entorno para poder operar, por lo que algunos autores no lo catalogan como un esquema puramente reactivo.

1.2 Esquemas basados en aprendizaje

Los esquemas basados en aprendizaje tratan de imitar comportamientos complejos sin necesidad de utilizar modelos analíticos que los describan, constituyendo una alternativa eficiente en la resolución de problemas difíciles de caracterizar, como es el caso de la navegación. Frente a la rigidez de los esquemas analíticos que pretenden modelar el comportamiento del sistema bajo todas las posibles circunstancias, los esquemas basados en aprendizaje poseen una mayor flexibilidad al particularizar la respuesta del sistema a los distintos escenarios que se puedan presentar, basándose siempre en el conocimiento y la experiencia acumulada en el sistema. De esta forma es posible incorporar distintos comportamientos de navegación, escogiendo siempre el más adecuado en función de las circunstancias particulares del entorno. Además, debido al sistema de aprendizaje es posible incorporar restricciones cinemáticas durante la etapa de entrenamiento, facilitando la adaptación del esquema de navegación a distintos agentes con diferentes características dinámicas.

Existen distintos paradigmas a la hora de implementar sistemas basados en aprendizaje, presentándose en los siguientes apartados una breve descripción de las alternativas más destacadas [BANS02].

1.2.1 Razonamiento basado en reglas

El paradigma del razonamiento basado en reglas intenta representar explícitamente el conocimiento a adquirir mediante un conjunto de reglas heurísticas [Cla85]. Dichas reglas pueden modificarse dinámicamente, por lo que el aprendizaje se garantiza actualizando el conjunto de reglas que posee el sistema. Este paradigma basa su operación en un modelo deductivo, por lo que requiere una comprensión considerable del conocimiento que se quiere adquirir. Es por ello por lo que su campo de aplicación se extiende fundamentalmente a aquellos problemas bien estructurados que pueden ser descritos con relativa facilidad por un conjunto de reglas bien definidas. Sin embargo, este tipo de paradigma no proporciona buenos resultados bajo determinadas circunstancias [IWB92]:

- No es apropiado en aquellos dominios de aplicación poco conocidos donde resulta muy complejo extraer un conjunto de reglas que modelen el comportamiento del sistema.
- Tampoco es adecuado cuando el conjunto de reglas es muy complejo y posee un excesivo número de excepciones, pues en dichas circunstancias se pueden degradar considerablemente las prestaciones de operación del sistema.
- No suele proporcionar soluciones adecuadas cuando se pretenden resolver circunstancias excepcionales e imprevistas no contenidas en el conocimiento del sistema.
- Conducen a sistemas difíciles de mantener cuando el conjunto de reglas es elevado, pues su interrelación puede llegar a ser extremadamente compleja.

1.2.2 Redes neuronales

El paradigma de las redes neuronales pretende inducir un conocimiento generalizado del comportamiento del sistema a partir de la observación de diversas experiencias. Este conocimiento se almacena en una estructura de neuronas interconectadas entre sí, caracterizadas por una serie de parámetros que determinan su operación [Lip87]. Las neuronas son unidades simples de procesamiento, y los parámetros que las caracterizan se obtienen tras un proceso de aprendizaje que intenta buscar un conjunto de parámetros aplicable a todas las muestras disponibles. Las redes neuronales presentan una alternativa interesante en la implementación de sistemas inteligentes basados en aprendizaje, y tienen un amplio campo de aplicación. No obstante, también presentan una serie de limitaciones para determinado tipo de sistemas [BANS02]:

- Suelen requerir un elevado número de patrones de entrenamiento para generalizar el comportamiento del sistema.

- No son adecuadas en aquellas circunstancias donde no todo el comportamiento del sistema se puede generalizar con un único modelo.

1.2.3 Razonamiento basado en casos

El paradigma del razonamiento basado en casos, también conocido como ***CBR*** (*Case Based Reasoning*), intenta resolver nuevos problemas reusando y adaptando las soluciones utilizadas al resolver problemas anteriores [RS89]. El conocimiento está formado por el conjunto completo de experiencias que describen el comportamiento del sistema, sin inducir ningún tipo de conocimiento generalizado a partir de ellas. Cada una de estas experiencias se representa en el sistema por medio de un **caso**, que se puede definir como un entidad abstracta que realiza una descripción completa del problema resuelto y de la solución adoptada en su resolución. Todos los casos son almacenados en lo que se conoce como **base de casos**, la cual contiene el conocimiento completo que posee el sistema mediante una descripción de todas las experiencias utilizadas en la resolución de problemas anteriores. Así pues, para resolver nuevos problemas se buscan experiencias similares en el conocimiento del sistema, y se adaptan dichas soluciones para resolver los nuevos problemas descritos.

A pesar de la relativa juventud de este paradigma los sistemas *CBR* manifiestan unas capacidades bastante prometedoras, por lo que esta alternativa está experimentando últimamente un afloramiento similar al que experimentaron en su día las redes neuronales. Entre las principales ventajas que presenta este nuevo paradigma se pueden destacar las siguientes [AP94, Wat95, BANS02]:

- No requiere un modelado explícito del dominio al que se intenta aplicar, simplemente una serie de casos que describan las experiencias acumuladas al resolver problemas anteriores.
- No realiza una generalización global del dominio sobre el que se aplica, constituyendo una alternativa muy adecuada en aquellas aplicaciones donde coexisten distintas generalizaciones y métodos para alcanzar una solución. La coexistencia de distintas generalizaciones permite la aplicación de soluciones muy distintas a situaciones aparentemente similares pero que no tienen ninguna relación entre sí, aumentando la flexibilidad en la resolución de problemas.
- Su implementación requiere únicamente la adecuada descripción de las distintas experiencias en forma de casos, debiendo seleccionar para ello las características más significativas que se desean almacenar de cada experiencia.
- Permite la aplicación de todas las técnicas de búsqueda en bases de datos para manejar eficientemente un volumen de información masivo.
- Posibilita un aprendizaje incremental y un sencillo mantenimiento del sistema, al ir incorporando nuevos casos que vayan describiendo nuevas experiencias.
- Frente a otros esquemas que requieren un proceso de entrenamiento exhaustivo este esquema puede ser aplicado inmediatamente, incluso con un muy reducido número de casos

que integren un conocimiento limitado en el sistema. La incorporación gradual de nuevas experiencias irá aumentando progresivamente el conocimiento que posee el sistema.

- En todo momento es posible extraer y analizar el conocimiento completo que posee el sistema accediendo a la base de casos. Esto facilita la depuración del sistema en la fase de desarrollo, al conocer exactamente qué solución es aplicada a cada nuevo problema que se presenta. Además, permite un procesamiento de la base de casos para introducir un mayor nivel de abstracción en el conocimiento que posee el sistema o algún tipo de optimización de la experiencia acumulada.

2 Esquema de navegación mediante *CBR*

Debido a su evidente potencial y versatilidad en el desarrollo de sistemas inteligentes, el paradigma *CBR* será el esquema utilizado para implementar el esquema de navegación reactivo desarrollado en la presente Tesis. En este apartado se realiza un análisis más profundo de este paradigma, para posteriormente diseñar el sistema de navegación reactivo propuesto.

2.1 Introducción

El paradigma *CBR* se basa en los estudios de Roger Schank y Robert Abelson realizados en 1977, donde propusieron una teoría para la resolución de problemas en el ser humano basados en el recuerdo de experiencias pasadas [SA77]. Roger Schank continuó explorando esta teoría hasta que elaboró un modelo cognitivo sobre el aprendizaje y la influencia de experiencias pasadas en la resolución de problemas en el ser humano [Sch82]. Este nuevo modelo se basaba en la teoría de la Memoria Dinámica, la cual relacionaba el recuerdo, la comprensión, la experiencia y el aprendizaje en el ser humano, caracterizado este último por poseer una memoria que varía dinámicamente en función de la experiencia acumulada. Posteriores estudios ampliaron el modelo cognitivo de Roger Schank incluyendo el razonamiento por analogía, la formación de conceptos y el aprendizaje desde el punto de vista de la filosofía y la psicología en el ser humano [Gen83, Car83, And83, RH84, Car86, Ros89].

El primer sistema basado en el paradigma *CBR* que se implementó fue *CYRUS* [Kol83] a principio de los años 80, desarrollado por el propio grupo de Roger Schank en la *Universidad de Yale*, y que constituyó la base para un gran número de sistemas: *MEDIATOR* [Sim85], *PERSUADER* [Syc88], *CHEF* [Ham89] y *JULIA* [Hin92]. A mediados de los 80 comenzaron a proliferar numerosos sistemas desarrollados por otros grupos de investigación, como *PROTOS* [PB86] y *GREBE* [Bra91] en la *Universidad de Texas*, *HYPO* [Ash91] y *CABARET* [SR92] en la *Universidad de Massachusetts* y *CASEY* [Kot89] en el *MIT*, entre otros. En Europa la investigación en el paradigma *CBR* fue algo más tardía, aunque a finales de los 80 comenzaron igualmente a desarrollarse numerosos sistemas, como *REFINER* [SS88], *MOLTKE* [Alt89], *PATDEX* [RW91], *CREEK* [Aam91] y *IULIAN* [Oeh92], entre otros. Actualmente existen una gran cantidad de sistemas basados en el paradigma *CBR*, con campos de aplicación tan diversos como la diagno-

sis médica, el asesoramiento legal, el mantenimiento de equipos, el control del tráfico aéreo, el diseño *VLSI* (*Very Large Scale Integration*), la planificación de tareas, la inversión bursátil, ...

El paradigma *CBR* ya ha sido aplicado con anterioridad al problema de la navegación. Algunos sistemas han pretendido abordar la navegación en entornos estáticos [BA95, FL95], mientras que otros han intentado abordar la navegación en entornos dinámicos [HV95]. Este último sistema se basa en la utilización de una representación topológica del entorno conocida a priori, por lo que no se la modificación del entorno durante el proceso de navegación salvo que se reorganice regularmente el mapa topológico empleado. Otros esquemas distintos emplean un proceso de planificación global basado en representaciones métricas de tipo probabilístico [Kru03], pero son aplicables únicamente en presencia de numerosos obstáculos de un tamaño considerable. Algunos esquemas alternativos integran la información adquirida del entorno durante una ventana de tiempo predeterminada [RS93, LA01].

Ninguna de estos esquemas puede considerarse puramente reactivo, pues no se basan en la información instantánea que el agente capta del entorno. Algunos de ellos utilizan representaciones del entorno, otros emplean algún tipo de planificación y algunos integran datos de la memoria pasada captada del entorno. La novedad que aporta la presente Tesis radica en el desarrollo de un sistema de navegación puramente reactivo mediante el paradigma *CBR*, válido para entornos dinámicos no estructurados. El uso de este paradigma conlleva algunas ventajas importantes:

- Posibilita el aprendizaje y la adaptación de la Capa de Navegación del agente a prácticamente cualquier situación que se presente en el entorno. Así pues, permite modificar la estrategia de navegación en función de las circunstancias particulares que se estén tratando.
- La cinemática del agente se encuentra implícitamente incorporada en el sistema de aprendizaje, por lo que el esquema de navegación es fácilmente adaptable a otros agentes con distintas características dinámicas.
- Permite trabajar en prácticamente cualquier entorno, realizando un adecuado entrenamiento del sistema para capturar las características de navegación particulares de cada entorno.

Se procede a continuación a realizar el diseño detallado del sistema de navegación propuesto mediante el paradigma *CBR*, analizando sus principales características y las alternativas más importantes existentes en su implementación.

2.2 Representación del caso

El paradigma *CBR* se basa en la reutilización de la experiencia para la resolución de problemas, experiencia que se almacena en el sistema mediante un conjunto de **casos**. Se puede definir un caso como una unidad de conocimiento contextualizada que describe una experiencia pasada [Wat95]. Un caso debe contener como mínimo una descripción del problema resuelto y de la solución adoptada en dicha resolución, aunque algunos sistemas incorporan además el resultado

obtenido con la solución adoptada como parte del caso [Sim85]. Si bien no existe un consenso generalizado sobre la información que debe contener un caso se considera que esa información debe permitir una adecuada funcionalidad del sistema y debe ser fácil de adquirir [Kol93]. De cualquier forma, la información contenida en la descripción de cada caso constituye la experiencia que posteriormente el sistema reusará para resolver situaciones similares.

Una adecuada representación del conocimiento a almacenar en forma de caso es un factor determinante a la hora de implementar un sistema mediante el paradigma *CBR*. Por motivos computacionales no suele ser eficiente incluir un volumen excesivo de información en cada caso, por lo que es necesario seleccionar un conjunto de rasgos significativos que describan de la forma más amplia posible la experiencia que se quiere almacenar.

El sistema que se desea implementar sobre el paradigma *CBR* pretende desarrollar un sistema de navegación reactiva. En este tipo de sistemas la única información disponible es la que se percibe del entorno de forma inmediata, formada básicamente por la posición actual del agente y por la situación de los obstáculos que se encuentran dentro del rango de detección del agente. Puesto que la navegación reactiva trata de dirigir al agente hacia un destino final desde la posición actual esquivando los obstáculos próximos del entorno, la información que describe completamente un determinado problema es la siguiente:

- **Dirección al destino:** dirección en la que se encuentra el destino final respecto a la dirección de avance del agente.
- **Obstáculos del entorno:** situación de los obstáculos próximos del entorno que se encuentran dentro del rango de detección del agente.

Por otra parte, en un sistema de navegación reactiva la información que describe la solución adoptada para resolver un determinado problema es la siguiente:

- **Dirección de avance:** dirección adoptada por el agente para alcanzar el destino final evitando los obstáculos próximos del entorno.

Combinando adecuadamente la información que describe un determinado problema con la información que describe la solución adoptada en la resolución de dicho problema queda totalmente caracterizada la experiencia que se quiere representar. Por lo tanto, para el sistema de navegación reactiva que se pretende desarrollar una experiencia queda completamente caracterizada mediante un caso en forma de vector con las siguientes componentes, también conocidas como **características**:

$$caso = [\alpha_{dest}, S_1, S_2, .., S_N, \alpha_{dir}] \tag{5.3}$$

donde α_{dest} representa la dirección en la que se encuentra el destino final respecto a la dirección de avance del agente, S_i representa la lectura del sensor i, N representa el número de sensores

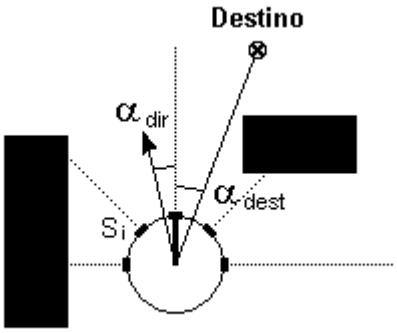

Figura 5.1: Representación del caso para el sistema *CBR* propuesto.

que posee el agente, y α_{dir} representa la dirección de avance del agente propuesta como solución. La figura 5.1 muestra la definición de los distintos parámetros involucrados en la representación del caso.

2.3 Estructura de la base de casos

Se conoce como **base de casos** el conjunto de casos almacenados en el sistema, el cual contiene toda la experiencia y conocimiento del mismo. La estructura empleada en la base de casos para almacenar y acceder a los distintos casos tiene una influencia directa en las prestaciones finales del sistema implementado, y debe equilibrar un almacenamiento con una elevada riqueza semántica y un método de búsqueda de los casos rápido y sencillo. Los principales paradigmas para establecer la estructura de la base de casos en un sistema *CBR* son [SMCR+97]:

- **Estructura plana:** la base de casos se limita a una colección de casos independientemente almacenados. Constituye la alternativa más sencilla de implementar, manifestando algunas ventajas y algunos inconvenientes. Entre las ventajas cabe destacar que garantiza que el proceso de búsqueda siempre devuelva el caso más parecido existente en la base de casos, pues realiza una comparación exhaustiva con todos los casos almacenados en el sistema. Además, facilita la incorporación directa de nuevos casos en la base de casos sin requerir ningún tipo de procesamiento de la misma. Entre los inconvenientes cabe destacar que este esquema no es apropiado cuando el número de casos en el sistema es elevado, pues al realizar una comparación exhaustiva con todos los casos de la base de casos el tiempo de respuesta del sistema puede crecer considerablemente.

- **Estructura jerárquica:** la base de casos se organiza jerárquicamente, agrupándose distintos casos según sus posibles semejanzas. Constituye una alternativa bastante compleja de implementar, manifestando igualmente algunas ventajas y algunos inconvenientes. Entre las ventajas cabe destacar que el proceso de búsqueda se realiza más eficientemente al no realizar una comparación exhaustiva con todos los casos de la base de casos, por lo que este esquema es el más adecuado cuando el número de casos del sistema es eleva-

do. Entre los inconvenientes cabe destacar que es necesario mantener una organización jerárquica óptima de la base de casos, lo que supone una sobrecarga en el sistema cuando es necesario incorporar nuevos casos y reestructurar la base de casos existente. Además, el proceso de búsqueda puede no devolver siempre el caso más parecido, circunstancia que ocurre cuando la búsqueda deriva en alguna rama equivocada de la estructura jerárquica. Algunas estructuras jerárquicas comúnmente utilizadas son [AP94]:

- **Modelo de memoria dinámica:** el modelo de memoria dinámica [Sch82, Kol83, Kot89] utiliza unas entidades conocidas como *episodios* organizadas jerárquicamente. Un episodio es una estructura abstracta que agrupa un número de casos que comparten propiedades similares, aunque también puede agrupar otros episodios más específicos, definiéndose pues una estructura jerárquica de episodios. El objetivo de los episodios es simplificar el almacenamiento, estructuración y extracción de los distintos casos de la base de casos mediante una jerarquización de los mismos. La búsqueda se realiza según una estrategia descendente (*top-down*) a través de los distintos episodios, y la incorporación de nuevos casos puede originar la aparición de nuevos episodios, asemejándose a un modelo de memoria dinámica que varía según el conocimiento que contiene.
- **Modelo de categorías y ejemplares:** el modelo de categorías y ejemplares [Bar89, PBH90] utiliza unas entidades conocidas como *categorías*, las cuales se corresponden con una agrupación de casos, también conocidos como *ejemplares*. Cada ejemplar pertenece a una categoría, existiendo una red de categorías relacionadas entre sí semánticamente. La búsqueda de casos se realiza examinando aquellas categorías que comparten varias de las características del caso, y la incorporación de nuevos casos se realiza asignándolo a la categoría que contiene una mayor semejanza con sus características.

Las estructuras planas son sencillas de implementar y siempre devuelven el caso más parecido existente en la base de casos, aunque requieren mantener acotado el número de casos para evitar un deterioro significativo de las prestaciones del sistema al aumentar el tiempo de búsqueda. Las estructuras jerárquicas pueden resultar más eficientes en sistemas con un elevado número de casos, si bien su implementación es más compleja y requiere un mantenimiento en condiciones óptimas de la base de casos mediante una adecuada reestructuración de la misma. Además, las estructuras jerárquicas pueden perder casos potencialmente significativos en el proceso de búsqueda si se opta por una zona equivocada de la estructura jerárquica.

En la presente Tesis se ha optado por una estructura plana por sencillez, así como para evitar posibles problemas por pérdida de casos en el proceso de búsqueda. Para que este esquema sea viable se ha implementado un mecanismo de optimización de la base de casos que permite mantener acotado el número de casos almacenados en el sistema, tal y como se describe en el apartado 2.5.

Para cuantificar la degradación de las prestaciones temporales del sistema *CBR* en función del tamaño de la base de casos utilizada se ha llevado a cabo una sencilla prueba, en la cual se

Pentium 4 - 2.4 GHz - 512 MB RAM				
	250 casos	500 casos	750 casos	1000 casos
Total	$3.39 \pm 0.04\ ms$	$7.04 \pm 0.10\ ms$	$14.9 \pm 0.6\ ms$	$20.1 \pm 0.7\ ms$
Pentium 3 - 1.0 GHz - 256 MB RAM				
	250 casos	500 casos	750 casos	1000 casos
Total	$11.5 \pm 0.4\ ms$	$20.1 \pm 0.5\ ms$	$54 \pm 3\ ms$	$73 \pm 4\ ms$
Pentium 3 - 500 MHz - 192 MB RAM				
	250 casos	500 casos	750 casos	1000 casos
Total	$12.9 \pm 0.5\ ms$	$25.6 \pm 1.1\ ms$	$67 \pm 4\ ms$	$93 \pm 5\ ms$

Tabla 5.1: Tiempo medio de respuesta del sistema *CBR* propuesto en función del número de casos almacenados en la base de casos.

ha medido el tiempo medio de respuesta del sistema *CBR* utilizado mientras se aumentaba paulatinamente el número de casos almacenados en el mismo. Para ello se ha utilizado una configuración con 8 sensores, y puesto que las prestaciones finales exhibidas dependen claramente de los recursos de la máquina empleada para implementar el sistema *CBR*, se ha repetido la misma prueba para distintos tipos de máquinas: una máquina de altas prestaciones (Pentium 4 - 2.4 GHz - 512 MB RAM), una máquina de prestaciones medias (Pentium 3 - 1.0 GHz - 256 MB RAM) y una máquina de bajas prestaciones (Pentium 3 - 500 MHz - 192 MB RAM). El tiempo medio de respuesta del sistema *CBR* se muestra en la tabla 5.1, donde los errores se han calculado sobre un intervalo de confianza del 95%.

Según la tabla 5.1 el tiempo de respuesta del sistema *CBR* empeora considerablemente cuando aumenta el número de casos, como era de esperar, pero aun así se mantiene dentro de unos valores muy aceptables. Incluso en condiciones extremas con una base de casos compuesta por 1000 casos y utilizando una máquina de bajas prestaciones el retardo medio del sistema está alrededor de los 90 ms, lo que implica que a una velocidad del agente de unos 300 mm/s se recorren unos 3 cm antes de obtener la respuesta adecuada ante un nuevo caso. En el sistema desarrollado se mantienen unos 250 casos gracias al mecanismo de optimización implementado, por lo que incluso utilizando una máquina de bajas prestaciones el tiempo de respuesta del sistema *CBR* está alrededor de los 13 ms, es decir, un desplazamiento del agente de unos 4 mm a una velocidad de 300 mm/s.

2.4 El ciclo *CBR*

El modo de operación de todo sistema *CBR* se concreta en lo que se conoce como **ciclo *CBR***, que Agnar Aamodt y Enric Plaza resumen en cuatro fases [1] [AP94]:

- **Recuperar:** rescatar de la base de casos el caso más parecido a la situación actual.
- **Reusar:** modificar la solución propuesta por el caso más parecido, generando una nueva

[1] Conocidas como las cuatro "REs": *REtrieve*, *REuse*, *REvise* y *REtain*.

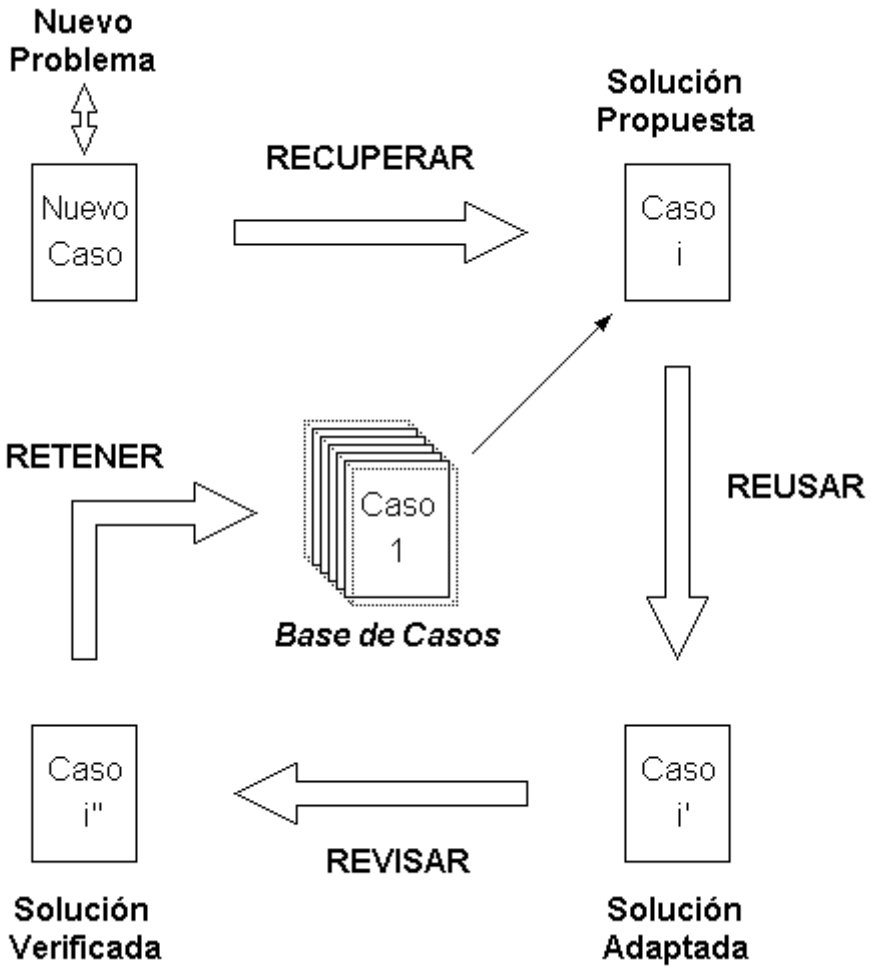

Figura 5.2: Fases que componen el ciclo *CBR*.

solución adaptada al nuevo problema a resolver.

- **Revisar:** verificar la viabilidad de la solución adaptada para resolver el nuevo problema, y modificarla en caso de ser necesario.

- **Retener:** almacenar la nueva solución verificada como un nuevo caso en la base de casos.

La figura 5.2 representa estas cuatro fases. Cuando se presenta un nuevo problema al sistema descrito mediante un nuevo caso, se recupera de la base de casos el caso más parecido que se posee actualmente, el cual describe la experiencia más parecida que tiene el sistema en la resolución de un problema equivalente al planteado. La solución propuesta por el caso más parecido se convierte en candidata para ser utilizada en la resolución del problema actual. No obstante, salvo que el caso recuperado sea muy similar al caso presentado, la solución propuesta puede tener que ser modificada para adecuarla al nuevo problema planteado, en cuyo caso se obtiene una nueva solución adaptada. Finalmente, tras evaluar la idoneidad de la nueva solución adaptada para resolver el nuevo problema se puede almacenar la nueva solución verificada en el

sistema, formando una nueva experiencia que puede ser utilizada en la resolución de problemas futuros.

Una descripción más detallada de las operaciones realizadas en cada una de estas fases es la que se realiza en los siguientes apartados.

2.4.1 Recuperar

En esta fase se debe proceder a la búsqueda en la base de casos del sistema del caso más parecido al que describe la situación actual que se quiere resolver, es decir, se debe rescatar del conocimiento almacenado en el sistema la experiencia más parecida al problema actual que se está planteando. Este proceso de búsqueda es más complejo de realizar que el llevado a cabo en sistemas compuestos por bases de datos, donde se realiza una búsqueda exacta de algún tipo de dato y el resultado final no tiene por qué devolver ninguna corcondancia. En un sistema *CBR* siempre se devuelve un caso, que será el más parecido al caso de entrada, por lo que es necesario establecer una medida de similitud entre casos para poder evaluar el parecido entre unos y otros.

Algunas de las técnicas más utilizadas para realizar el proceso de búsqueda en sistemas *CBR* son [Wat95]:

- **Vecindad:** es la técnica de búsqueda más habitual, y consiste en establecer una correspondencia ponderada entre las características de los casos a comparar [Kol93]. Para ello se define una función de distancia que determine la similitud relativa entre casos, combinando adecuadamente cada una de las características del caso al evaluar dicha distancia. La problemática asociada a esta técnica radica en la adecuada selección de los pesos para cada una de las características de los casos.
- **Inducción:** este tipo de técnicas se basan en seleccionar aquellas características más determinantes a la hora de discriminar entre casos, y construir de esa forma una estructura jerárquica para acelerar el proceso de búsqueda [Qui79]. Este tipo de algoritmos son útiles en aquellos sistemas donde existe una característica determinante que diferencia los casos, de la cual dependen el resto de características.
- **Prototipos:** esta técnica pretende realizar una primera búsqueda en la base de casos para identificar un conjunto menor de casos que puedan corresponderse con el caso de entrada. Suele emplearse de manera combinada con las otras técnicas para acotar el conjunto inicial de casos y acelerar el proceso de búsqueda.

En la presente Tesis se va a utilizar la técnica de vecindad, puesto que la técnica de inducción no es apropiada al no ser posible identificar ninguna característica determinante en la representación del caso que se ha diseñado en el apartado 2.2. No se va a utilizar la técnica de prototipos para intentar reducir el conjunto inicial de casos sobre los que realizar la búsqueda, pues como se ha comentado en el apartado 2.3 se va a mantener acotado el número de casos del sistema al utilizar una estructura plana en la base de casos.

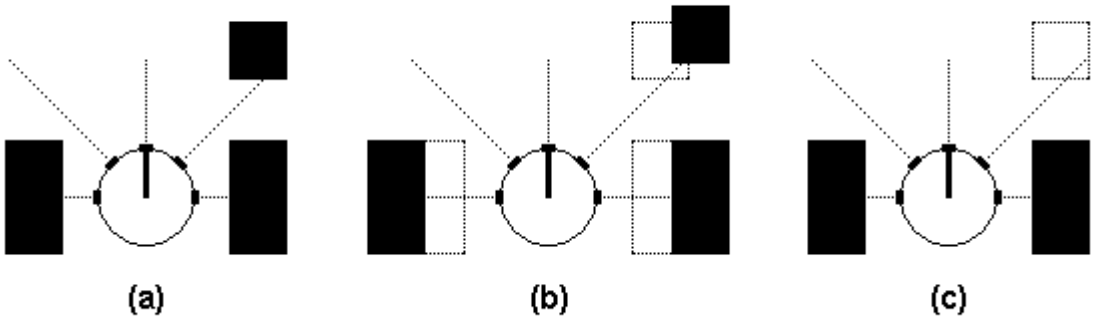

Figura 5.3: Selección de la función distancia: a) caso actual; b) caso almacenado 1; c) caso almacenado 2.

Para concretar por completo el proceso de búsqueda basado en la técnica de vecindad es necesario escoger los pesos asociados a cada una de las características del caso y la función distancia que calcule el grado de similitud entre casos. A continuación se describe el diseño de estos aspectos en el proceso de búsqueda:

- **Pesos:** los pesos de la técnica de vecindad pretenden poner de manifiesto la importancia relativa entre las distintas características que posee un caso, que en el sistema diseñado son la dirección hacia el destino y las lecturas de los sensores del agente. Ninguna de estas características es más importante que la otra a la hora de seleccionar casos similares, puesto que suele ser tan importante recuperar experiencias pasadas donde el agente se dirigía en un dirección parecida a la actual como aquellas donde el entorno circundante mantenía un parecido evidente con el actual. Por lo tanto, el peso asignado a las distintas características del caso ha sido el mismo para todas ellas.

- **Distancia:** se ha constatado que la distancia Euclídea, determinada por la ecuación:

$$d_E(X,Y) = \sqrt{(X-Y)\cdot(X-Y)^T} \tag{5.4}$$

  no resulta la más adecuada para medir el parecido entre los casos X e Y, como muestra la figura 5.3. Supóngase que se dispone de la situación actual representada en la figura 5.3.a, y que se tienen almacenados en el sistema los casos representados por las figuras 5.3.b y 5.3.c. Aunque parece evidente que el caso actual de la figura 5.3.a es más parecido al caso de la figura 5.3.b, donde todos los obstáculos se han alejado radialmente respecto al agente, la distancia Euclídea devuelve una distancia menor entre el caso actual de la figura 5.3.a y el caso de la figura 5.3.c, donde los obstáculos laterales están exactamente a la misma distancia del agente en ambos casos. La distancia Euclídea favorece la igualdad entre características, pues a pesar de la enorme variación que puede experimentar una característica en concreto, la exactitud con la que pueden coincidir el resto puede compensar la distancia global de forma que el sistema relacione casos claramente diferentes desde un

punto de vista conceptual. Así pues, en el sistema desarrollado se ha optado por utilizar la distancia de Tanimoto [ST79], la cual favorece la similitud o correlación entre casos y no únicamente la distancia entre los mismos [WBD98]. La distancia de Tanimoto empleada para medir el grado de similitud entre los casos X e Y tiene la expresión:

$$d_T(X,Y) = 1 - \frac{X \cdot Y^T}{|X|^2 + |Y|^2 - X \cdot Y^T} \tag{5.5}$$

2.4.2 Reusar

En esta fase la solución propuesta por el caso más parecido recuperado de la base de casos debe ser reutilizada para resolver el problema actual planteado. Salvo que el caso devuelto sea muy similar al caso actual que describe el nuevo problema, suele ser necesario adaptar la solución propuesta por el sistema para que sea aplicable al problema a resolver. Si bien existen numerosas técnicas para llevar a cabo esta fase, las diferentes aproximaciones se pueden agrupar en tres grandes categorías [Kol93]:

- **Adaptación nula:** no realiza ningún tipo de adaptación de la solución propuesta, por lo que propone directamente la solución aportada por el caso devuelto, siendo útil en sistemas sencillos que requieren soluciones muy simples [Alt88]
- **Adaptación estructural:** realiza la adaptación directamente sobre la solución propuesta por el caso devuelto, ajustando algunas de sus características [Bai86, Syc87].
- **Adaptación derivada:** recalcula una nueva solución para el caso actual, utilizando para ello un modelo analítico que permita obtener una solución a partir del nuevo problema planteado [Car86, KS89].

Esta técnicas no son excluyentes entre sí, sino que se pueden combinar. En el sistema desarrollado en la presente Tesis se ha escogido por sencillez la adaptación estructural, modificación el caso devuelto alejar al agente de los obstáculos circundantes mientras se dirige al destino final. Para ello, cuando la distancia entre el caso devuelto y el caso actual es superior a un umbral de adaptación U_{adap}, umbral que se puede fijar experimentalmente según la tolerancia que se desee en el sistema, se considera que el sistema se enfrenta a una situación desconocida y es necesario adaptar la solución propuesta. Esta adaptación se realiza según el procedimiento que se describe a continuación:

1. **Repulsión:** mediante el uso de los Campos Potenciales descritos en el apartado 1.1.1 se calcula la dirección de avance que aleja al agente de los obstáculos del entorno. Para ello, cada obstáculo dentro del rango de detección del agente genera una fuerza repulsiva tanto más potente cuanto más cercano se encuentre al agente.
2. **Corrección:** la solución generada por los Campos Potenciales es combinada con la solución propuesto por el caso devuelto, de forma que se tenga en cuenta la presencia de los

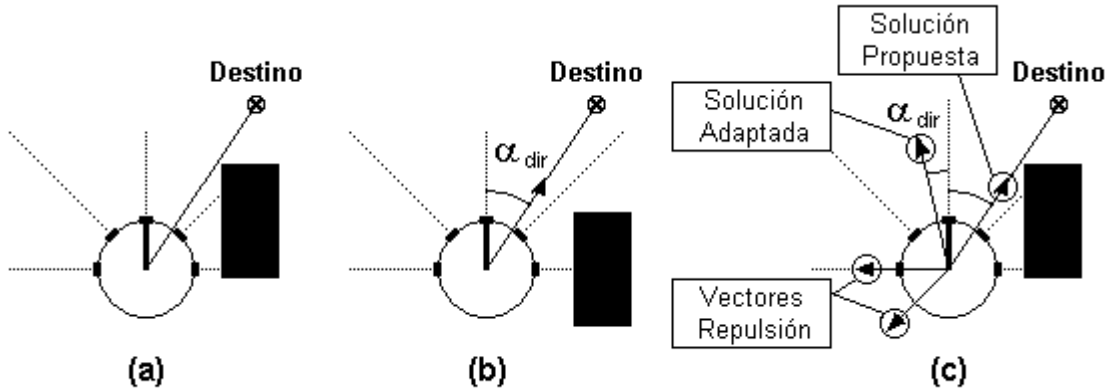

Figura 5.4: Adaptación del caso devuelto.

nuevos obstáculos del entorno. Cuanto más cercanos se encuentren los obstáculos, mayor será la fuerza repulsiva generada por los Campos Potenciales, por lo que la solución propuesta será modificada en mayor medida.

Es importante notar que la nueva solución adaptada no coincide con la solución que aportaría el esquema analítico de los Campos Potenciales, pues en ningún momento se emplea la fuerza atractiva hacia el objetivo que utiliza dicho esquema. La fuerza repulsiva calculada simplemente modifica la solución propuesta si se considera que el problema planteado presenta una situación desconocida para el sistema, alejando al agente de los obstáculos del entorno en función de la distancia a los mismos.

La figura 5.4 muestra un ejemplo de adaptación mediante la estrategia anteriormente descrita. Mientras el agente avanza hacia el destino final detecta un obstáculo a la derecha que imposibilita el avance directo hacia el mismo, tal y como se aprecia en la figura 5.4.a. El caso más parecido disponible en la base de casos se corresponde con una situación en la que el obstáculo se encontraba igualmente a la derecha pero no directamente en la línea de avance hacia el destino, por lo que permitía una trayectoria de aproximación directa como se aprecia en la figura 5.4.b. Si dicha solución se utilizase sin adaptación, evidentemente el agente colisionaría con el nuevo obstáculo detectado. Combinando esta solución propuesta con la solución planteada por los Campos Potenciales se genera una nueva solución adaptada al nuevo problema que permite que el agente continúe navegando mientras se separa suavemente del obstáculo detectado, como se aprecia en la figura 5.4.c.

2.4.3 Revisar

En esta fase se debe evaluar la viabilidad de la solución adaptada para resolver el nuevo problema planteado, proceso que no se suele realizar en tiempo real, sino tras aplicar la solución adaptada por el sistema y conocer sus consecuencias. En aquellas circunstancias donde el resultado de la evaluación sea negativo suele ser necesario una nueva modificación de la solución adaptada, con

el fin de corregir las posibles deficiencias detectadas en la misma. Obviamente, esta fase sólo tiene sentido en aquellos casos en los que se ha realizado una adaptación del caso recuperado, pues es necesario verificar que la nueva solución adaptada satisface los requisitos propios de la aplicación que se está desarrollando.

La evaluación de la nueva solución adaptada se puede llevar a cabo siguiendo distintas estrategias:

- Manualmente mediante la intervención de algún operador humano que valide la solución adaptada.
- Automáticamente mediante algún algoritmo que simule o evalúe los resultados de la solución adaptada.
- Mediante una simple observación de los resultados obtenidos tras aplicar la solución adaptada.

Con el objetivo de evaluar la viabilidad de la solución descrita se ha implementado un mecanismo automático que mide su eficiencia. Para poder cuantificar la eficiencia de un caso es necesario en primer lugar determinar cualitativamente los factores que influyen en dicha eficiencia. El objetivo de todo esquema reactivo de navegación es alcanzar el destino final garantizando la seguridad del agente al evitar la colisión con los obstáculos del entorno. No obstante, al tratarse de un sistema reactivo únicamente se posee información instantánea del entorno inmediato, por lo que la eficiencia de un caso no se puede evaluar con factores globales como distancias totales recorridas o suavidad de la trayectoria seguida, sino que hay que identificar aquellos factores puntuales que inciden positiva o negativamente en la evaluación instantánea de cada caso.

Tal y como se ha descrito en el apartado 2.2 cada caso en el sistema posee la siguiente información:

- α_{dest}: dirección en la que se encuentra el destino final respecto a la dirección de avance del agente.
- S_i: lectura del sensor i.
- α_{dir}: solución propuesta como dirección de avance del agente.

Por lo tanto, ante un nuevo problema caracterizado por la dirección en la que se encuentra el destino final y por la configuración de obstáculos del entorno que detecta el agente por medio de los sensores, se propone la dirección de avance que debe seguir el agente. En la medida de la eficiencia de esta solución propuesta por un determinado caso se han distinguido tres factores distintos:

- **Suavidad:** un caso resulta tanto más eficiente cuanto menores son los cambios de dirección que propone. El reducir los cambios de dirección reduce los cambios bruscos que

realiza el agente y conduce a trayectorias más suaves, lo cual es deseable para reducir otros efectos indeseados de la navegación como el conocido deslizamiento (*slippage*), que introduce errores en el sistema odométrico del agente e induce a errores en la localización del mismo. Este factor de suavidad se puede medir heurísticamente mediante la siguiente expresión:

$$F_{soft} = e^{-C_{soft} \cdot |\alpha_{dir}|} \tag{5.6}$$

donde C_{soft} es un parámetro genérico que permite ajustar la expresión y α_{dir} es la solución propuesta como dirección de avance del agente. Esta expresión varía entre 0 y 1, y es máxima cuando la dirección de avance es cero, es decir, cuando el agente avanza en línea recta sin ningún cambio de dirección, tal y como se pretende medir.

- **Distancia:** un caso resulta tanto más eficiente cuanto menos se separa de la dirección a la que se encuentra el destino final. Este factor intenta calibrar la posible longitud de la trayectoria final recorrida, la cual tenderá a reducirse si se aproxima a la trayectoria en línea recta hacia el destino final. Es importante recordar que en un esquema reactivo no es posible evaluar instantáneamente la trayectoria final, pues no se conoce al no existir ningún tipo de planificación. No obstante, ante distintas alternativas es conveniente favorecer aquellos casos que no se desvían excesivamente del destino final. Este factor de distancia se puede medir heurísticamente mediante la siguiente expresión:

$$F_{dist} = e^{-C_{dist} \cdot |\alpha_{dest} - \alpha_{dir}|} \tag{5.7}$$

donde C_{dist} es un parámetro genérico que permite ajustar la expresión, α_{dest} es la dirección en la que se encuentra el destino final respecto a la dirección de avance del agente y α_{dir} es la solución propuesta como dirección de avance del agente. Esta expresión varía entre 0 y 1, y es máxima cuando la dirección de avance coincide con la dirección a la que se encuentra el destino final, tal y como se pretende medir.

- **Seguridad:** un caso resulta tanto más eficiente cuanto menos se aproxima el agente a los obstáculos del entorno. Este factor intenta penalizar aquellos casos que se dirigen hacia los obstáculos, con una penalización mayor cuanto más próximo se encuentren los obstáculos, pues mayor es el riesgo de colisión en dichas circunstancias. Este factor de seguridad se puede medir heurísticamente mediante la siguiente expresión:

$$F_{sec} = 1 - e^{-C_{sec} \cdot d_{min} \cdot |\alpha_{min} - \alpha_{dir}|} \tag{5.8}$$

donde C_{sec} es un parámetro genérico que permite ajustar la expresión, α_{min} es la dirección en la que se ha detectado el obstáculo más cercano, d_{min} es la distancia al obstáculo más cercano y α_{dir} es la solución propuesta como dirección de avance del agente. Esta expresión varía entre 0 y 1, y es mínima cuando la dirección de avance coincide con la dirección a la que se encuentra el obstáculo más cercano, reduciéndose además cuanto más próximo se encuentra dicho obstáculo, tal y como se pretende medir.

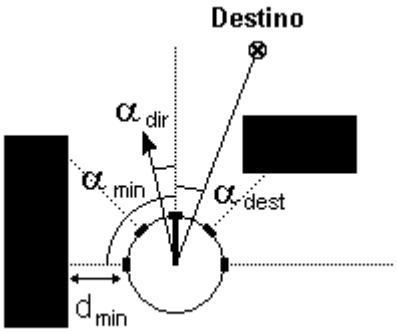

Figura 5.5: Parámetros involucrados en la eficiencia de un caso.

La figura 5.5 representa los distintos parámetros implicados en estos tres factores, los cuales se pueden extraer instantáneamente a partir de las características que posee un caso, no necesitándose ningún tipo de información global tal y como corresponde a un esquema reactivo. La eficiencia final de un caso, pues, se puede determinar mediante la combinación de estos tres factores calculados independientemente. Para añadir un mayor grado de flexibilidad se ha realizado una combinación ponderada de estos tres factores, definiéndose la eficiencia de un caso mediante la siguiente expresión:

$$\eta = \frac{K_{soft} \cdot F_{soft} + K_{dist} \cdot F_{dist} + K_{sec} \cdot F_{sec}}{K_{soft} + K_{dist} + K_{sec}} \tag{5.9}$$

donde K_{soft}, K_{dist} y K_{sec} representan unos coeficientes de ponderación que definen la importancia relativa que se le pretende otorgar a cada uno de estos factores respecto a los demás. Como se puede deducir de esta expresión la eficiencia de un caso varía entre 0 y 1. Gracias a la combinación ponderada de estos tres factores se puede adaptar y medir la eficiencia del sistema siguiendo distintos criterios, según se le otorgue mayor importancia a la suavidad, a la distancia o a la seguridad de la trayectoria final recorrida.

Una vez obtenida la eficiencia del nuevo caso se debe proceder a su evaluación para determinar si la solución adaptada es adecuada o no para el nuevo problema planteado. Una posible solución consiste en evaluar positivamente sólo aquellos casos cuya eficiencia se encuentre por encima de un umbral determinado. Sin embargo, este sencillo esquema no resulta adecuado en el sistema desarrollado, pues en algunas circunstancias el agente puede verse obligado a adoptar soluciones poco eficientes para resolver determinados problemas, como por ejemplo al atravesar un pasillo estrecho en el que se tiene que aproximar irremediablemente a los obstáculos. Si simplemente se eliminan del sistema aquellos casos con menor eficiencia se le obligaría al sistema a reaprender una y otra vez estos casos problemáticos con baja eficiencia.

En el sistema propuesto se ha implementado un esquema algo más complejo pero más apropiado para la evaluación de los distintos casos. En este esquema todos los nuevos casos adaptados son evaluados positivamente, aunque cada vez que se introduce un nuevo caso en la base de casos

del sistema es etiquetado con la eficiencia obtenida por la expresión 5.9. Esta eficiencia con la que se etiqueta cada caso es utilizada posteriormente por el mecanismo de optimización de la base de casos descrito en el apartado 2.5, que además de mantener acotado el número de casos almacenados en el sistema favorece que los casos más eficientes tengan mayor influencia en la experiencia almacenada en el sistema.

2.4.4 Retener

En esta fase se incorporan al sistema todas aquellas soluciones verificadas que se han obtenido tras la fase de adaptación y que han sido evaluadas satisfactoriamente, por lo que se realiza el aprendizaje de las nuevas experiencias que pueden ser útiles en la resolución de futuros problemas. Esta fase es una de las más importantes en un sistema *CBR*, pues permite introducir un mecanismo de realimentación que posibilita el aprendizaje, existiendo principalmente dos tipos de técnicas distintas:

- **Aprendizaje por observación:** implementado cuando se introduce en la base de casos un conjunto inicial de casos válidos, aportados por un operador humano o por observación directa de casos reales.
- **Aprendizaje por experiencia:** implementado implícitamente en el sistema cuando una nueva solución es evaluada satisfactoriamente y se incorpora en la base de casos.

Ambas técnicas se suelen emplear de forma combinada, pues representan mecanismos de aprendizaje complementarios. En el sistema propuesto se ha utilizado tanto el aprendizaje por observación como el aprendizaje por experiencia, por lo que se va a proceder a continuación a describir con mayor detalle la implementación de ambas técnicas.

Aprendizaje por observación

El aprendizaje por observación [vLL98] forma parte del desarrollo inicial de cualquier sistema que implementa el paradigma *CBR*, y se asemeja a la etapa de entrenamiento inicial de otros sistemas basados en aprendizaje, como pueden ser las redes neuronales. Consiste en introducir en la base de casos un conjunto inicial de casos que formarán la experiencia básica con la que el sistema comenzará a operar. En el sistema propuesto este conjunto inicial de casos se ha obtenido tras capturar distintas situaciones mientras el agente recorre diferentes trayectorias. Así pues, mientras el agente navega se van analizando de forma continua la dirección en la que se encuentra el destino final y la posición de los obstáculos del entorno mediante la lectura de los sensores, es decir, se van obteniendo continuamente los casos correspondientes a dicha trayectoria. Cuando la distancia entre el último caso almacenado y el actual supera un cierto umbral de disparidad U_{disp} se captura un nuevo caso, incluyendo en dicho caso la dirección que sigue en ese instante el agente como solución al problema. El umbral de disparidad U_{disp} se encarga de que no se almacenen casos demasiado parecidos en la base de casos, creciendo innecesariamente el número de casos en el sistema. Siempre que un nuevo caso es incorporado

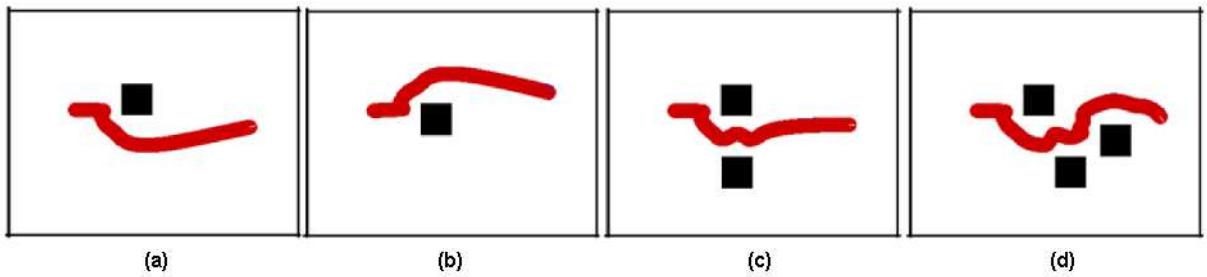

Figura 5.6: Aprendizaje por observación a partir del esquema de los Campos Potenciales: a) un obstáculo a la izquierda; b) un obstáculo a la derecha; c) dos obstáculos; d) tres obstáculos.

al sistema se etiqueta con la eficiencia obtenida por la expresión 5.9, tal y como se describe en el apartado 2.4.3. Esta eficiencia será posteriormente utilizada por el esquema de optimización de la base de casos descrito en el apartado 2.5.

Para que este esquema de aprendizaje por observación se realice correctamente es necesario que el agente recorra una serie de trayectorias iniciales. Estas trayectorias pueden ser generadas por cualquier sistema de navegación que se desee imitar, o incluso directamente por un operador humano. En el sistema propuesto se ha posibilitado la generación de las trayectorias de entrenamiento a partir del esquema analítico de los Campos Potenciales y a partir de un operador humano, con el fin de dotar de mayor flexibilidad al agente.

Con el fin de evaluar la influencia del esquema de aprendizaje por observación propuesto en la operación del sistema se ha sometido al mismo a distintas pruebas, las cuales se describen a continuación con mayor detalle.

Una primera prueba se ha centrado en el análisis de la capacidad de navegación alcanzada por el agente partiendo de un entrenamiento basado en el esquema de los Campos Potenciales. Para ello se han generado distintas trayectorias en diferentes entornos y se han capturado los casos más significativos de cada una. Cabe señalar que se puede utilizar cualquier otro esquema de navegación, pero se ha empleado el esquema de los Campos Potenciales por su sencillez y para verificar si el sistema *CBR* es capaz de mejorar los inconvenientes manifestados por dicho esquema aun utilizándolo en su fase de aprendizaje. Tras ajustar los parámetros necesarios de los Campos Potenciales para evitar colisiones y garantizar que el sistema funcione correctamente, se han realizado las trayectorias que se presentan en las figuras 5.6.a a 5.6.d. De todas estas trayectorias únicamente la trayectoria de la figura 5.6.a se ha utilizado para adquirir los casos significativos necesarios para materializar el aprendizaje por observación. El resto de trayectorias se han realizado para posibilitar la comparación entre el esquema de los Campos Potenciales y el sistema *CBR* propuesto, no habilitándose la adquisición de ningún caso durante las mismas. Puede observarse en las trayectorias de la figura 5.6 las oscilaciones típicas de los Campos Potenciales.

A continuación, y usando únicamente los casos aprendidos durante la trayectoria de la figura 5.6.a, se ha procedido a repetir estas pruebas con el sistema *CBR*, mostrándose los resultados en las figuras 5.7.a a 5.7.d. Los mapas métricos utilizados en estas figuras se emplean únicamente

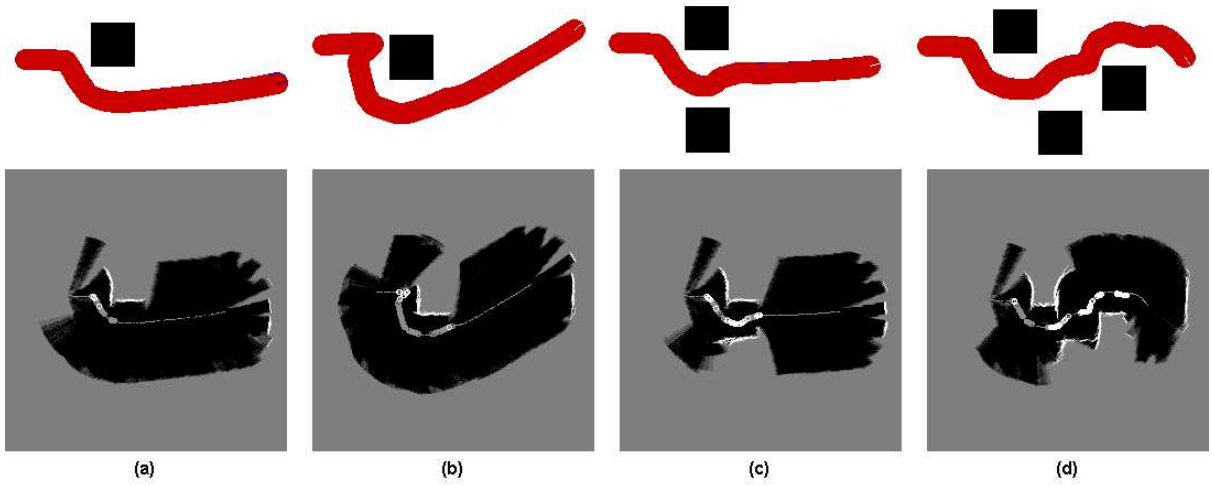

Figura 5.7: Operación del sistema *CBR* con aprendizaje por observación a partir del esquema de los Campos Potenciales: a) un obstáculo a la izquierda; b) un obstáculo a la derecha; c) dos obstáculos; d) tres obstáculos.

para la visualización de los resultados, ya que la capa reactiva no hace uso de ellos en momento alguno. Sobre estos mapas se han superpuesto las trayectorias seguidas por el agente, marcándose con círculos grises aquellas circunstancias donde ha sido utilizado el sistema *CBR* [2] y con círculos blancos aquellas circunstancias donde los casos devueltos tuvieron que ser adaptados. Es importante remarcar que las nuevas soluciones obtenidas en estos casos adaptados no fueron incorporadas al sistema, pues el objetivo de estas pruebas era verificar el funcionamiento únicamente del esquema de aprendizaje por observación.

Como puede observarse en los resultados de la figura 5.7, a pesar del escaso entrenamiento realizado en el sistema las trayectorias resultantes presentan menores oscilaciones que las correspondientes al esquema de los Campos Potenciales de la figura 5.6. La figura 5.7.b presenta un comportamiento interesante, pues aunque en esta situación particular es más eficiente alcanzar el destino final dejando el obstáculo a la derecha como ocurre en la figura 5.6.b, el sistema deja el obstáculo a la izquierda al ser el único comportamiento que ha aprendido de la figura 5.6.a.

Una vez analizado el aprendizaje por observación a partir del esquema de los Campos Potenciales se ha realizado otra prueba para medir la capacidad de operación del sistema tras realizar el aprendizaje por observación a partir de un operador humano. Para implementar dicho esquema se ha desarrollado un sencillo sistema de telecontrol mediante teclado que permite controlar directamente el movimiento del agente. El operador humano se limita pues a conducir el agente en un campo de obstáculos hasta el destino final, mientras el agente almacena automáticamente los casos más significativos capturados durante el recorrido. Esta estrategia basada en el aprendizaje por observación a partir de un operador humano plantea interesantes ventajas:

[2] Recuérdese que tal y como se ha descrito en el apartado 4.3 del capítulo 3 el sistema de navegación intenta alcanzar el destino final en línea recta, y sólo cuando los obstáculos del entorno se encuentran demasiado cercanos al agente se activa el esquema de evitación de obstáculos implementado.

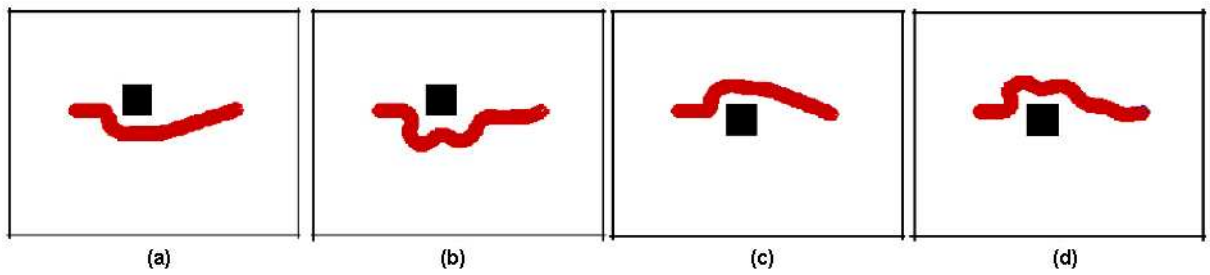

Figura 5.8: Aprendizaje por observación a partir de un operador humano: a-b) un obstáculo a la izquierda; c-d) un obstáculo a la derecha.

- Permite aprender comportamientos de navegación complejos de forma intuitiva y sin necesidad de racionalizarlos.
- Posibilita la optimización del esquema de navegación en circunstancias concretas mediante la acción directa de un operador humano.
- Puesto que el operador humano maneja restricciones cinemáticas y dinámicas al mover el agente se incorporan implícitamente dichos modelos en los casos aprendidos por el sistema.

Para verificar este nuevo esquema de aprendizaje por observación se ha utilizado un operador humano para guiar al agente a través de las trayectorias de la figura 5.8. Puesto que el operador humano puede no ser demasiado preciso ni escoger la trayectoria más óptima se han introducido intencionadamente factores indeseables en el entrenamiento en forma de oscilaciones. De esta forma se pretende verificar la sensibilidad del sistema ante un entrenamiento no muy preciso, pues si el sistema no es capaz de recuperarse de los errores en que puede incurrir el operador humano un entrenamiento de este tipo puede resultar poco apropiado. Durante la realización de este entrenamiento únicamente se ha habilitado el sistema de aprendizaje por observación en las trayectorias de las figuras 5.8.a y 5.8.b.

La figura 5.9 muestra el resultado obtenido por el sistema de navegación basado en *CBR* sobre distintos entornos. Como puede observarse, las trayectorias resultantes presentan menores oscilaciones que las trayectorias con el esquema de los Campos Potenciales de la figura 5.7, incluso en entornos no conocidos como los de las figuras 5.9.c y 5.9.d. Curiosamente en estos ejemplos se ha dejado el obstáculo de la figura 5.9.b a la derecha, debido a que los casos aprendidos son distintos y a que el operador humano tiende a acercarse más a los obstáculos mientras navega, por lo que aquellos casos que deben ser adaptados presentan una mayor fuerza de repulsión de los obstáculos. Así pues, al tener más peso en la composición de fuerzas de los casos adaptados la repulsión de los obstáculos, el agente ha optado por esquivarlo dejándolo a la derecha, puesto que el obstáculo se encuentra mayoritariamente en esa posición. Es también significativo el hecho de que, a pesar de aprender de trayectorias con oscilaciones, los recorridos resultantes tienden a eliminarlas como es deseable. Este hecho se debe a que no todos los casos originales del proceso de entrenamiento son aprendidos, sino únicamente los más significativos en función del umbral de disparidad U_{disp} del proceso de entrenamiento. Así pues, se realiza un filtrado

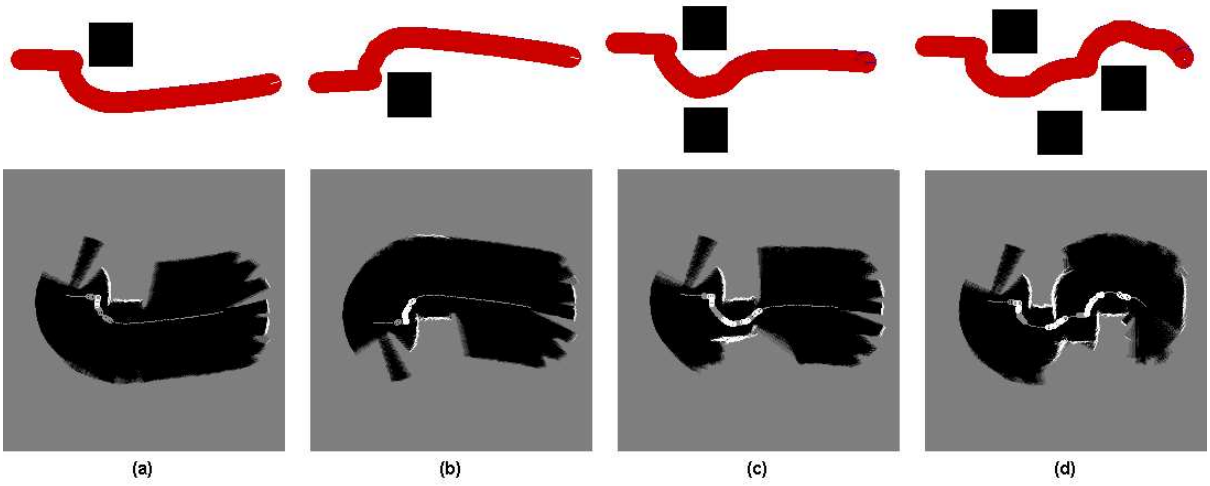

Figura 5.9: Operación del sistema *CBR* con aprendizaje por observación a partir de un operador humano: a) un obstáculo a la izquierda; b) un obstáculo a la derecha; c) dos obstáculos; d) tres obstáculos.

implícito que elimina numerosos casos y por extensión las posibles oscilaciones que presentan dichos casos.

Como ponen de manifiesto las pruebas de la figura 5.9 respecto a las pruebas de la figura 5.7 el comportamiento del sistema es diferente ante distintas situaciones de entrenamiento, como cabe esperar. De hecho, la trayectoria seguida en las pruebas de las figuras 5.7.b y 5.9.b son bastante diferentes, dejando una de ellas el obstáculo a la izquierda y la otra el obstáculo a la derecha. Este modo de operación es totalmente lógico, pues el sistema tiende a emular los comportamientos aprendidos, y si estos son distintos obviamente el comportamiento final exhibido también lo será. No obstante, es necesario matizar este hecho con algunas consideraciones. En primer lugar, las pruebas realizadas han demostrado un comportamiento tan sensible porque el entrenamiento realizado ha sido muy escaso. Con un mayor entrenamiento el sistema puede detectar un mayor número de circunstancias y no verse obligado a adaptar tantos casos, por lo que el comportamiento final es más predecible y menos sensible a nuevas circunstancias desconocidas. En segundo lugar, el mecanismo de optimización de la base de casos descrito en el apartado 2.5 que en estas pruebas no ha sido utilizado tiende a eliminar experiencias poco eficientes en el sistema, por lo que reduce la posibilidad de reproducir casos que impliquen un acercamiento excesivo a los obstáculos o que describan oscilaciones indeseadas. Por último, si bien es cierto que el comportamiento del sistema ha sido muy diferente en estas pruebas, es importante destacar que sea cual sea la trayectoria descrita el sistema ha esquivado adecuadamente los obstáculos del entorno y ha suavizado las oscilaciones realizadas en el proceso de entrenamiento. Por todo ello hay que concluir que el funcionamiento del sistema ha sido correcto independientemente del entrenamiento realizado, existiendo mecanismos suficientes para absorber pequeñas ineficiencias y obtener un esquema de navegación robusto.

Por último, se ha realizado otra prueba para verificar el comportamiento del sistema ante un mayor nivel de entrenamiento. Para destacar la flexibilidad del esquema de aprendizaje por

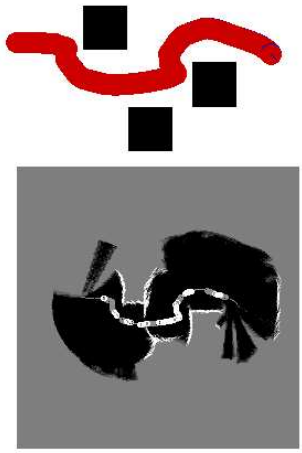

Figura 5.10: Operación del sistema *CBR* con aprendizaje por observación mixto a partir de los Campos Potenciales y a partir de un operador humano.

observación propuesto se ha utilizado un entrenamiento mixto combinando las trayectorias generadas por los Campos Potenciales con las trayectorias generadas por un operador humano. Este esquema de aprendizaje combinado es bastante interesante, pues permite utilizar un entrenamiento generalizado mediante esquemas de navegación analíticos como los Campos Potenciales y particularizar algunas situaciones poco eficientes con otro tipo de entrenamiento generado por un operador humano.

La figura 5.10 muestra el resultado obtenido al navegar en un entorno con tres obstáculos tras realizar un aprendizaje por observación de todas las trayectorias de las figuras 5.6 y 5.8 conjuntamente. Tras un análisis de los resultados obtenidos se puede constatar que prácticamente ningún caso necesita ya adaptarse, por lo que en esta figura los círculos blancos se corresponden con casos aprendidos a partir del esquema de los Campos Potenciales y los círculos grises con casos aprendidos a partir del operador humano. De esta prueba se pueden obtener algunas conclusiones interesantes. La primera es que, al aproximarse al primer obstáculo, el agente ha utilizado los casos derivados del entrenamiento del operador humano, ya que este tiende a acercarse más a los obstáculos. A partir del momento en el que se supera el primer obstáculo el sistema opta mayoritariamente por utilizar los casos derivados del entrenamiento de los Campos Potenciales, pues cabe recordar que no existe ninguna trayectoria guiada por un operador humano en la figura 5.8 que contemplase entornos con dos o tres obstáculos, disponiendo el agente en estas situaciones únicamente de las trayectorias de la figura 5.6. No obstante, es importante notar que la trayectoria final recorrida es bastante más suave que las correspondientes a las figuras 5.7.d y 5.9.d donde el entrenamiento del sistema era bastante más escaso.

Aprendizaje por experiencia

El aprendizaje por experiencia [Sch82] es una de la etapas más importantes en todo sistema *CBR*, pues permite ir aumentando gradualmente el conocimiento que posee el sistema al ir incorporando nuevas experiencias en la resolución de nuevos problemas. Consiste en ir intro-

duciendo en la base de casos las nuevas soluciones aportadas por los casos adaptados una vez han sido positivamente evaluadas, de forma que puedan ser utilizadas como referencias futuras para resolver problemas similares a los recién adquiridos.

Existen numerosas estrategias que pretenden implementar el esquema de aprendizaje por experiencia de un sistema *CBR*, aunque se pueden distinguir claramente dos grandes técnicas:

- **Aprendizaje positivo:** se realiza cuando la evaluación de la nueva solución adaptada ha sido positiva, por lo que el nuevo caso puede ser inmediatamente incorporado a la base de casos. Esto implica introducirlo adecuadamente en el sistema, operación que viene determinada por la estructura de la base de casos empleada. Cuando la base de casos posee una estructura plana el proceso de incorporación del nuevo caso no requiere ningún tipo de procesamiento adicional. Sin embargo, cuando la base de casos posee una estructura jerárquica es necesario localizar la ubicación más adecuada para que sea encontrado en el proceso de búsqueda, proceso que suele requerir la reestructuración de la memoria.
- **Aprendizaje negativo:** se realiza cuando la evaluación de la nueva solución adaptada ha sido negativa. Suponiendo que el proceso de adaptación es correcto, la nueva solución adaptada puede ser incorrecta debido básicamente a dos circunstancias. La primera de ellas se produce cuando se ha localizado correctamente el caso más parecido al actual en la base de casos pero ambos casos difieren sustancialmente. Este hecho se produce por un escaso entrenamiento del sistema, y se puede corregir fácilmente reforzando el conocimiento del mismo mediante una etapa de aprendizaje por observación. La segunda circunstancia en la que puede fallar la solución adaptada se produce cuando el caso devuelto no es el más parecido que existe en el sistema, por lo que el proceso de adaptación puede conducir a resultados erróneos. Esta es una característica grave de los sistemas que usan un estructura jerárquica en la base de casos, y puede implicar una reorganización de la estructura de la base de casos [VC93].

  De cualquier forma, ante una evaluación negativa de cualquier solución adaptada por el sistema se presentan distintas alternativas para intentar que dicha solución no se vuelva a realizar:

  - Almacenar el caso fallido en la propia base de casos, precediendo el proceso de búsqueda de una nueva fase conocida como *Anticipación*, la cual realiza un filtrado de aquellos casos no deseados.
  - Almacenar el caso en otra base de casos específicamente dedicada a los casos fallidos [Ham89], por lo que antes de realizar la búsqueda en la base de casos normal se realiza una búsqueda en esta base de casos fallidos para no volver a repetir el error.
  - Corregir el error mediante algún tipo de procesamiento, que puede incluir por supuesto la consulta a un operador humano, y así almacenar la solución correcta en la base de casos normal del sistema.

No todos los sistemas implementan ambos tipos de aprendizaje por experiencia. En el sistema propuesto se ha utilizado únicamente el aprendizaje positivo, que al usar una estructura plana

para la base de casos tal y como se ha descrito en el apartado 2.3 se realiza simplemente mediante la incorporación directa del caso en la base de casos. Los motivos que han conducido a implementar únicamente la técnica de aprendizaje positivo son los que se describen a continuación:

- Puesto que el sistema emplea una estructura plana en la base de casos los posibles fallos cometidos en el proceso de adaptación están originados porque el sistema no posee suficiente conocimiento, situación que se puede corregir aumentando el entrenamiento del sistema mediante una etapa adicional de aprendizaje por observación.

- Según se ha descrito en el apartado 2.4.3 todos los nuevos casos adaptados en el sistema obtienen una evaluación positiva, aunque son etiquetados con la eficiencia obtenida por la expresión 5.9, que será utilizada por el esquema de optimización de la base de casos descrito en el apartado 2.5. Al no existir casos evaluados negativamente no es viable implementar la técnica de aprendizaje negativo.

- El aprendizaje negativo trata de detectar y eliminar posibles casos incorrectos en el sistema para evitar que las soluciones erróneas que proponen se vuelvan a adoptar en el futuro. El mecanismo de optimización de la base de casos descrito en el apartado 2.5 penaliza los casos menos eficientes del sistema, por lo que se descartan aquellas soluciones almacenadas que no conducen a resultados correctos. Como ya se ha comentado, el disponer de las soluciones correctas para esas circunstancias se resuelve con un mayor nivel de entrenamiento.

Para verificar la influencia del aprendizaje por experiencia en la operación del sistema se han llevado a cabo diferentes pruebas en el sistema, las cuales se describen detalladamente a continuación.

Una primera prueba pretende analizar la mejora en la operación del sistema utilizando adecuadamente el aprendizaje por experiencia. Para ello se ha ido estrechando un pasillo paulatinamente hasta que el esquema de navegación de los Campos Potenciales es incapaz de atravesarlo, como se muestra en la figura 5.11.a. Tal y como se ha descrito en el apartado 1.1.1, la imposibilidad de atravesar obstáculos muy cercanos es uno de los inconvenientes típicos de los Campos Potenciales. Una vez configurado el entorno se ha utilizado el sistema de navegación mediante *CBR* con una base de casos vacía, es decir, sin ningún tipo de conocimiento [3]. Así pues, todas las situaciones encontradas por el sistema son desconocidas y el sistema tiene que adaptar todos los casos presentados. Puesto que el proceso de adaptación utiliza el esquema de los Campos Potenciales como se ha descrito en el apartado 2.4.2 y la prueba se ha diseñado para que dicho esquema no sea capaz de atravesar el pasillo, el sistema *CBR* tampoco es capaz de resolver esta situación, como era de esperar. Los resultados de esta prueba se muestran en la figura 5.11.b.

En este punto se ha utilizado el aprendizaje por experiencia para que el sistema sea capaz de adquirir algún tipo de conocimiento y utilizarlo para resolver la situación recién planteada. Para ello se ha utilizado el sistema *CBR* sin ningún tipo de conocimiento en los entornos más

[3] En realidad, por motivos de implementación la base de casos contiene un único caso, que se corresponde con la situación donde no hay ningún obstáculo en el entorno y el agente avanza en línea recta.

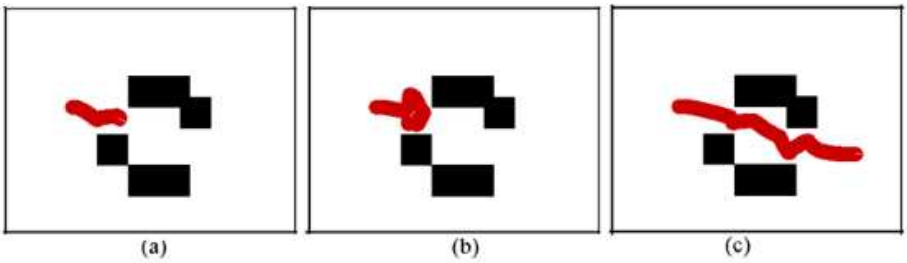

Figura 5.11: Operación del sistema al atravesar un pasillo estrecho: a) esquema de los Campos Potenciales; b) sistema *CBR* sin aprendizaje; c) sistema *CBR* con aprendizaje por experiencia.

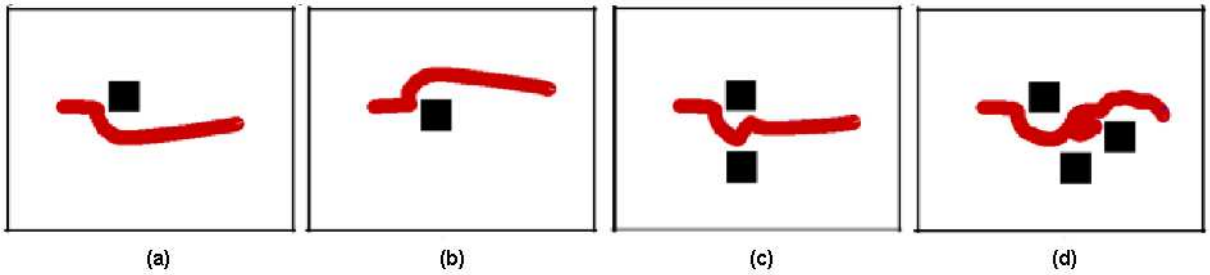

Figura 5.12: Operación del sistema *CBR* sin aprendizaje: a) un obstáculo a la izquierda; b) un obstáculo a la derecha; c) dos obstáculos; d) tres obstáculos.

sencillos mostrados en la figura 5.12, de forma que el sistema puede resolver satisfactoriamente las distintas situaciones presentadas y adquirir un conocimiento de navegación satisfactorio. Durante el proceso de navegación en estos entornos sencillos se ha habilitado el aprendizaje por experiencia, de forma que se han almacenado los casos adaptados en dicho proceso de navegación. Es importante destacar que al no poseer conocimiento alguno las trayectorias generadas en estos entornos sencillos son claramente ineficientes al utilizar únicamente soluciones adaptadas, pero aun así todas ellas son capaces de resolver satisfactoriamente las situaciones planteadas. Una vez recorridas estas trayectorias se ha vuelto a repetir la prueba del pasillo con este conocimiento recién adquirido, cuyo resultado se muestra en la figura 5.11.c. Puesto que el sistema sí posee ya algún conocimiento específico de cómo navegar en presencia de obstáculos, no todas las situaciones presentadas son desconocidas, por lo que no todos los casos fueron adaptados y ese sencillo conocimiento le sirve al agente para atravesar el pasillo.

Resulta interesante analizar el motivo por el cual el sistema es capaz de atravesar el pasillo aun cuando utiliza el esquema de los Campos Potenciales durante su aprendizaje, ya que este esquema es incapaz de resolver dicha situación. La razón se encuentra en que el sistema *CBR* es capaz de utilizar correctamente el conocimiento derivado de la navegación con Campos Potenciales en la resolución de algunos problemas sencillos, como son los planteados en la figura 5.12. Este conocimiento adquirido es el que posteriormente se emplea para resolver la situación más compleja del pasillo, y puesto que se utiliza un conocimiento correcto para esquivar obstáculos en otras circunstancias permite que el sistema atraviese el pasillo satisfactoriamente. La diferencia con el esquema de los Campos Potenciales radica en el hecho de que en la situación del pasillo este esquema adopta otra solución distinta no relacionada con ninguna de las soluciones de

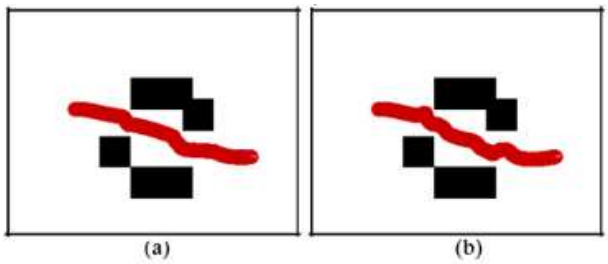

Figura 5.13: Operación del sistema *CBR* al atravesar un pasillo estrecho: a) con aprendizaje por observación a partir del esquema de los Campos Potenciales; b) con aprendizaje por observación a partir de un operador humano.

la figura 5.12, la cual no permite atravesar el pasillo. Es decir, se puede concluir en que la principal diferencia entre el sistema *CBR* y el esquema de los Campos Potenciales radica en el hecho de que el sistema *CBR* aprende y reutiliza correctamente el conocimiento derivado de la resolución de problemas anteriores, mientras que los Campos Potenciales no reutiliza ningún tipo de conocimiento y siempre propone nuevas soluciones para nuevos problemas.

Por último, es preciso matizar que si bien el sistema *CBR* es capaz de atravesar el pasillo satisfactoriamente la trayectoria seguida no es muy eficiente, debido a que el conocimiento adquirido sigue mostrando algunas características de los Campos Potenciales y a que dicho conocimiento es escaso y no cubre completamente la nueva situación planteada. De hecho, un adecuado entrenamiento del sistema mediante la realización de una fase previa de aprendizaje por observación suele mejorar las prestaciones del mismo. Para verificar este hecho se ha realizado última prueba, consistente en atravesar de nuevo el pasillo de la figura 5.11 pero utilizando el conocimiento derivado de un proceso de entrenamiento previo y no únicamente el de la propia experiencia del sistema. La figura 5.13.a muestra dicha prueba tras haber realizado un aprendizaje por observación a partir del esquema de los Campos Potenciales en las trayectorias de la figura 5.6, y la figura 5.13.b muestra la misma prueba pero tras un aprendizaje por observación a partir de un operador humano en las trayectorias de la figura 5.8. En ambas situaciones las trayectorias resultan más suaves que la obtenida únicamente con aprendizaje por experiencia, mostrada en la figura 5.11.c. La trayectoria basada en Campos Potenciales de la figura 5.13.a presenta un mejor comportamiento que la trayectoria basada en un operador humano de la figura 5.13.b porque en el entrenamiento realizado en este último caso no se contemplan casos con dos o tres obstáculos, por lo que el sistema tiene que adaptar bastantes situaciones con posibles soluciones no muy eficientes. En el caso del aprendizaje por observación basado en el esquema de los Campos Potenciales tanto las situaciones con dos como con tres obstáculos ya se han contemplado, por lo que a pesar de la naturaleza oscilatoria de los Campos Potenciales la trayectoria obtenida de la figura 5.13.b es más suave.

Como principal conclusión de esta última prueba se puede afirmar que si bien el aprendizaje por experiencia es importante porque permite mejorar el comportamiento del sistema al adquirir nuevas experiencias, un adecuado aprendizaje por observación es la base para obtener una mayor eficiencia en el sistema desde el primer momento en el que se empieza a utilizar. Es por ello por lo que ambas técnicas de aprendizaje han sido combinadas en el sistema propuesto.

2.5 Optimización de la base de casos

En todo sistema *CBR* es deseable aumentar progresivamente el conocimiento almacenado en el sistema para manejar cada vez mayor número de situaciones en base a una mayor experiencia. En el sistema propuesto, tal y como se ha descrito en el apartado 2.4.3, todos los casos adaptados son evaluados positivamente e incorporados en el sistema. Sin embargo, la incorporación indiscriminada de nuevos casos plantea serios inconvenientes que desembocan en una apreciable degradación del funcionamiento del sistema propuesto:

- Produce un crecimiento incontrolado de la base de casos, lo cual redunda en un aumento del tiempo de respuesta del sistema al emplear una estructura plana como se ha descrito en el apartado 2.3.
- Una vez un caso es almacenado en el sistema puede ser utilizado siempre que se precise. Así pues, aquellos casos poco eficientes que han sido almacenados en el sistema son utilizados una y otra vez cada vez que se plantea un problema similar al que describen, siendo posible que se propongan soluciones poco apropiadas.
- Se favorece la existencia de distintos casos que describen situaciones muy similares pero que presentan soluciones bastante distintas, en función del entrenamiento del sistema, por lo que el sistema puede optar por uno o por otro dependiendo únicamente de las pequeñas variaciones que se presenten en la entrada, produciendo oscilaciones en el comportamiento del sistema.

Es por todo ello por lo que el conocimiento que posee el sistema necesita ser procesado para evitar estos inconvenientes. Con tal fin se ha desarrollado un mecanismo de optimización que favorece los casos más eficientes y penaliza los menos eficientes. Este mecanismo se basa en dividir la base de casos en un conjunto de clases, de forma que todos aquellos casos que describan problemas similares se agrupen conjuntamente en una clase caracterizada por un caso prototipo. Para ello se ha utilizado el algoritmo de segmentación en clases *MaxMin* [Mar93], por tratarse de un algoritmo sencillo que sólo depende de un parámetro, la distancia mínima entre clases $Dist\_Cluster$. Como función de distancia en dicho algoritmo se ha utilizado la distancia de Tanimoto, descrita mediante la expresión 5.5 del apartado 2.4.1.

Para realizar la segmentación en clases de la base de casos se utilizan únicamente las características α_{dest} y S_i del vector de cada caso, que son las que describen completamente un determinado problema. La solución aportada α_{dir} no es utilizada durante este proceso de segmentación, pues los distintos casos deben ser agrupados en función de lo semejantes que sean los problemas que describen, no de las soluciones que adoptan para cada problema.

No obstante, esta estrategia plantea un serio problema a la hora de generar los distintos casos prototipo de cada clase: si bien todos los casos de una misma clase presentan características muy similares que describen el mismo problema, las soluciones aportadas por los distintos casos de una misma clase pueden ser muy diferentes. Así pues, en la generación del prototipo de cada clase es

necesario implementar un esquema que fusione adecuadamente las diversas soluciones propuestas por todos los casos de la misma clase. Para realizar esta fusión y generar adecuadamente el prototipo de cada clase se va a utilizar la eficiencia asociada a cada caso, de forma que el prototipo de la clase X se genera mediante la fusión ponderada de todos los casos de dicha clase según la siguiente expresión:

$$P_X = \sum_{\forall C_i \in X} \eta_i \cdot C_i \tag{5.10}$$

donde η_i representa la eficiencia del caso C_i calculada mediante la expresión 5.9 del apartado 2.4.3. Estos prototipos se convierten en los nuevos casos almacenados en el sistema, los cuales son etiquetados con su correspondiente eficiencia.

Tras generar los correspondientes prototipos se pueden seguir añadiendo nuevos casos mediante los adecuados procesos de aprendizaje por observación y aprendizaje por experiencia. Estos nuevos casos son incorporados en las clases ya existentes, y pueden generar nuevas clases o fusionarse con las clases existentes modificando sus prototipos. La modificación de estos prototipos también puede originar la fusión de algunas clases si estas se aproximan lo suficiente.

Este mecanismo de optimización tiene importantes ventajas:

- El proceso de segmentación reduce el número de casos almacenados en el sistema, en función del número de clases obtenidas en dicho proceso. El parámetro *Dist_Cluster* del algoritmo *MaxMin* influye directamente en el número final de clases obtenido, pues cuanto mayor sea dicho parámetro mayor tamaño tendrán las clases y menor número de clases aparecen en el proceso de segmentación. La relación exacta entre dicho parámetro y el número de clases es complejo de determinar, ya que depende de los casos particulares almacenados en el sistema y de la de distancia entre ellos.
- El prototipo de una clase no se genera simplemente como la media de los casos que lo forman, sino como una media ponderada en función de la eficiencia de cada uno. Así pues, los casos más eficientes colaboran más en la generación del prototipo que los casos menos eficientes. De esta forma el prototipo tenderá a tener también una eficiencia alta, aunque también incluirá información de los casos poco eficientes que puede resultar relevante ante determinadas circunstancias.
- El mecanismo de optimización implementado penaliza aquellos casos con menor eficiencia en el sistema, por lo que aquellas soluciones poco adecuadas se van descartando paulatinamente al contribuir en menor medida al prototipo de cada clase. De hecho, si algún caso es evaluado con una eficiencia nula no contribuye en el proceso de promediado, por lo que es automáticamente descartado. Esta característica es útil en algunas circunstancias donde se almacenan casos erróneos debido por ejemplo a errores en las lecturas de los sensores sonar. Así pues, este mecanismo de optimización implementa un mecanismo de olvido similar al aprendizaje por experiencia negativo que evita la repetición de soluciones poco óptimas.

- El mecanismo de optimización es más eficiente que una mera eliminación de todos aquellos casos con una eficiencia menor que un determinado umbral. Con el mecanismo implementado aquellos casos con baja eficiencia parecidos a otros casos con mayor eficiencia son automáticamente absorbidos en la generación del prototipo, por lo que sólo son descartados cuando existen soluciones alternativas más óptimas. Sin embargo, cuando todos los casos de una clase presentan baja eficiencia el prototipo final poseerá así mismo una baja eficiencia, pero no serán eliminados al no existir una alternativa mejor. De esta forma se preservan aquellos casos que, aunque poco eficientes, sirvieron en su momento para cubrir situaciones que casos más eficientes no podrían manejar, como por ejemplo atravesar pasillos estrechos donde el agente debe aproximarse necesariamente a los obstáculos.

El mecanismo de optimización descrito se puede realizar sobre la base de casos en cualquier momento, aunque es conveniente realizarlo periódicamente tras adquirir un número suficiente de nuevos casos en el sistema tras un proceso de aprendizaje. Cada vez que se realiza un nuevo proceso de optimización se van ajustando los prototipos existentes mediante la incorporación de nuevos casos, por lo que el sistema tiende a converger a soluciones cada vez más eficientes a medida que se realizan sucesivas optimizaciones de la base de casos.

A efectos de comprobar la operación del sistema incorporando el mecanismo de optimización de la base de casos recién descrito se han llevado a cabo diversas pruebas en el mismo, las cuales se describen a continuación.

En primer lugar se ha pretendido valorar la influencia del parámetro *Dist_Cluster* en la actuación del sistema. Para ello se ha llevado a cabo un proceso de entrenamiento exhaustivo del agente con aprendizaje por observación a partir del esquema de los Campos Potenciales, según las trayectorias representadas en la figura 5.14. Para evaluar el comportamiento del mecanismo de optimización en condiciones extremas se ha fijado un valor bastante bajo para el umbral de disparidad U_{disp} del proceso de aprendizaje por observación, que como se describió en el apartado 2.4.4 determina la distancia que debe existir entre dos casos consecutivos para que sean adquiridos. En consecuencia, el número final de casos es bastante alto, llegando a 840 para las trayectorias descritas de la figura 5.14. Estas trayectorias presentan, además, las oscilaciones típicas del esquema de los Campos Potenciales.

A continuación se ha aplicado el mecanismo de optimización de la base de casos obtenida tras el proceso de entrenamiento. Para ello se han fijado heurísticamente los distintos parámetros genéricos de la expresión 5.9 del apartado 2.4.3 a los siguientes valores:

$$\begin{cases} F_{soft}[|\alpha_{dir}| = 30^{o}] = 0.5 & \Rightarrow \quad C_{soft} = 2.3 \cdot 10^{-2} \\ F_{dist}[|\alpha_{dest} - \alpha_{dir}| = 30^{o}] = 0.5 & \Rightarrow \quad C_{dist} = 2.3 \cdot 10^{-2} \\ F_{sec}[|\alpha_{min} - \alpha_{dir}| = 30^{o}, d_{min} = 400mm] = 0.5 & \Rightarrow \quad C_{sec} = 5.8 \cdot 10^{-5} \end{cases} \tag{5.11}$$

Así mismo, los coeficientes de ponderación de los distintos factores de dicha expresión se han igualado entre sí, puesto que en esta prueba no se intenta determinar la importancia relativa

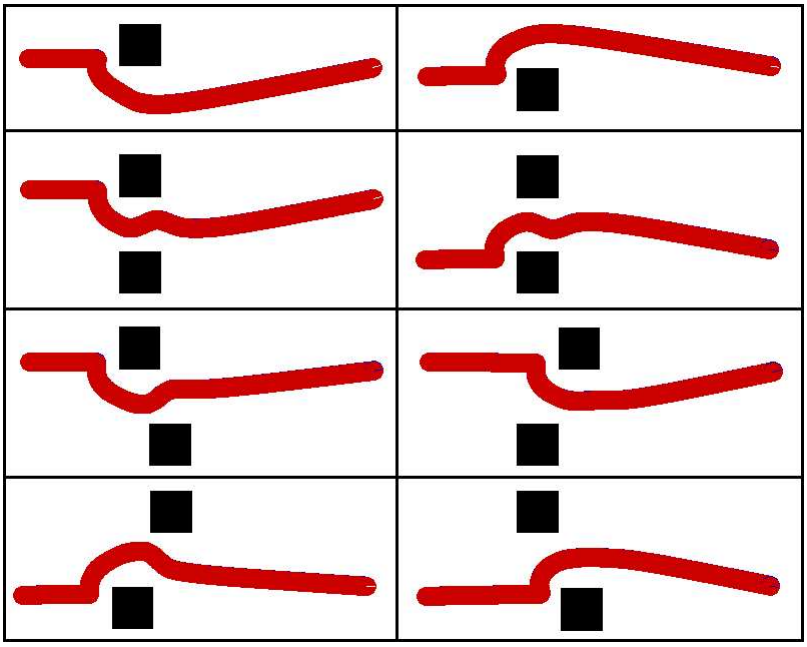

Figura 5.14: Aprendizaje por observación a partir del esquema de los Campos Potenciales.

Dist_Cluster	**Número de casos**
500	289
2500	144
10000	39

Tabla 5.2: Número de casos obtenidos tras el proceso de optimización de la base de casos en función del parámetro *Dist_Cluster*.

de unos parámetros respecto a otros: K_{soft}=K_{dist}=K_{sec}=1. Tras ajustar los valores necesarios de la expresión utilizada para el cálculo de la eficiencia de cada caso se ha procedido a variar progresivamente el valor del parámetro *Dist_Cluster*. Recuérdese que a mayor valor de dicho parámetro menor número de clases aparecen en el proceso de segmentación. El número de casos obtenidos finalmente en función del parámetro *Dist_Cluster* se muestra en la tabla 5.2.

Es interesante observar en primer lugar la importante reducción conseguida en el tamaño de la base de casos, pasando de los 840 casos originales a 289 casos, 144 casos y 39 casos, lo que supone un 34%, un 17% y un 5% respectivamente del tamaño original. Tras analizar detenidamente el contenido de estas nuevas bases de casos obtenidas mediante este mecanismo de optimización se ha comprobado que cuanto mayor es el número de clases, es decir, el tamaño final de la base de casos, menor número de casos contiene cada clase, como era de esperar, y más se parecen los prototipos finales a casos reales.

A continuación se ha procedido a aplicar el conocimiento obtenido tras el proceso de optimización en la navegación del agente a través de un nuevo entorno que no se correspondiese con ninguna de las trayectorias de entrenamiento de la figura 5.14. La figura 5.15 muestra los resultados obtenidos en dicho entorno para la base de casos original y para cada una de las bases de casos obtenidas tras el proceso de optimización al variar el parámetro *Dist_Cluster*.

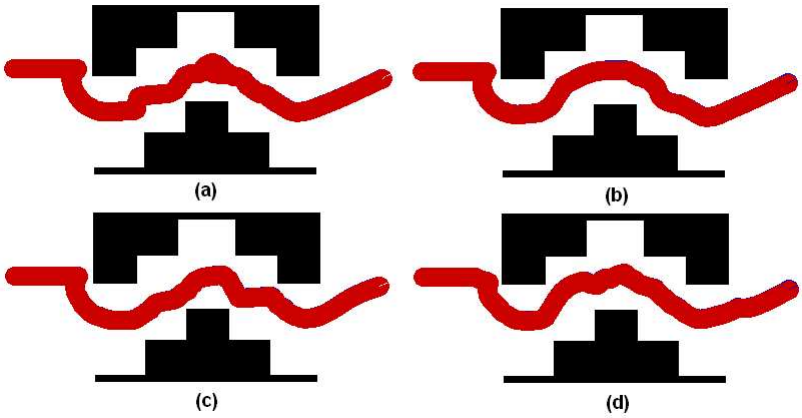

Figura 5.15: Influencia del proceso de optimización de la base de casos en la operación del sistema: a) base de casos original de 840 casos; b) base de casos optimizada con $Dist\_Cluster$=500 y 289 casos; c) base de casos optimizada con $Dist\_Cluster$=2500 y 144 casos; d) base de casos optimizada con $Dist\_Cluster$=10000 y 39 casos.

La figura 5.15.a se corresponde con la base de casos original de 840 casos, mientras que las figuras 5.15.b, 5.15.c y5.15.d se corresponden con las bases de casos optimizadas de 289 casos, 144 casos y 39 casos respectivamente. Como deja de manifiesto la trayectoria de la figura 5.15.a, un excesivo número de casos puede ser contraproducente en la operación del sistema, pues se alterna entre casos parecidos que pueden proponer soluciones muy distintas en función de las pequeñas variaciones de los datos de entrada. Esto suele provocar efectos no deseados como oscilaciones y bucles o giros completos del agente. Así pues, un conocimiento más compacto puede mejorar la operación del sistema, como muestra la trayectoria de la figura 5.15.b. Es importante observar que aunque el conocimiento del sistema ha sido considerablemente reducido el comportamiento final sigue siendo bastante adecuado. Este hecho se debe a que el mecanismo de optimización implementado es bastante eficaz, pues permite compactar adecuadamente la información al fusionar distintos casos favoreciendo siempre los más eficientes.

Sin embargo, a medida que se reduce el número de casos en la base de casos las acciones emprendidas por el agente difieren más de las que adquirió en el proceso de entrenamiento, como era de esperar, ya que cada prototipo es el resultado de promediar un mayor número de casos. Por lo tanto, las adaptaciones necesarias aumentan al presentarse mayor número de situaciones desconocidas. Estas nuevas situaciones desconocidas se resuelven mediante la generación de nuevos casos adaptados, que suelen introducir cambios de dirección algo bruscos debido a la repulsión que generan los obstáculos del entorno en el mecanismo de adaptación, como se ha descrito en el apartado 2.4.2. En las trayectorias de las figuras 5.15.b, 5.15.c y 5.15.d se han adaptado 230, 260 y 323 nuevos casos respectivamente. Se puede apreciar claramente que la trayectoria más suave se corresponde con la situación de la figura 5.15.b, donde el conocimiento del sistema es mayor y por lo tanto las adaptaciones necesarias se reducen.

A la vista de estos resultados se puede concluir que el funcionamiento del sistema es óptimo cuando el conocimiento no es ni demasiado elevado ni demasiado escaso. Demasiados casos producen un comportamiento excesivamente sensible a los datos de entrada, con resultados no

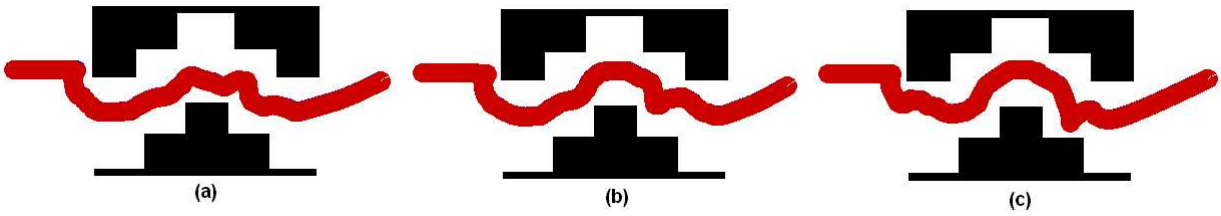

Figura 5.16: Influencia de los criterios de eficiencia en la operación del sistema: a) suavidad; b) distancia; c) seguridad.

deseados como oscilaciones, aunque una excesiva compactación de la base de casos redunda en un conocimiento más genérico y menos específico, lo que provoca cierta brusquedad en las trayectorias generadas al presentarse más situaciones desconocidas y aumentar el número de casos adaptados. El valor óptimo de compactación dependerá de la aplicación particular que se esté implementando y del grado de abstracción o generalización que se desee en la base de casos optimizada. Para el sistema desarrollado se ha comprobado empíricamente que el mejor funcionamiento se obtiene cuando la base de casos contiene alrededor de 300 casos.

Por último, se ha realizado otra prueba para evaluar la influencia de los distintos coeficientes de ponderación de la expresión 5.9 del apartado 2.4.3 en la actuación del agente: K_{soft} para la suavidad de la trayectoria final, K_{dist} para la distancia recorrida y K_{sec} para la seguridad en la aproximación a los obstáculos del entorno [4]. Para ello se ha aplicado el mecanismo de optimización sobre la base de casos original de 840 casos variando el valor de estos coeficientes de ponderación, de forma que sólo uno de ellos tenga influencia en el proceso:

$$\begin{cases} Suavidad & \Rightarrow & (K_{soft}, K_{dist}, K_{sec}) = (1,0,0) \\ Distancia & \Rightarrow & (K_{soft}, K_{dist}, K_{sec}) = (0,1,0) \\ Seguridad & \Rightarrow & (K_{soft}, K_{dist}, K_{sec}) = (0,0,1) \end{cases} \tag{5.12}$$

Una vez generadas las bases de datos optimizadas con estos parámetros se ha repetido la prueba de navegación sobre el entorno del pasillo de la figura 5.15, mostrándose los resultados obtenidos en la figura 5.16. Sin embargo, como se puede apreciar en esta figura no parece evidente la influencia de estos parámetros sobre las trayectorias generadas. Este hecho se debe a la inevitable aparición de casos adaptados en el sistema ante situaciones desconocidas. Es necesario recordar que estos casos adaptados son siempre aceptados como solución al problema presentado independientemente de su eficiencia, que pueden incluir por supuesto situaciones poco eficientes con cambios bruscos. Por lo tanto, si bien los casos contenidos en la base de casos han sido específicamente optimizados siguiendo alguno de los criterios de eficiencia perseguidos (suavidad, distancia o seguridad), los nuevos casos adaptados no favorecen ninguno de estos criterios hasta que se realice el proceso de optimización.

[4] En esta prueba el parámetro *Dist_Cluster* se ha fijado al valor 500, para permitir que exista un número representativo de casos.

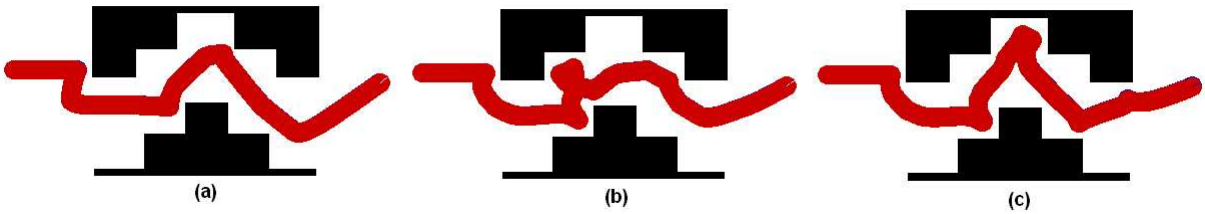

Figura 5.17: Influencia de los coeficientes de ponderación de la función de eficiencia sin adaptación: a) suavidad; b) distancia; c) seguridad.

Para intentar eliminar esta interferencia de los casos adaptados en las pruebas realizadas, se ha procedido a desactivar la capacidad de adaptación del sistema. Es preciso indicar que sin el proceso de adaptación el sistema no es capaz de resolver nuevas situaciones desconocidas, por lo que su comportamiento no será adecuado. Este escenario, pues, no se corresponde con una situación real, por lo que los resultados obtenidos al repetir las pruebas de la figura 5.16 eliminando el proceso de adaptación se muestran en la figura 5.17. Los cambios bruscos que muestran estas trayectorias no los generan ya los casos adaptados, sino el sistema de alarma que posee el agente y que se describió en el apartado 1.1 del capítulo 4. Recuérdese que dicho sistema de alarma detiene al agente cuando se detecta un obstáculo por debajo de un umbral de seguridad U_{sec}, y hace que gire hasta que dicho obstáculo desaparece de la dirección de avance.

La trayectoria de la figura 5.17.a utiliza los casos obtenidos tras el proceso de optimización considerando únicamente el factor de suavidad. En este caso el sistema tiende a minimizar los cambios efectuados recorriendo líneas rectas. La trayectoria de la figura 5.17.b emplea aquellos casos obtenidos tras el proceso de optimización considerando únicamente el factor de distancia. Ahora el sistema tiende a dirigirse directamente hacia el destino final independientemente de la brusquedad en los cambios de dirección. Por último, la trayectoria de la figura 5.17.c emplea los casos obtenidos tras el proceso de optimización considerando únicamente el factor de la seguridad. Independientemente de la inevitable proximidad a los obstáculos resuelta por el sistema de alarma, el sistema intenta navegar separándose de los obstáculos más cercanos.

Los valores escogidos para los coeficientes de ponderación durante estas pruebas presentan situaciones extremas donde únicamente uno de los parámetros de suavidad, distancia y seguridad se ha considerado simultáneamente. Dependiendo de la aplicación puede generarse un conjunto de valores adecuados para estos coeficientes de forma que se combinen sus efectos.

Por último, es importante destacar que aunque los resultados obtenidos en este apartado parezcan poco eficientes, este hecho se debe a que el proceso de entrenamiento del agente y el entorno de prueba son muy diferentes, por lo que el sistema debe recurrir frecuentemente a la generación de soluciones adaptadas para resolver situaciones desconocidas. Las pruebas se han diseñado intencionadamente de esta forma para poner de manifiesto que incluso en situaciones extremas como las aquí presentadas el sistema es capaz de optimizar su funcionamiento y adaptarse según los criterios de suavidad, distancia y seguridad deseados. Indudablemente, si se somete al sistema a un proceso de entrenamiento más exhaustivo que incluya pasillos se pueden obtener trayectorias mucho más eficientes y adecuadas.

3 Implementación de la Capa de Navegación

En función de la información sensorial captada del entorno por el agente el esquema de navegación debe generar los comandos de movimiento adecuados para alcanzar el destino final evitando los posibles obstáculos del entorno. En un sistema híbrido como el que se está desarrollando el esquema de navegación se basa en un paradigma reactivo por las excelentes prestaciones temporales de este tipo de sistemas, permitiendo una navegación segura en entornos dinámicos con presencia de obstáculos tanto fijos como móviles.

El objetivo de la Capa de Navegación es implementar el esquema de evitación de obstáculos apropiado para alcanzar con seguridad y sin riesgo de colisiones el destino final. De los distintos paradigmas existentes se ha implementado el esquema analítico de los Campos Potenciales y el esquema basado en *CBR*.

3.1 Módulo *Navigation*

La Capa de Navegación se va a implementar mediante el módulo *Navigation*, que deberá desarrollar las siguientes funciones básicas:

- **Alcanzar el destino:** responsable de generar los comandos de movimiento adecuados para que el agente se dirija hacia el destino final especificado.
- **Evitación de obstáculos:** responsable de evitar la colisión con los posibles obstáculos fijos y/o móviles del entorno mientras el agente se dirige al destino final.

La operación del esquema de navegación implementado trata de alcanzar el destino final especificado en línea recta. Para ello se calcula en primer lugar el siguiente vector de atracción normalizado que apunta hacia el destino final:

$$\vec{V}_{atrac} = \frac{\vec{P}_{dest} - \vec{P}_{act}}{|\vec{P}_{dest} - \vec{P}_{act}|} \tag{5.13}$$

donde $\vec{P}_{dest}$ es el vector asociado con el destino final y $\vec{P}_{act}$ es el vector asociado con la posición actual del agente. Una vez calculado el vector de atracción normalizado hacia el destino final $\vec{V}_{atrac}$ se puede calcular la dirección de avance α_{dir} como el ángulo formado entre la orientación actual del agente y la dirección en la que apunta el vector $\vec{V}_{atrac}$, proceso que se muestra en la figura 5.18.

Mientras el agente se dirige en línea recta hacia el destino final monitoriza la distancia a la que se encuentran los distintos obstáculos del entorno a través de las lecturas proporcionadas por sus sensores. Si el agente se aproxima a los obstáculos del entorno por debajo de un umbral de proximidad U_{prox} se activa el mecanismo de evitación de obstáculos del sistema, procediendo a

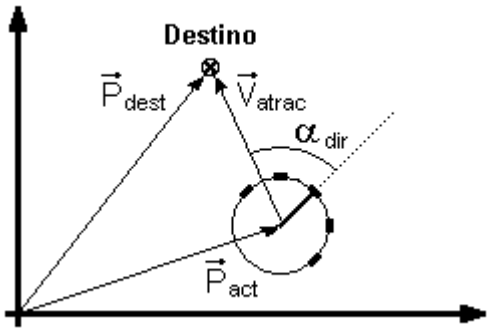

Figura 5.18: Cálculo de la dirección de avance en línea recta el destino final.

calcular la nueva dirección de avance para alcanzar el destino final mientras se trata de esquivar los obstáculos del entorno. El cálculo de la dirección de avance correcta en estas circunstancias depende del esquema de evitación de obstáculos implementado, que en el sistema desarrollado son el esquema de los Campos Potenciales y el esquema *CBR*:

- **Campos Potenciales:** descrito en el apartado 1.1.1. Consiste en generar la dirección de avance mediante la composición de dos fuerzas, una de atracción hacia el destino especificado y otra de repulsión hacia los obstáculos del entorno:

$$\vec{F} = \vec{F}_{atrac} + \vec{F}_{rep} \tag{5.14}$$

La fuerza de atracción se genera a partir del vector de atracción normalizado $\vec{V}_{atrac}$ de la expresión 5.13 que pretende alcanzar el destino en línea recta. La fuerza de repulsión se calcula mediante un vector de repulsión normalizado que considera los obstáculos circundantes que existen en el entorno. Dicho vector de repulsión normalizado se calcula como:

$$\vec{V}_{rep} = \frac{\sum_{i=1}^{N} \vec{V}_i}{|\sum_{i=1}^{N} \vec{V}_i|} \tag{5.15}$$

donde N representa el número de sensores del agente, y $\vec{V}_i$ representa un vector de repulsión para cada una de las lecturas de los sensores que detecta el agente y que se encuentra por debajo del umbral de proximidad U_{prox}. Estos parámetros se representan en la figura 5.19.

Cada uno de los vectores de repulsión $\vec{V}_i$ debe ser generado de forma que disminuya su efecto a medida que aumente la distancia entre el agente y el obstáculo. Existen numerosas funciones que pueden generar un comportamiento de este tipo en función de la distancia d al obstáculo, como funciones proporcionales a $(1/d^2)$ inspiradas en la electrostática, funciones proporcionales a $(1/d)$ similares a la anterior pero con un comportamiento más suave, o funciones proporcionales a $(1-d)$ con un comportamiento lineal todavía más suave. Debido a que proporcionan comportamientos más suaves que tienden a reducir

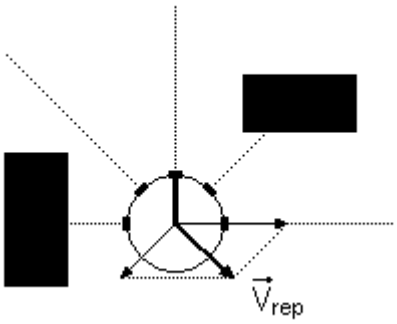

Figura 5.19: Cálculo del vector de repulsión de los obstáculos próximos mediante Campos Potenciales.

las típicas oscilaciones producidas por los Campos Potenciales se ha escogido una función proporcional a $(1-d)$, según la siguiente expresión:

$$\vec{V}_i = -\frac{U_{prox} - S_i}{U_{prox} - U_{sec}}\vec{\theta}_i \tag{5.16}$$

donde U_{prox} es el umbral de proximidad, U_{sec} es el umbral de seguridad del sistema de alarma descrito en el apartado 1.1 del capítulo 4, S_i representa la lectura del sensor i, y $\vec{\theta}_i$ representa un vector unitario que apunta en la dirección del sensor i. El signo negativo de la expresión se debe a que el vector calculado es repulsivo. Esta expresión es máxima con un valor igual a 1 cuando el obstáculo se encuentra a la distancia determinada por el umbral de seguridad U_{sec}, y es mínima con un valor igual a 0 cuando el obstáculo se encuentra a la distancia determinada por el umbral de proximidad U_{prox}. Es necesario recordar que si algún sensor devuelve una lectura por debajo del umbral de seguridad U_{sec} se activa el sistema de alarma descrito, y que aquellos sensores cuya lectura se encuentre por encima del umbral de proximidad U_{prox} no intervienen en el cálculo del vector de repulsión $\vec{V}_{rep}$, por lo que el módulo de la expresión 5.16 nunca puede tomar valores negativos.

Finalmente, a partir de los vectores de atracción y de repulsión normalizados se obtienen las correspondientes fuerzas de atracción y repulsión, cuya composición determinará la dirección correcta de avance:

$$\begin{cases} \vec{F}_{atrac} = G_{attrac} \cdot \vec{V}_{atrac} \\ \\ \vec{F}_{rep} = G_{rep} \cdot \vec{V}_{rep} \end{cases} \tag{5.17}$$

donde G_{attrac} y G_{rep} son parámetros genéricos de ganancia que permiten ponderar adecuadamente la relevancia relativa que se le quiere otorgar a la fuerza de atracción frente a la fuerza de repulsión, permitiendo que el agente se aproxime en mayor o menor medida a los obstáculos del entorno en función de estos parámetros.

- **Razonamiento basado en casos:** descrito en el apartado 2. La dirección correcta de avance para alcanzar el destino final mientras se evitan los obstáculos próximos del entorno

es directamente la salida proporcionada por el sistema *CBR*. Para implementar el paradigma *CBR* se ha utilizado el trabajo realizado por el grupo *KEMLg* (*Knowledge Engineering & Machine Learning Group*) de la *Universitat Politècnica de Catalunya* [SMCR+97]. Dicho trabajo implementa un servidor *CBR* que devuelve el caso más parecido de los que posee el sistema una vez presentado un caso a la entrada, todo ello a través de una conexión remota. Este servidor ha sido convenientemente adaptado a las particularidades del sistema diseñado en la presente Tesis según se ha descrito en el apartado 2, y se ha denominado *CBRServer*.

Naturalmente es posible implementar más esquemas de navegación reactiva modificando únicamente el módulo *Navigation*. El disponer de distintos esquemas de navegación permite alternar entre ellos en el momento que se desee, permitiendo una mayor flexibilidad en el proceso de navegación. En la presente Tesis el esquema de navegación utilizado se basa en el paradigma *CBR*, si bien se ha utilizado puntualmente el esquema de los Campos Potenciales para generar algunas de las trayectorias de entrenamiento de dicho paradigma.

Una vez determinada la dirección correcta de avance es necesario generar los comandos de movimiento adecuados para que el agente la siga correctamente. Estos comandos de movimiento están compuestos por dos velocidades distintas, que son las que posteriormente se enviarán al agente. Recuérdese que el servidor hardware del robot *RobotServer* descrito en el apartado 4.2.2 del capítulo 3 es el encargado de controlar y gestionar el acceso a la plataforma robótica, por lo que si dicha plataforma no admite comandos de velocidad será este servidor el encargado de traducirlos a los comandos que admita la plataforma empleada. Estas dos velocidades que se calculan a partir de la dirección de avance son:

- **Velocidad de avance:** velocidad de traslación que el agente debe seguir, considerando el sentido de avance como positivo y el sentido de retroceso como negativo.
- **Velocidad de rotación:** velocidad de giro que el agente debe seguir, considerando el giro en el sentido contrario a las agujas del reloj como positivo y el giro en el sentido de las agujas del reloj como negativo.

La obtención de estas dos velocidades a partir de la dirección de avance es extremadamente sencilla, pues únicamente hay que descomponer el vector que indica la dirección de avance en dos componentes ortogonales: una en la dirección de avance del agente que indicará la velocidad de avance y otra en la dirección perpendicular a la de avance del agente que indicará la velocidad de rotación.

El interfaz que define los datos de entrada y salida que este módulo comparte con el resto de módulos es el siguiente:

- **Datos de entrada:**
  - **Posición del agente:** generada por el módulo *IFRobot*.

Pentium 4 - 2.4 GHz - 512 MB RAM				
	8 sensores	16 sensores	24 sensores	32 sensores
Dirección avance	$1.012 \pm 0.019\ \mu s$	$1.015 \pm 0.018\ \mu s$	$1.025 \pm 0.018\ \mu s$	$1.034 \pm 0.018\ \mu s$
Dirección *CBR*	$3.64 \pm 0.09\ ms$	$4.67 \pm 0.04\ ms$	$5.84 \pm 0.04\ ms$	$7.14 \pm 0.07\ ms$
Pentium 3 - 1.0 GHz - 256 MB RAM				
	8 sensores	16 sensores	24 sensores	32 sensores
Dirección avance	$2.17 \pm 0.04\ \mu s$	$2.19 \pm 0.04\ \mu s$	$2.21 \pm 0.05\ \mu s$	$2.22 \pm 0.03\ \mu s$
Dirección *CBR*	$3.79 \pm 0.15\ ms$	$4.82 \pm 0.20\ ms$	$6.19 \pm 0.13\ ms$	$7.34 \pm 0.18\ ms$
Pentium 3 - 500 MHz - 192 MB RAM				
	8 sensores	16 sensores	24 sensores	32 sensores
Dirección avance	$2.43 \pm 0.05\ \mu s$	$2.44 \pm 0.05\ \mu s$	$2.45 \pm 0.05\ \mu s$	$2.48 \pm 0.05\ \mu s$
Dirección *CBR*	$3.97 \pm 0.23\ ms$	$4.95 \pm 0.14\ ms$	$6.32 \pm 0.14\ ms$	$7.45 \pm 0.18\ ms$

Tabla 5.3: Prestaciones temporales del proceso de navegación reactiva.

  - **Lectura de los sensores:** generadas por el módulo *IFRobot*.
  - **Destino final:** generado por el módulo *IFUser* si el sistema se encuentra en el nivel de Navegación Reactiva o por el módulo *PathPlanning* si el sistema se encuentra en el nivel de Navegación Planificada o en el nivel de Navegación Topológica.

- **Datos de salida:**

  - **Comandos de movimiento:** utilizados por el módulo *IFRobot*.

Por último, se van a analizar las prestaciones temporales del proceso de navegación reactiva. Obviamente estas prestaciones dependen claramente del número de sensores del agente y de los recursos de la máquina empleada para implementar el sistema de navegación, por lo que se ha repetido la misma prueba para distinto número de sensores y para distintos tipos de máquinas: 8, 16, 24 y 32 sensores, utilizando en cada caso una máquina de altas prestaciones (Pentium 4 - 2.4 GHz - 512 MB RAM), una máquina de prestaciones medias (Pentium 3 - 1.0 GHz - 256 MB RAM) y una máquina de bajas prestaciones (Pentium 3 - 500 MHz - 192 MB RAM). En todas las pruebas realizadas el servidor *CBRServer* se ha ubicado en una máquina de altas prestaciones (Pentium 4 - 2.4 GHz - 512 MB RAM), pues constituye un elemento crítico en el sistema de navegación implementado, y tras el proceso de optimización la base de casos contiene unos 250 casos. El tiempo medio de procesamiento empleado se muestra en la tabla 5.3, donde los errores se han calculado sobre un intervalo de confianza del 95%.

Puesto que el proceso de navegación es distinto si hay que evitar la presencia de obstáculos o no la tabla 5.3 muestra dos tipos de información distinta. Los resultados etiquetados como *Dirección avance* muestran el tiempo medio empleado en calcular la dirección de avance para alcanzar el destino final en línea recta sin la presencia de obstáculos en el entorno, es decir, cuando ninguna lectura de los sensores se encuentra por debajo del umbral de proximidad U_{prox}. El retardo medio del proceso de navegación en estas circunstancias no supera en ningún caso los 2.5 μs, pudiendo considerarse pues prácticamente despreciable.

Los resultados etiquetados como *Dirección CBR* muestran el tiempo medio empleado en calcular la dirección de avance para alcanzar el destino final utilizando el sistema *CBR* en presencia de obstáculos en el entorno, es decir, cuando alguna lectura de los sensores se encuentra por debajo del umbral de proximidad U_{prox}. Este proceso implica no sólo obtener del sistema *CBR* una respuesta ante el caso actual, sino también adaptar los casos devueltos cuando sea necesario. Incluso en condiciones extremas con 32 sensores y utilizando una máquina de bajas prestaciones el retardo medio del proceso de navegación está alrededor de los 7.5 ms, lo que implica que a una velocidad del agente de unos 300 mm/s se recorren aproximadamente 2 mm antes de generar los comandos adecuados de movimiento para la evitación de los obstáculos del entorno. En el sistema desarrollado el agente utilizado posee 8 sensores, por lo que incluso utilizando una máquina de bajas prestaciones el retardo medio está alrededor de los 4 ms, es decir, un desplazamiento del agente de 1 mm a una velocidad de 300 mm/s. Es importante notar que estos tiempos son muy bajos, del orden de los milisegundos, a pesar de que el servidor *CBRServer* se encuentra en una máquina remota del sistema, siendo necesario enviar y recibir a través de la red los datos de entrada y salida del sistema *CBR*.

Estos resultados ponen de manifiesto que el proceso de navegación reactiva es capaz de operar prácticamente en tiempo real, respondiendo instantáneamente a los distintos estímulos procedentes del entorno.

4 Conclusiones

En el presente capítulo se ha abordado el diseño de la Capa de Navegación del sistema, la cual es responsable de implementar el comportamiento reactivo que permite alcanzar un destino final evitando los obstáculos fijos y/o móviles del entorno. Un estudio de las principales alternativas existentes a la hora de implementar esquemas de navegación reactiva revela que los distintos esquemas se dividen en dos grandes grupos: **esquemas analíticos** y **esquemas basados en aprendizaje**.

Los esquemas analíticos pretenden resolver el problema de la navegación reactiva mediante la determinación de una función analítica que modele el comportamiento de navegación. Existen diferentes alternativas para modelar este comportamiento analíticamente, siendo la más conocida la de los Campos Potenciales. Estos esquemas suelen ser excesivamente rígidos al intentar modelar todas las particularidades propias de la navegación mediante una única función, resultando por lo general bastante complejo determinar dicha función y optimizar aquellos parámetros que poseen de forma genérica para todas las circunstancias con las que se puede encontrar el agente.

Los esquemas basados en aprendizaje no utilizan un modelo explícito del comportamiento de navegación que se desea implementar, sino que tratan de generalizarlo implícitamente tras un proceso de entrenamiento. Suelen ser bastante más flexibles que los esquemas analíticos, pues permiten particularizar el comportamiento del agente para determinadas circunstancias mediante un adecuado entrenamiento. Además, permiten incorporar implícitamente restricciones cinemáticas durante el proceso de entrenamiento. Existen así mismo distintas alternativas a la

hora de implementar un esquema basado en aprendizaje, aunque debido a su potencial y versatilidad se ha optado por utilizar el paradigma del razonamiento basado en casos (*CBR*) frente a otras alternativas posibles como el razonamiento basado en reglas o las redes neuronales.

El paradigma *CBR* se basa en modelos cognitivos del razonamiento del ser humano, y utiliza para la resolución de nuevos problemas el conocimiento derivado de la experiencia en la resolución de problemas anteriores. Estas experiencias pasadas en la resolución de problemas se representan mediante lo que se conoce como **casos**, los cuales describen el problema resuelto así como la solución adoptada en su resolución. Todos los casos se almacenan conjuntamente en la **base de casos**, que es la estructura que contiene todo el conocimiento que posee el sistema para abordar la resolución de nuevos problemas. Tanto la información contenida en cada caso para describir una experiencia como la estructura utilizada en la base de casos deben ser cuidadosamente diseñadas en todo sistema *CBR*, existiendo distintas alternativas que deben ser correctamente seleccionadas.

La operación de un sistema *CBR* se sintetiza en lo que se conoce como **ciclo *CBR***, que está compuesto de cuatro fases: **recuperar** la experiencia pasada más similar al nuevo problema a resolver, **reusar** la solución propuesta adaptándola si es necesario al problema actual, **revisar** si la nueva solución adaptada es adecuada en la resolución del nuevo problema y **retener** la nueva solución si es adecuada para ser utilizada en la resolución de futuros problemas. Cada una de estas fases presenta a su vez diversas alternativas, y la correcta elección de las mismas determinarán las prestaciones finales exhibidas por el sistema.

El paradigma *CBR* posibilita el **aprendizaje** continuo de nuevas experiencias, permitiendo así mejorar las prestaciones finales del sistema a medida que aumenta el conocimiento almacenado. Existen básicamente dos esquemas distintos de aprendizaje: el **aprendizaje por observación** que incorpora nuevas soluciones tras un período de entrenamiento, y el **aprendizaje por experiencia** que incorpora las propias soluciones generadas por el sistema al afrontar una situación no contemplada en el conocimiento del mismo. El aprendizaje por observación dota de una gran flexibilidad al sistema, pues permite incorporar distintos comportamientos específicos para determinadas circunstancias. El aprendizaje por experiencia permite ir aumentando paulatinamente el propio conocimiento del sistema, por lo que se puede conseguir un comportamiento cada vez más óptimo conforme el sistema afronta más problemas.

Para mantener acotado el número de casos almacenados en el sistema y para aumentar la eficiencia en la operación del mismo se ha implementado un mecanismo de **optimización** de la base de casos, el cual permite compactar la información contenida en el sistema a la vez que se favorecen los casos más eficientes y se penalizan los menos eficientes. La eficiencia se puede medir en función de diversos criterios, como son **suavidad**, **distancia** y **seguridad** en la navegación. Estos tres criterios se pueden combinar adecuadamente según los objetivos perseguidos en el operación del sistema.

Por último, se ha descrito la implementación de la Capa de Navegación mediante el módulo *Navigation*, responsable de desarrollar la navegación reactiva para alcanzar el destino final mientras se evitan los obstáculos fijos y/o móviles del entorno. Dicho módulo implementa tanto el es-

quema analítico de los Campos Potenciales como el esquema basado en aprendizaje mediante el paradigma *CBR*, siendo posible conmutar entre uno u otro según se desee. Las pruebas realizadas demuestran que el esquema reactivo de navegación implementado es muy rápido, permitiendo una operación prácticamente en tiempo real, como corresponde a un esquema reactivo.

Capítulo 6

Capa de Planificación

Tras implementar el resto de capas del sistema, se va a realizar el diseño de la Capa de Planificación, responsable de desarrollar los comportamientos deliberados de planificación que implementan los niveles de Navegación Planificada y Navegación Topológica del sistema.

Este capítulo se ha organizado como se indica a continuación. El apartado 1 realiza una introducción a la planificación deliberada en el desarrollo de agentes con capacidades avanzadas de navegación, describiendo los distintos niveles de planificación que se necesitan en el sistema. El apartado 2 analiza la implementación de un planificador que calcule un camino libre de obstáculos en el entorno conocido para alcanzar el destino final, presentando las nuevas funcionalidades incorporadas en el sistema. El apartado 3 describe la implementación de un planificador que genere una representación topológica del entorno que permita descomponerlo en regiones y generar rutas a través de las mismas, analizando las nuevas funcionalidades incorporadas en el sistema. El apartado 4 realiza una comparativa entre las principales características de los niveles de planificación desarrollados. El apartado 5 describe la implementación de la Capa de Planificación del sistema. Por último, en el apartado 6 se describen las principales conclusiones extraídas a lo largo del presente capítulo.

1 Introducción

La capacidad de navegación implica determinar la secuencia de movimientos que el agente debe llevar a cabo para alcanzar el destino final sin colisionar con los obstáculos del entorno, lo cual ha constituido tradicionalmente un problema bastante complejo de resolver [Rei79, Lat91].

La Capa de Navegación incorporada en el sistema permite resolver el problema de la evitación de obstáculos durante el proceso de navegación, si bien posee una serie de limitaciones:

- El carácter local de la navegación reactiva no garantiza que se puedan alcanzar aquellos destinos finales que se encuentren más allá del entorno inmediato del agente. De hecho, uno de los mayores inconvenientes de la navegación puramente reactiva radica en la exis-

tencia de situaciones donde el agente puede quedar bloqueado, situaciones conocidas como trampas de mínimos locales.

- La falta total de planificación imposibilita la operación eficiente del sistema mediante la optimización de algunos parámetros globales como la distancia recorrida o el tiempo requerido durante el proceso de navegación.

En estas circunstancias, una aproximación híbrida que combine adecuadamente comportamientos reactivos con algún tipo de planificación deliberada de alto nivel constituye la mejor alternativa si se pretende añadir una mayor eficiencia en la operación del sistema, tal y como se ha analizado en el apartado 2.1 del capítulo 2. Este proceso de planificación requiere una adecuada representación de la realidad sobre la cual operar, lo que permite incorporar en el sistema cierta capacidad de abstracción que posibilita la consecución de objetivos más elaborados que la simple navegación reactiva. De hecho, una adecuada planificación es indispensable para desarrollar los niveles superiores de la Jerarquía de Navegación presentada en el apartado 2.2 del capítulo 1.

La Capa de Planificación va a ser la responsable de implementar todos aquellos procesos de planificación necesarios en el sistema para el desarrollo de sus funciones, generando los planes de actuación adecuados que permitirán al agente la consecución de determinados objetivos. Estos planes generados por la Capa de Planificación se utilizarán posteriormente para dirigir la operación de la Capa de Navegación, posibilitando así la implementación de comportamientos de navegación más complejos y eficientes. Dependiendo del grado de abstracción que se desee alcanzar se pueden implementar distintos procesos de planificación, teniendo en cuenta que a mayor nivel de abstracción los planes generados tienen mayor generalidad y suelen ser menos eficientes, aunque permiten desarrollar comportamientos de navegación más complejos.

Tal y como se ha descrito en el apartado 4.1 del capítulo 3 los niveles de navegación que debe implementar el sistema para alcanzar los niveles superiores de la Jerarquía de Navegación son el nivel de Navegación Planificada y el nivel de Navegación Topológica. Los procesos de planificación necesarios para implementar estos niveles de navegación son los siguientes:

- **Planificación de caminos:** necesario para implementar el nivel de Navegación Planificada del sistema, responsable de determinar un camino libre de obstáculos sobre la representación del entorno que posee el sistema para alcanzar el destino final.
- **Planificación de rutas:** necesario para implementar el nivel de Navegación Topológica del sistema, responsable de descomponer la representación del entorno que posee el sistema en un conjunto de regiones tanto exploradas como no exploradas, determinando las distintas regiones a atravesar hasta alcanzar el destino final.

Como ha sido analizado en el apartado 2.1 del capítulo 2 cabe recordar que el principal inconveniente de los procesos de planificación radica en el coste temporal que requiere la elaboración de los planes adecuados para alcanzar los objetivos marcados. Por lo tanto, la principal dificultad a la hora de incorporar estos procesos de planificación en el sistema radica en combinarlos

adecuadamente con los comportamientos reactivos sin alterar sus características temporales. También es importante resaltar que todo proceso de planificación requiere una representación del entorno sobre la cual operar. Como ha sido descrito en el apartado 2 del capítulo 4 el sistema dispone de una representación métrica de tipo probabilístico, que será pues la que deban utilizar los procesos de planificación diseñados en la generación de los distintos planes.

En los siguientes apartados se describe en profundidad la implementación de cada uno de estos procesos de planificación necesarios en el sistema, analizando para cada uno de ellos las posibles alternativas existentes y escogiendo la más adecuada para la aplicación desarrollada.

2 Planificación de caminos

La planificación de caminos pretende calcular un camino libre de obstáculos sobre la representación del entorno que posee el sistema para alcanzar el destino final desde la posición actual. En el sistema bajo desarrollo constituye el nivel de planificación básico, incorporando nuevas funcionalidades que resuelven las limitaciones de la navegación reactiva:

- Permite alcanzar todos aquellos destinos finales que se encuentren más allá del entorno inmediato del agente, resolviendo adecuadamente las posibles trampas de mínimos locales.
- Permite aumentar la eficiencia del proceso de navegación al posibilitar la optimización de parámetros globales como distancia recorrida o tiempo requerido durante el proceso de navegación.

El proceso de planificación de caminos mantiene un claro compromiso entre el coste temporal del proceso de búsqueda y el grado de optimización del camino generado, pues la complejidad del problema aumenta considerablemente con el tamaño de la representación del entorno que se posee, imposibilitando la generación de caminos óptimos en un tiempo razonable. Así pues, se han desarrollado numerosas estrategias que intentan abordar el problema de la planificación desde distintos ángulos, ofreciendo cada una de ellas unas características determinadas. Entre las principales estrategias desarrolladas se encuentran [Lat91, HA92]:

- ***Esqueletización*:** tratan de reducir el espacio libre a una red de líneas conectadas entre sí, estableciendo posteriormente el conjunto de líneas que definen el camino a seguir. Existen distintas técnicas para aplicar esta estrategia, entre las cuales se destacan por su importancia:
  - **Grafos de visibilidad:** consiste en determinar el conjunto de líneas que conectan entre sí las distintas características que poseen los obstáculos del entorno, siempre que dichas líneas no atraviesen los propios obstáculos [Nil69] (figura 6.1.a).
  - **Diagramas de Voronoi:** consiste en erosionar el espacio libre del entorno hasta reducirlo a un conjunto de líneas conocido como esqueleto [DY82] (figura 6.1.b).

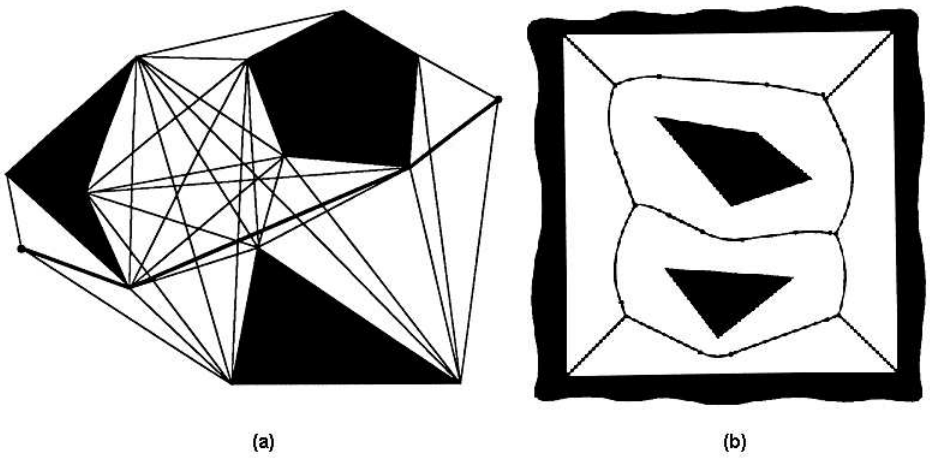

Figura 6.1: Estrategia de esqueletización para planificación de caminos: a) grafos de visibilidad; b) diagramas de Voronoi.

- **Descomposición:** tratan de descomponer el espacio libre del entorno en un conjunto de celdas, buscando un camino desde la celda correspondiente a la posición actual hasta la celda correspondiente al destino final. Existen distintas técnicas para aplicar esta estrategia, entre las cuales se destacan por su importancia:
  - **Exacta:** consiste en dividir el espacio libre en un conjunto de celdas cuya forma está determinada por los propios obstáculos del entorno, por lo que la unión de todas ellas coincide exactamente con el espacio libre del mismo [Lat91] (figura 6.2.a).
  - **Aproximada:** consiste en dividir el espacio libre en un conjunto de celdas cuya forma está predefinida, por lo que la unión de todas ellas está contenida en el espacio libre del entorno. La técnica más utilizada es la de los *quadtrees*, consistente en dividir recursivamente cada celda en 4 menores hasta llegar a un nivel de descomposición adecuado [FB74] (figura 6.2.b).
- **Campos Potenciales:** tratan de definir una función escalar denominada potencial que posee un mínimo en el destino final, un valor muy elevado en los obstáculos del entorno y un valor decreciente hacia el destino final en el espacio libre del entorno [Kha86] (figura 6.3). Siguiendo el gradiente negativo de esta función desde la posición actual es posible alcanzar el destino final evitando los obstáculos del entorno, debido al elevado valor del potencial en los mismos. La función potencial es similar al potencial electrostático generado por cargas atractivas (el destino final) y cargas repulsivas (los obstáculos del entorno), de ahí el nombre de la función escalar.

Todas estas técnicas son directamente aplicables sobre mapas métricos probabilísticos, por lo que cualquiera de ellas puede ser directamente incorporada en el sistema. De entre estas estrategias los Campos Potenciales se destaca como la más simple y eficiente, requiriendo un bajo procesamiento computacional y proporcionando unas muy buenas prestaciones temporales [HA92]. Así pues, será la estrategia escogida como planificador de caminos en el sistema propuesto.

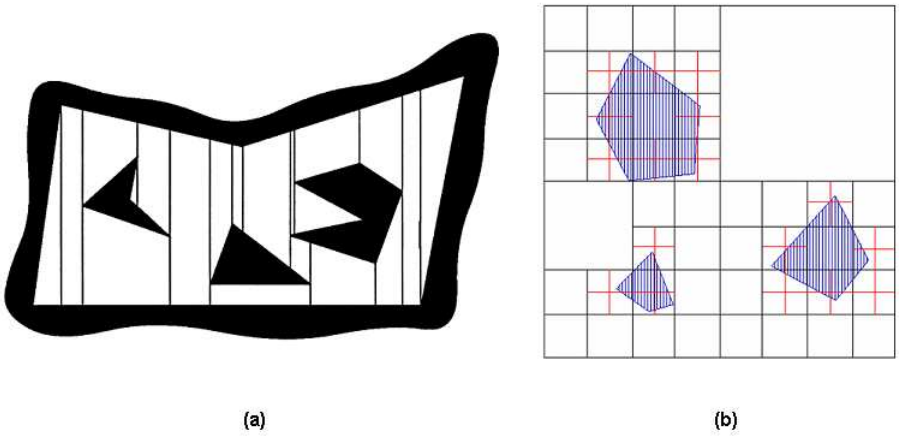

Figura 6.2: Estrategia de descomposición para planificación de caminos: a) exacta; b) aproximada mediante *quadtrees*.

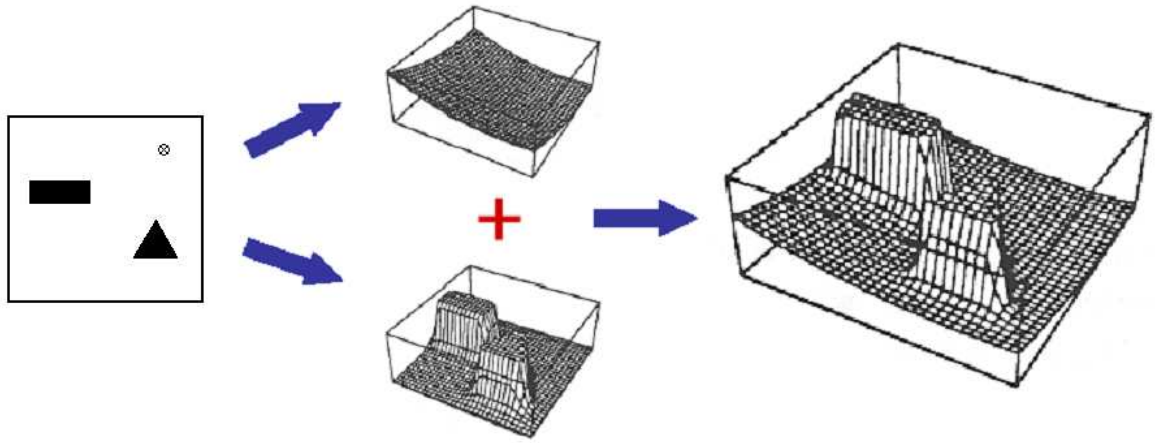

Figura 6.3: Estrategia de los Campos Potenciales para planificación de caminos.

2.1 Cálculo de caminos

La estrategia de los Campos Potenciales pretende generar una función escalar que dirija directamente al agente hacia el destino final. La principal dificultad a la hora de implementar esta estrategia radica en el establecimiento de una función escalar que posea un único mínimo situado en dicho destino final. De hecho, la aplicación directa de la teoría electrostática suele generar una función con numerosos mínimos locales, por lo que el planificador puede quedar atrapado en uno de esos mínimos locales.

Barraquand desarrolló un método de planificación de caminos con Campos Potenciales utilizando funciones de potencial con un único mínimo [BLL92]. El método consiste en una **propagación de ondas** por el espacio libre del entorno partiendo del destino final. Para ello se etiqueta el destino final con un valor determinado, y a continuación se etiquetan con un valor una unidad mayor todas las posiciones adyacentes con vecindad 4. Este proceso se repite hasta que todas las

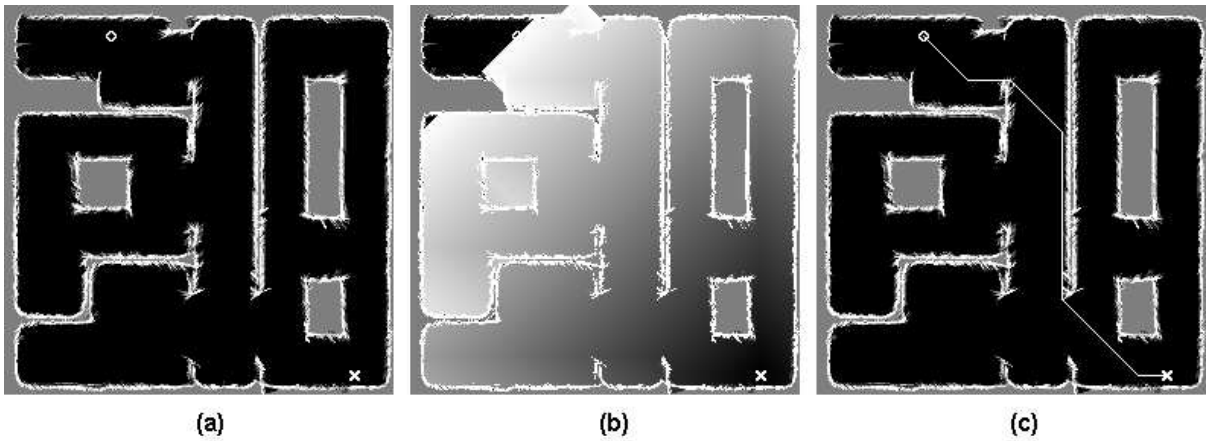

Figura 6.4: Método de planificación de caminos propuesto por Barraquand: a) entorno original; b) propagación de ondas por el espacio libre; c) cálculo del camino.

posiciones de interés estén etiquetadas, por lo que el resultado final se asemeja a una onda cuyo valor irá creciendo a medida que se aleje de la posición inicial. Se conoce como frente de onda al conjunto de posiciones que comparten el mismo valor. La característica que más interesa de este sencillo método radica en que la función resultante de propagar esta onda posee un único mínimo en el destino final, por lo que se puede seguir un gradiente negativo desde la posición actual para alcanzarlo. Si la onda propagada no alcanza la posición actual, no existe ningún camino posible para alcanzar el destino final.

La figura 6.4 muestra la aplicación de este método. La figura 6.4.a representa el entorno sobre el que se desea realizar la planificación de caminos, así como la posición actual del agente (marcada con un círculo) y el destino final (marcado con una "X"). Aunque en la figura 6.4 no se representa por simplicidad, antes de procesar el entorno original se ha realizado una umbralización del mismo para separar los obstáculos del espacio libre, separando así las celdas del entorno que deben ser etiquetadas de las que no. Para ello, todas aquellas celdas del mapa métrico cuya probabilidad de ocupación se encuentre por encima de un valor P_{obst} se consideran obstáculos, y las que se encuentren por debajo de dicho valor se consideran espacio libre. La figura 6.4.b muestra la propagación de una onda desde el destino final hasta la posición actual del agente. A medida que la onda se propaga el valor de su frente de ondas aumenta, por lo que en la imagen la onda adquiere valores más claros, partiendo desde el valor 0 en el destino final representado por el color negro. En la figura 6.4.c se calcula el camino deseado siguiendo el gradiente negativo de la onda propagada a través del entorno, partiendo para ello de la posición actual y alcanzando directamente el destino final al poseer el único mínimo existente.

Este sencillo método posee unas características excepcionales, entre las cuales cabe destacar [BLL92]:

- El camino calculado es cuasi-óptimo al poseer la longitud Manhattan mínima de todos los posibles.
- El camino tiende a estar formado por una serie de líneas rectas, excepto en la vecindad

de los obstáculos donde sigue el contorno de los mismos. Así pues, el camino resultante es bastante sencillo de seguir.

- Al propagar la onda se actualizan únicamente los vecinos de una celda con conectividad 4, por lo que a la hora de seguir el gradiente negativo se favorece el descenso en diagonal, lo cual redunda en que el camino resultante es más suave y tiene cambios de curvatura menos bruscos. Este hecho reduce efectos indeseados de la navegación como el deslizamiento (*slippage*), que introduce errores en el sistema de odometría del agente e induce a errores en la localización del mismo.
- La complejidad del método es lineal con el número de celdas del mapa métrico del entorno, e independiente del número y tamaño de los obstáculos del entorno. Así pues, posee unas excelentes prestaciones temporales.

No obstante, dicho método tal y como fue presentado por Barranquad posee algunos inconvenientes, entre los que cabe destacar:

- El camino calculado bordea los obstáculos del entorno.
- Si existen obstáculos poco definidos en la representación del entorno el camino resultante puede atravesar dichos obstáculos.
- El proceso de planificación trata al agente según un modelo puntual, por lo que no es posible incorporar sus dimensiones en el cálculo del camino.

Evidentemente el inconveniente de bordear los obstáculos genera en el agente un elevado riesgo de colisión con dichos obstáculos. Por su parte, la presencia de obstáculos poco definidos es bastante negativa para el proceso de planificación, pues basta con que exista una anchura de una celda a través de un obstáculo para que el método genere un camino a través suyo. Este efecto se muestra en la figura 6.5. La figura 6.5.a muestra el mismo entorno de la figura 6.4.a, aunque se ha especificado otro destino final que obviamente está fuera del alcance del agente. No obstante, como se aprecia en la figura 6.5.b el proceso de planificación encuentra un camino a través de un obstáculo poco definido. Desafortunadamente, la existencia de obstáculos poco definidos es bastante común en los mapas métricos de tipo probabilístico generados con lecturas sonar, debido a la escasa precisión de dichos sensores.

Ambos inconvenientes fueron resueltos por Barraquand modificando el método de planificación de caminos, al realizar primero una esqueletización del espacio libre mediante una propagación de ondas desde los contornos de los obstáculos, y posteriormente una propagación de onda partiendo del esqueleto recién generado para calcular el nuevo camino [BLL92].

La figura 6.6 muestra la aplicación de este método mejorado. La figura 6.6.a representa el entorno sobre el que se desea realizar la planificación de caminos, así como la posición actual del agente (marcada con un círculo) y el destino final (marcado con una "X"). De nuevo, aunque en la figura 6.6 no se representa se ha realizado una umbralización del entorno para separar los obstáculos

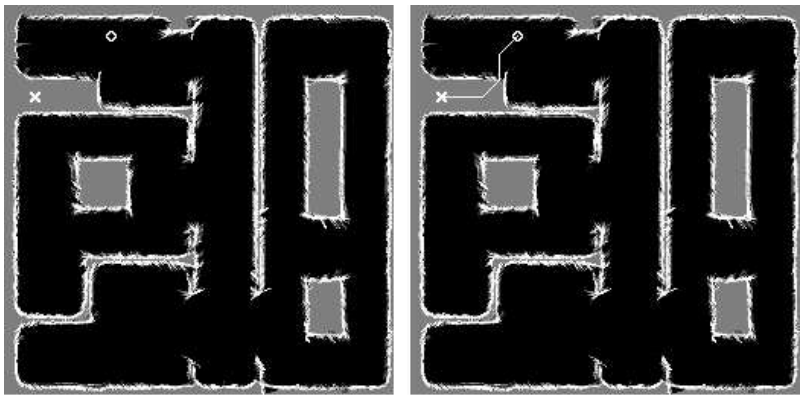

Figura 6.5: Efecto de los obstáculos poco definidos en el método de planificación de caminos propuesto por Barraquand: a) entorno original; b) camino calculado.

del espacio libre. En la figura 6.6.b se ha calculado el esqueleto del espacio libre utilizando una propagación de ondas desde el borde de los obstáculos. En la figura 6.6.c se calcula el esqueleto aumentado mediante una propagación de ondas que una el destino final con el propio esqueleto. En la figura 6.6.d se realiza una propagación de ondas a lo largo del esqueleto aumentado. La figura 6.6.e muestra una propagación de ondas por el espacio libre desde el propio esqueleto aumentado. Finalmente, en la figura 6.6.f se calcula el camino deseado siguiendo el gradiente negativo de la onda propagada a través del espacio libre, partiendo para ello de la posición actual y alcanzando directamente el destino final a través del esqueleto aumentado al poseer el único mínimo existente.

Mediante este nuevo método de propagación de ondas mejorado se resuelven los problemas de riesgo de colisión y obstáculos poco definidos del método original, ya que se consigue una característica importante:

- Al utilizar el esqueleto el camino resultante está lo más separado posible de los obstáculos del entorno, por lo que se obtiene un camino de máxima seguridad y que no atraviesa los obstáculos poco definidos del entorno.

Sin embargo, este nuevo método también posee una serie de inconvenientes, entre los que se pueden destacar los siguientes:

- El camino suele ser bastante más largo, pues siempre recorre el espacio libre lo más alejado posible de los obstáculos del entorno.
- El camino resultante es bastante ruidoso y con cambios de curvatura bastante bruscos, por lo que es más difícil de seguir y empeora el problema del deslizamiento.
- La complejidad del método mejorado es mayor que la del método original, por lo que las prestaciones temporales del nuevo método son peores.

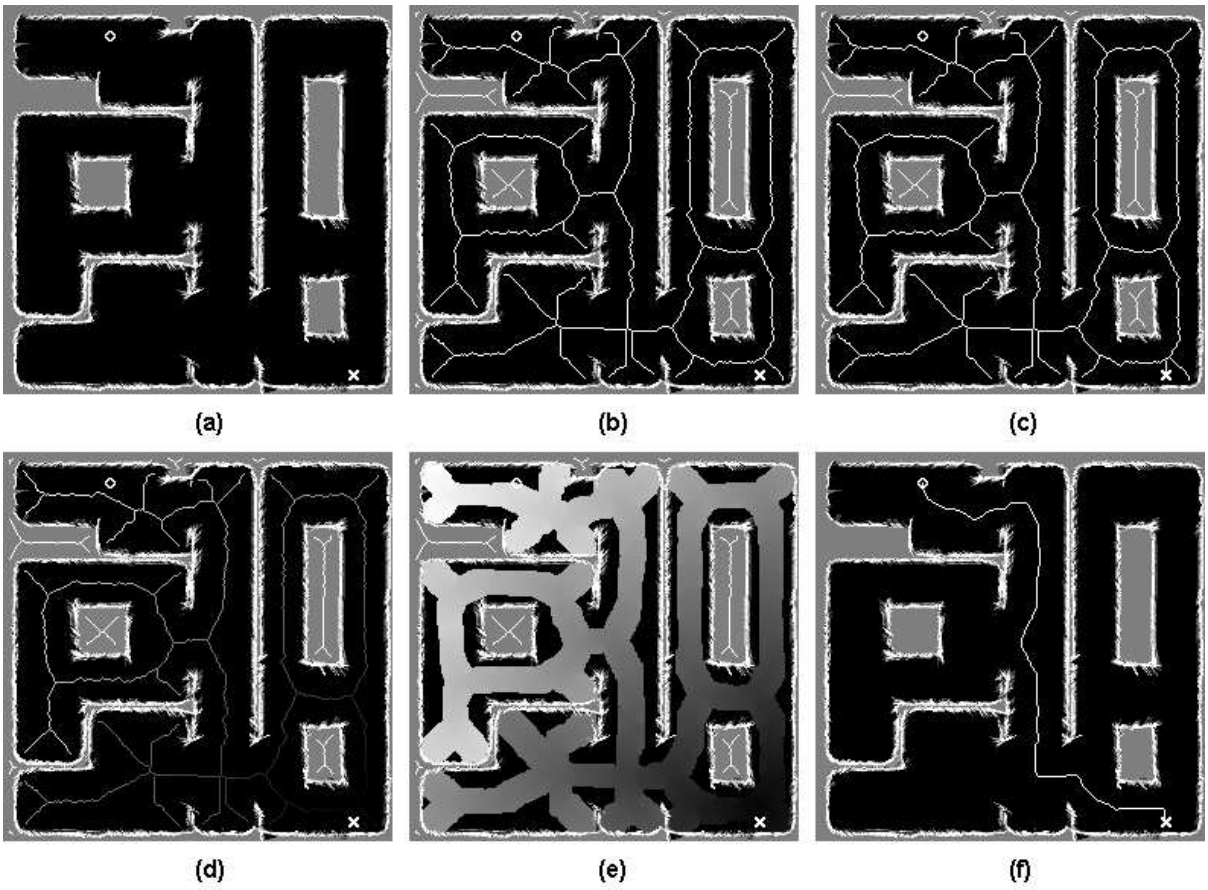

Figura 6.6: Método de planificación de caminos mejorado propuesto por Barraquand: a) entorno original; b) cálculo del esqueleto; c) cálculo del esqueleto aumentado; d) propagación de ondas por el esqueleto aumentado; e) propagación de ondas por el espacio libre; f) cálculo del camino.

- Al igual que en el método original el proceso de planificación trata al agente según un modelo puntual, por lo que tampoco es posible incorporar sus dimensiones en el cálculo del camino.

A la vista de estos resultados es evidente que el nuevo método propuesto por Barraquand conlleva un deterioro significativo de las prestaciones obtenidas por el método original. Ante esta situación se ha optado por seguir manteniendo el método original e intentar resolver los problemas que manifestaba usando alguna otra técnica, pues la pérdida de prestaciones al usar el método mejorado se han considerado demasiado importantes. Para resolver los problemas de riesgo de colisión y de obstáculos poco definidos, así como para incorporar el tamaño del agente en el proceso de planificación se ha optado por preprocesar el mapa métrico del entorno aplicando un **crecimiento de obstáculos** de *Obst_Grow* celdas, operación realizada mediante una propagación de ondas desde los contornos de los obstáculos del mismo. Esta técnica permite aumentar artificialmente el tamaño de cada uno de los obstáculos del entorno por una cantidad determinada, tras lo cual se puede aplicar el método originalmente propuesto por Barranquad al nuevo mapa métrico probabilístico resultante.

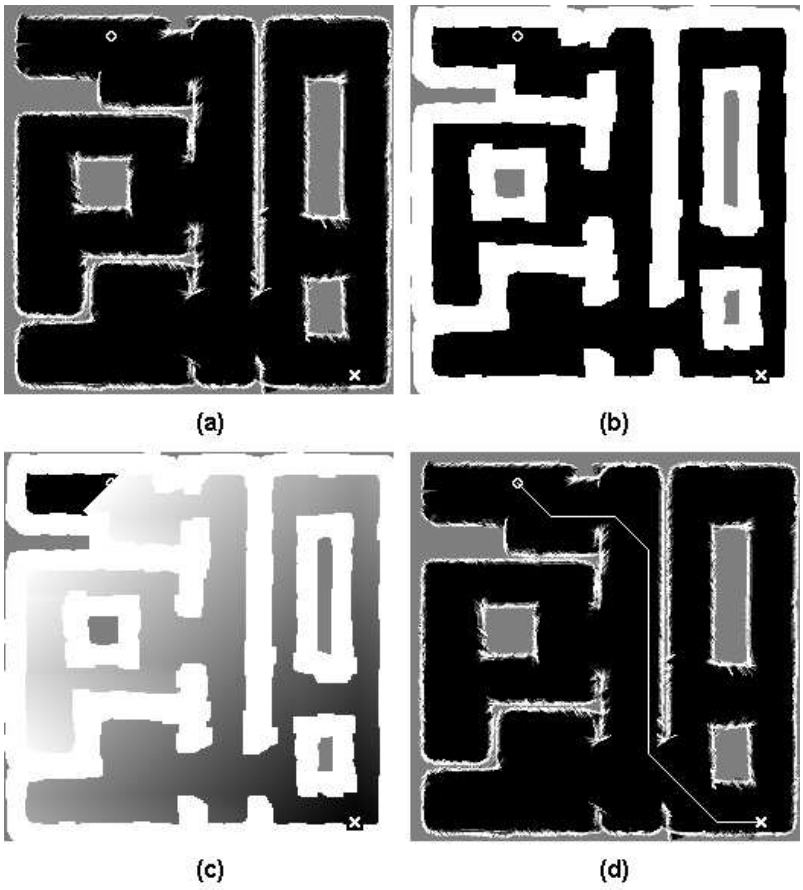

Figura 6.7: Método de planificación de caminos propuesto: a) entorno original; b) crecimiento de obstáculos; c) propagación de ondas por el espacio libre; d) cálculo del camino.

La figura 6.7 muestra la aplicación del método propuesto. La figura 6.7.a representa el entorno sobre el que se desea realizar la planificación de caminos, así como la posición actual del agente (marcada con un círculo) y el destino final (marcado con una "X"). De nuevo, aunque en la figura 6.7 no se representa se ha realizado una umbralización del entorno para separar los obstáculos del espacio libre. En la figura 6.7.b se ha realizado un crecimiento de obstáculos de 5 celdas mediante una propagación de ondas desde el borde de los obstáculos del entorno. En la figura 6.7.c se ha propagado una onda desde el destino final a lo largo de la representación del entorno con el crecimiento de obstáculos. Finalmente, en la figura 6.7.d se calcula el camino deseado siguiendo el gradiente negativo de la onda propagada a través del entorno, partiendo para ello de la posición actual y alcanzando directamente el destino final al poseer el único mínimo existente.

El método propuesto resuelve adecuadamente el riesgo de colisión e incorpora el tamaño del agente en el proceso de planificación, mostrando importantes características:

- El camino calculado mantiene todas las características del método originalmente propuesto por Barraquand: longitud cuasi-óptima, sencillo de seguir, suave y con excelentes prestaciones temporales.

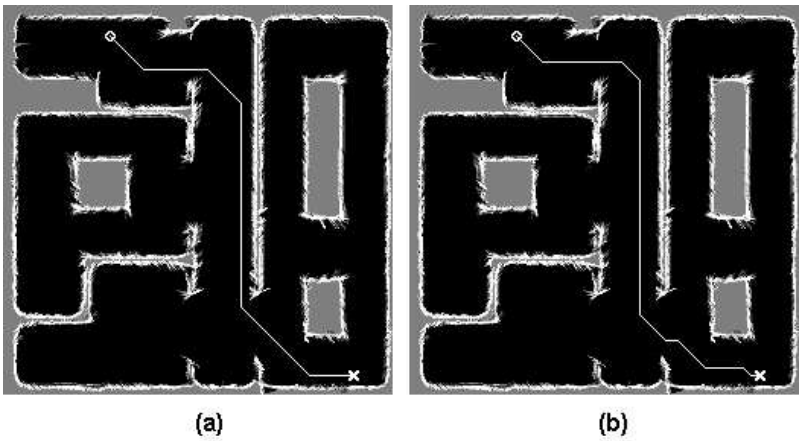

Figura 6.8: Influencia del crecimiento de obstáculos en el método de planificación de caminos propuesto: a) con un crecimiento de obstáculos de 5 celdas; b) con un crecimiento de obstáculos de 10 celdas.

- El crecimiento de obstáculos realizado en el método propuesto controla directamente la distancia de seguridad entre el camino generado y los obstáculos del entorno, evitando bordearlos. Esta distancia de seguridad puede ser muy fácilmente reajustada aumentando o disminuyendo el crecimiento de obstáculos llevado a cabo sobre dichos obstáculos.
- El crecimiento de obstáculos aumenta la definición de los obstáculos del entorno, inhabilitando la generación de caminos erróneos a través de obstáculos poco definidos.
- El tamaño del agente se incorpora en el proceso de planificación mediante el crecimiento de obstáculos, permitiendo ajustar muy fácilmente el método propuesto a distintos agentes según la distancia de seguridad que requiera cada uno en función de su tamaño.
- La operación de crecimiento de obstáculos requiere un menor procesamiento que la operación de esqueletización, por lo que el método propuesto posee mejores prestaciones temporales que el método mejorado de Barraquand.

A efectos ilustrativos, la figura 6.8 muestra la influencia del crecimiento de obstáculos en el método propuesto. A medida que el crecimiento de obstáculos aumenta, la distancia de seguridad entre el camino generado y los obstáculos del entorno se incrementa, por lo que el camino resultante se va alejando en mayor medida de los obstáculos. Este efecto se puede apreciar en las figuras 6.8.a y 6.8.b, donde se ha empleado un crecimiento de obstáculos de 5 celdas y 10 celdas respectivamente.

Debido a las excelentes características exhibidas por el nuevo método propuesto será el utilizado como planificador de caminos en el sistema desarrollado. Este proceso de planificación garantiza que se alcanza el destino final evitando las posibles trampas de mínimos locales del entorno, a la vez que optimiza parámetros globales del camino generado como distancia recorrida, cambios de dirección suaves y seguridad al pasar lejos de los obstáculos circundantes.

Para concluir con este apartado se van a analizar las prestaciones temporales del proceso de

Pentium 4 - 2.4 GHz - 512 MB RAM				
	128x128 celdas	256x256 celdas	512x512 celdas	1024x1024 celdas
Umbralización	73.60 ± 1.50 μs	296.2 ± 0.4 μs	1.6003 ± 0.0022 ms	6.837 ± 0.009 ms
Crecimiento	342.79 ± 0.22 μs	1.1154 ± 0.0011 ms	5.63 ± 0.03 ms	17.19 ± 0.04 ms
Propagación	97.46 ± 0.07 μs	620 ± 30 μs	4.142 ± 0.011 ms	20.90 ± 0.021 ms
Camino	5.172 ± 0.023 μs	10.46 ± 0.18 μs	65.46 ± 0.14 μs	224.5 ± 1.8 μs
Total	519.02 ± 1.60 μs	2.04 ± 0.03 ms	11.44 ± 0.03 ms	45.15 ± 0.06 ms
Pentium 3 - 1.0 GHz - 256 MB RAM				
	128x128 celdas	256x256 celdas	512x512 celdas	1024x1024 celdas
Umbralización	338.1 ± 2.3 μs	2.247 ± 0.012 ms	15.02 ± 0.03 ms	62.67 ± 0.22 ms
Crecimiento	1.570 ± 0.014 ms	6.32 ± 0.04 ms	26.4 ± 0.08 ms	88.8 ± 0.3 ms
Propagación	404.8 ± 0.7 μs	2.832 ± 0.016 ms	15.63 ± 0.04 ms	72.61 ± 0.15 ms
Camino	22.73 ± 0.11 μs	56.0 ± 0.3 μs	230.9 ± 0.6 μs	629 ± 7 μs
Total	2.336 ± 0.014 ms	11.46 ± 0.04 ms	57.28 ± 0.10 ms	224.7 ± 0.4 ms
Pentium 3 - 500 MHz - 192 MB RAM				
	128x128 celdas	256x256 celdas	512x512 celdas	1024x1024 celdas
Umbralización	350.2 ± 1.0 μs	3.171 ± 0.018 ms	15.78 ± 0.06 ms	70.4 ± 0.5 ms
Crecimiento	2.096 ± 0.010 ms	6.62 ± 0.03 ms	30.01 ± 0.3 ms	99.8 ± 0.8 ms
Propagación	466.9 ± 1.2 μs	2.995 ± 0.019 ms	24.5 ± 0.4 ms	126.7 ± 0.6 ms
Camino	26.52 ± 0.11 μs	69.4 ± 1.6 μs	266 ± 11 μs	716 ± 16 μs
Total	2.940 ± 0.010 ms	12.86 ± 0.04 ms	70.6 ± 0.5 ms	297.6 ± 1.6 ms

Tabla 6.1: Prestaciones temporales del proceso de planificación de caminos propuesto.

planificación de caminos propuesto. Obviamente estas prestaciones dependen claramente del tamaño del entorno y de los recursos de la máquina empleada para implementar el planificador, por lo que se ha repetido la misma prueba para distintos tamaños de entorno y para distintos tipos de máquinas: 128x128, 256x256, 512x512 y 1024x1024 celdas, utilizando en cada caso una máquina de altas prestaciones (Pentium 4 - 2.4 GHz - 512 MB RAM), una máquina de prestaciones medias (Pentium 3 - 1.0 GHz - 256 MB RAM) y una máquina de bajas prestaciones (Pentium 3 - 500 MHz - 192 MB RAM). Evidentemente el tiempo de procesamiento depende del número de celdas empleadas en el crecimiento de obstáculos, utilizando en todos los casos 5 celdas. El tiempo medio de procesamiento empleado en cada una de las etapas del método propuesto se muestra en la tabla 6.1, donde los errores se han calculado sobre un intervalo de confianza del 95%.

Según la tabla 6.1 el proceso de planificación de caminos es muy rápido, debido sin lugar a dudas a la sencillez del método propuesto. Incluso en condiciones extremas con un mapa métrico de tamaño 1024x1024 celdas y utilizando una máquina de bajas prestaciones el retardo medio del proceso de planificación está alrededor de los 300 ms, lo que implica que a una velocidad del agente de unos 300 mm/s se recorren aproximadamente 9 cm antes de calcular el camino. En el sistema desarrollado el mapa métrico utilizado posee un tamaño de 256x256 celdas, por lo que incluso utilizando una máquina de bajas prestaciones el retardo medio está alrededor de los 13 ms, es decir, un desplazamiento del agente de unos 4 mm a una velocidad de 300 mm/s.

Estos resultados ponen de manifiesto que el proceso de planificación de caminos es capaz de operar prácticamente en tiempo real, generando inmediatamente los planes de navegación adecuados a partir de los nuevos comandos.

2.2 Seguimiento de caminos

Una vez obtenido el camino a seguir para alcanzar el destino final es necesario realizar el seguimiento del mismo por parte de la Capa de Navegación, problema conocido como *tracking*. El camino propuesto no tiene por qué ser estrictamente seguido, pues pueden aparecer en el entorno obstáculos inesperados que lo hagan inviable. De hecho, en lugar de un seguimiento exhaustivo que obligue a recalcular un nuevo camino cada vez que aparece un nuevo obstáculo en el entorno es más eficiente realizar un seguimiento aproximado siempre y cuando se permita alcanzar de forma segura el destino final. Es decir, hay que tratar el camino generado por el planificador de caminos como una guía para que el agente alcance el destino final, no como una imposición que deba ser seguida estrictamente.

El problema del seguimiento de caminos se puede abordar siguiendo distintas estrategias, dependiendo de las propias características del camino a seguir. Una de ellas consiste en dirigirse sucesivamente mediante un esquema reactivo a los distintos puntos que forman el camino propuesto [PUB+99], aunque una estrategia de este tipo requiere un procesamiento excesivo al modificar continuamente el siguiente destino al que dirigirse y no es adecuada en presencia de caminos ruidosos. Una estrategia más eficiente consiste en la descomposición y aproximación del camino completo mediante la unión de un conjunto de puntos característicos. Estos puntos se pueden extraer calculando los puntos de inflexión del camino mediante el uso de una función de curvatura [PPB+00], de forma que el camino original pueda ser seguido de forma aproximada tras dirigirse sucesivamente mediante un esquema reactivo a dichos puntos de inflexión.

Debido a sus mejores prestaciones la estrategia de descomposición en puntos de inflexión será la empleada para realizar el seguimiento del camino. No obstante, puesto que el planificador propuesto proporciona caminos suaves compuestos en su mayor parte por líneas rectas, la extracción de los puntos de inflexión mediante el uso de la función de curvatura puede ser sustituida por un simple análisis del **código cadena** del camino, operación mucho más sencilla y rápida de realizar.

La figura 6.9 muestra el método propuesto de descomposición del camino en puntos de inflexión mediante el análisis del código cadena. En el camino final resultante predominan las líneas rectas, por lo que es muy fácil extraer directamente a partir del mismo un conjunto reducido de puntos característicos que describan el camino completo mediante su unión directa. Obviamente la sencillez de esta descomposición depende de la complejidad del entorno sobre el cual desarrolla su actividad el agente, pero aun en entornos muy complejos con caminos más ruidosos el método de descomposición es muy sencillo de aplicar.

Por último, se van a analizar las prestaciones temporales del proceso de seguimiento de caminos propuesto. Obviamente estas prestaciones dependen claramente de la longitud y complejidad del

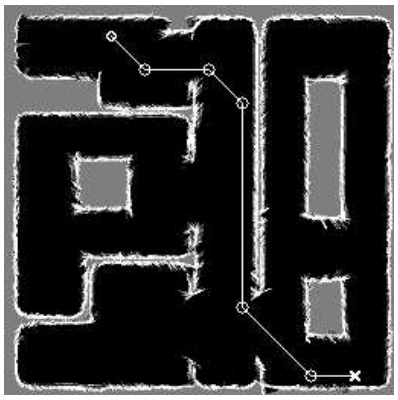

Figura 6.9: Método de descomposición del camino propuesto mediante el análisis del código cadena.

Pentium 4 - 2.4 GHz - 512 MB RAM				
	128x128 celdas	256x256 celdas	512x512 celdas	1024x1024 celdas
Total	4.35 ± 0.03 μs	5.53 ± 0.04 μs	7.41 ± 0.03 μs	10.17 ± 0.04 μs
Pentium 3 - 1.0 GHz - 256 MB RAM				
	128x128 celdas	256x256 celdas	512x512 celdas	1024x1024 celdas
Total	17.97 ± 0.19 μs	22.70 ± 0.14 μs	27.92 ± 0.11 μs	42.7 ± 0.4 μs
Pentium 3 - 500 MHz - 192 MB RAM				
	128x128 celdas	256x256 celdas	512x512 celdas	1024x1024 celdas
Total	20.03 ± 0.12 μs	24.10 ± 0.20 μs	34.92 ± 0.13 μs	49.5 ± 0.5 μs

Tabla 6.2: Prestaciones temporales del proceso de seguimiento de caminos propuesto.

camino calculado y de los recursos de la máquina empleada para implementar el planificador, por lo que se ha repetido la misma prueba para distintos tamaños de entorno y para distintos tipos de máquinas: 128x128, 256x256, 512x512 y 1024x1024 celdas, utilizando en cada caso una máquina de altas prestaciones (Pentium 4 - 2.4 GHz - 512 MB RAM), una máquina de prestaciones medias (Pentium 3 - 1.0 GHz - 256 MB RAM) y una máquina de bajas prestaciones (Pentium 3 - 500 MHz - 192 MB RAM). La relación entre el tamaño del entorno y la longitud del camino calculado no es en absoluto directa, pero evidentemente dicho camino suele poseer mayor longitud cuanto mayor sea el tamaño del entorno. El tiempo medio de procesamiento empleado se muestra en la tabla 6.2, donde los errores se han calculado sobre un intervalo de confianza del 95%.

Como indica la tabla 6.2 el proceso de descomposición del camino es extremadamente rápido, debido sin lugar a dudas a la suavidad del camino propuesto. El retardo medio de la descomposición del camino no supera en ningún caso los 50 μs, pudiendo considerarse pues prácticamente despreciable.

2.3 Integración en el sistema de la planificación de caminos

La adecuada integración de los procesos de planificación con los comportamientos reactivos constituye un factor crítico en todo sistema híbrido. En este apartado se va a realizar un análisis detallado de la progresión temporal del proceso de planificación de caminos y de su interacción con el sistema reactivo.

Los puntos característicos en los que ha sido descompuesto el camino generado formarán los destinos intermedios a los que se deberá dirigir sucesivamente el agente para seguir aproximadamente dicho camino. Así pues, una vez generado y descompuesto el camino en una serie de destinos intermedios la Capa de Planificación los enviará sucesivamente a la Capa de Navegación, la cual desarrollará la función de evitación de obstáculos mientras se sigue el camino propuesto. El análisis y descomposición del camino propuesto por parte de la Capa de Planificación libera de esta función a la Capa de Navegación, por lo que esta última puede seguir operando sin modificar sus prestaciones temporales. Además, con este esquema la Capa de Planificación únicamente debe pasar a la Capa de Navegación el siguiente destino intermedio a visitar en lugar del camino completo a recorrer, reduciendo así el tamaño de los datos a compartir y disminuyendo por lo tanto el tiempo empleado en intercambiar dichos datos mediante el servidor central de la arquitectura *DLA*.

Esta forma de operación garantiza una adecuada integración entre los procesos de planificación de la Capa de Planificación y los comportamientos reactivos de la Capa de Navegación, ya que al operar ambas capas independientemente el tiempo de procesamiento de cada una de ellas no influye en el tiempo de procesamiento de la otra. El proceso de navegación del sistema combinando la operación de ambas capas se realiza según el siguiente esquema:

1. El usuario introduce un destino final para ser alcanzado, el cual es inmediatamente enviado tanto a la Capa de Navegación como a la Capa de Planificación del sistema.

2. El planificador de caminos calcula un camino libre de obstáculos sobre el mapa métrico completo disponible para alcanzar el destino final, es decir, sobre todo el entorno conocido. Tras calcular dicho camino, se descompone en una serie de destinos intermedios que hay que seguir sucesivamente para completarlo, enviando el siguiente a la Capa de Navegación.

3. Inicialmente la Capa de Navegación se dirige directamente hacia el destino final. Una vez que el planificador de caminos ha finalizado su procesamiento y dispone de un camino descompuesto, la Capa de Navegación comienza a dirigirse sucesivamente al siguiente destino intermedio con el fin de completar el camino calculado.

4. Si se detecta un cambio significativo en el mapa métrico durante el proceso de navegación, el planificador de caminos es relanzado de nuevo para calcular un nuevo camino más acorde con las nuevas circunstancias del entorno. Mientras tanto, la Capa de Navegación sigue navegando según el camino anteriormente propuesto.

Del análisis detallado de este modo de operación se pueden extraer algunas conclusiones importantes:

- Al enviar directamente el destino final tanto a la Capa de Navegación como a la Capa de Planificación se permite que el agente comience a navegar inmediatamente sin necesidad de esperar a que el proceso de planificación finalice. El principal inconveniente de este esquema radica en el hecho de que la navegación reactiva puede dirigir al agente en una dirección no adecuada inicialmente. No obstante, puesto que el proceso de planificación es bastante rápido no resulta tan costoso deshacer el camino recorrido en el caso de optar por una dirección errónea, frente a la ventaja que supone tener al agente en movimiento si se ha optado por una dirección correcta.

- Como se ha descrito en el apartado 4.3 del capítulo 3, el camino calculado se mantiene hasta que se detectan cambios significativos en el entorno. Para ello, se computan el número de celdas del mapa métrico que pasan de libres a ocupadas o viceversa [1], y se considera que el entorno ha variado sustancialmente cuando este número supera un cierto umbral U_{path}. Con este esquema se optimiza la operación del sistema, pues únicamente se realiza la planificación cuando se considera necesaria, manteniendo el camino calculado hasta que es lo suficientemente obsoleto por los cambios del entorno.

- Cuando los cambios en el entorno requieren que se recalcule un nuevo camino la Capa de Navegación sigue navegando según el anteriormente calculado, presentando la ventaja de que el agente no necesita pararse mientras finaliza el nuevo proceso de planificación. El principal inconveniente, de nuevo, radica en el hecho de que puede dirigirse al agente en una dirección equivocada. Sin embargo, se ha preferido mantener este esquema debido a que el proceso de planificación es muy rápido, por lo que no es tan costoso deshacer la distancia recorrida en el caso de optar por una dirección errónea frente a la ventaja de no detener al agente si la dirección es acertada.

Gracias a esta integración el sistema es capaz de seguir operando en tiempo real, pues la Capa de Navegación sigue respondiendo rápidamente a cualquier estímulo que se presente en el entorno. Tras finalizar el proceso de planificación la Capa de Navegación es adecuadamente dirigida según los nuevos planes generados, manteniendo sus características de operación en tiempo real.

3 Planificación de rutas

La planificación de rutas pretende calcular una ruta con las regiones explorados y no exploradas a atravesar para alcanzar el destino final. En el sistema bajo desarrollo constituye el nivel de planificación más avanzado, incorporando nuevas funcionalidades sobre el nivel básico de

[1] Recuérdese que tal y como se ha descrito en el apartado 2.1 todas aquellas celdas del mapa métrico cuya probabilidad de ocupación se encuentre por encima de un valor P_{obst} se consideran obstáculos, y las que se encuentren por debajo de dicho valor se consideran espacio libre.

planificación de caminos que aumentan la flexibilidad en el proceso de navegación del sistema. De hecho, la planificación de caminos descrita en el apartado 2 posee las siguientes limitaciones:

- La complejidad del planificador de caminos es proporcional al tamaño del mapa métrico, por lo que las prestaciones temporales del planificador se degradan considerablemente para entornos grandes.
- Sólo se pueden planificar caminos hacia un único destino final, por lo que no es posible elaborar planes más complejos donde sea necesario visitar varios destinos consecutivamente.
- Presenta un interfaz poco útil al usuario, pues este debe especificar destinos finales mediante posiciones absolutas dentro del entorno.

La capacidad de navegación del sistema puede ser mejorada incorporando procesos de planificación que operen con un nivel de abstracción mayor de la realidad, permitiendo realizar una interpretación del mundo de más alto nivel y desarrollando actividades más complejas y eficientes [Kui00, JZ00]. Para ello es necesario utilizar otro tipo de representación del entorno más compacta y abstracta en el proceso de planificación, aspecto que será analizado en el siguiente apartado.

3.1 Representación topológica del entorno

Los dos tipos de representaciones más utilizados a la hora de modelar el entorno donde el agente desarrolla su actividad son [Thr98]:

- **Representación métrica:** se basa en realizar una representación geométrica del entorno [Mor88]. Este tipo de representación es muy sencilla de generar y gestionar, pero presenta una elevada cantidad de datos para grandes entornos, lo que impone un tiempo de procesamiento bastante elevado de los procesos de planificación.
- **Representación topológica:** se basa en realizar una representación del entorno mediante un conjunto de nodos enlazados que representan regiones conectadas entre sí [KB91b]. Este tipo de representación suele ser más compleja de generar y gestionar, pero produce representaciones más compactas al depender su resolución de la complejidad del entorno, presentando una cantidad de datos bastante menor y siendo más rápida de procesar.

Ambos paradigmas son complementarios. La representación métrica es generada por el sistema tal y como se ha descrito en el apartado 2.1 del capítulo 4, y es la utilizada por el planificador de caminos descrito en el apartado 2. La representación topológica realiza una descripción más abstracta del entorno al dividirlo en regiones conectadas entre sí, lo que permite desarrollar comportamientos de navegación más complejos.

Desde el punto de vista funcional, un esquema que combine adecuadamente las representaciones métrica y topológica puede aumentar la capacidad de navegación del sistema:

- Permite generar una representación más compacta del entorno, por lo que se puede realizar un procesamiento más profundo con excelentes prestaciones temporales para grandes entornos.
- Realiza una representación simbólica muy apropiada para realizar un razonamiento abstracto de alto nivel al describir el entorno como un conjunto de regiones conectadas entre sí, posibilitando la planificación de rutas completas para visitar varias regiones minimizando la distancia recorrida o la exploración completa de entornos en el menor tiempo posible.
- Presenta un interfaz más natural al usuario, el cual puede especificar destinos finales como regiones y no como posiciones absolutas dentro del entorno.

Es importante observar que en el sistema propuesto la representación topológica complementa a la representación métrica, pero no la sustituye. El proceso de planificación asociado a la representación topológica permite enumerar las distintas regiones que es necesario atravesar para alcanzar el destino final, pero no determina el camino completo a seguir. Dicho camino debe ser generado por el proceso de planificación de caminos asociado a la representación métrica.

No obstante, y a pesar de la usual utilización que se realiza de las representaciones topológicas, no existe actualmente consenso alguno sobre la información básica que deben contener dichas representaciones, ni por lo tanto sobre el procedimiento más adecuado para generarlas. Esta situación ha originado la existencia de numerosas técnicas [RK04, BS04, Sav05], cada una de ellas específicamente relacionada con la estrategia de exploración seguida por el agente [YB96, CK97, SD92, GS99, Kui00, Tom01, BN01, DMS02, BNL+03]. Es por ello por lo que resulta complejo determinar cual es la más apropiada y adaptarla a las características particulares de la aplicación que se pretende desarrollar.

A la hora de trabajar con representaciones topológicas una de los mayores dificultades que es necesario resolver se encuentra en el conocido problema de la **ambigüedad**, que consiste en la dificultad de discernir distintas regiones entre sí a partir únicamente de la información sensorial captada del entorno. Para resolver este problema se suele incorporar algún tipo de información métrica en la representación topológica, de forma que se facilite la identificación de cada región utilizando esta información. A la hora de incorporar esta información métrica existen numerosas aproximaciones, si bien la gran mayoría se suelen clasificar en uno de estos dos grupos [SK97]:

- **Estrategia directa:** consiste en construir una representación topológica a partir de la información sensorial captada del entorno, añadiendo cierta información métrica para solucionar el problema de la ambigüedad [KB91b, Mat92, CN01].
- **Estrategia indirecta:** consiste en construir una representación topológica a partir de una representación métrica previamente generada, la cual posee ya la información métrica necesaria para evitar el problema de la ambigüedad [Thr98, AdRMF91, FS00].

Como en el sistema ya se dispone de una representación métrica, la estrategia indirecta resulta más conveniente, por lo que será la opción escogida en el sistema a desarrollar. De entre las

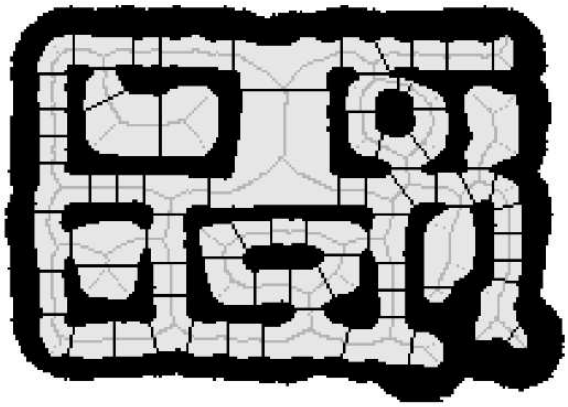

Figura 6.10: Representación topológica mediante la técnica propuesta por Thrun.

distintas soluciones existentes de este tipo, algunas de las más destacadas son las que se describen a continuación:

- **Thrun:** [Thr98]

  La técnica propuesta por Thrun realiza una esqueletización mediante diagramas de Voronoi de la representación métrica para obtener las distintas regiones del entorno y generar la representación topológica (figura 6.10). Aunque este método puede realizar una descomposición bastante estable del entorno en regiones irregulares, sólo es capaz de operar tras haber generado por completo el mapa métrico del entorno. Además, únicamente representa las regiones libres, por lo que no se pueden incorporar las regiones no exploradas en los posteriores procesos de planificación que utilicen esta representación topológica.

- **Arleo:** [AdRMF91]

  La técnica propuesta por Arleo realiza una descomposición aproximada mediante regiones rectangulares de la representación métrica, obteniendo así las distintas regiones del entorno y generando la correspondiente representación topológica (figura 6.11). Esta técnica utiliza un proceso previo de exploración que modela los obstáculos del entorno como líneas rectas, agrupando e identificando posteriormente los distintos obstáculos tras detectar las intersecciones entre dichas líneas rectas. Aunque el método es capaz de operar en tiempo real, plantea el inconveniente de no ser muy apropiado para entornos donde no predominan las regiones rectangulares, además de que únicamente representa las regiones libres del entorno.

- **Fabrizi:** [FS00]

  La técnica propuesta por Fabrizi realiza una división mediante lógica difusa de la representación métrica, obteniendo así las distintas regiones del entorno y generando la correspondiente representación topológica (figura 6.12). El procesamiento realizado mediante técnicas de lógica difusa permite identificar las distintas regiones del entorno en función de su tamaño, lo cual puede resultar de interés en los posteriores procesos de planificación.

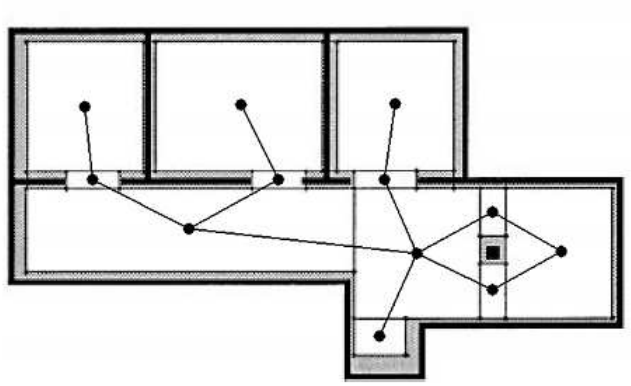

Figura 6.11: Representación topológica mediante la técnica propuesta por Arleo.

Figura 6.12: Representación topológica mediante la técnica propuesta por Fabrizi.

Este método es capaz de operar en tiempo real con regiones irregulares, aunque únicamente representa las regiones libres del entorno.

Desde el punto de vista del sistema a implementar todas estas técnicas plantean el inconveniente de representar únicamente las regiones libres del entorno, no siendo adecuadas para desarrollar los niveles superiores de la Jerarquía de Navegación. En la presente Tesis se propone una nueva técnica para la generación de la representación topológica del entorno, siendo capaz de operar con todo tipo de regiones de forma irregular y mostrando unas prestaciones temporales muy destacables. Esta técnica es la que se describe en el siguiente apartado.

3.2 Generación del mapa topológico

La técnica utilizada para la generación de la representación topológica del entorno se basa en el trabajo previamente desarrollado por Urdiales [Urd99], donde se genera una **estructura jerárquica piramidal** a partir del mapa métrico del entorno para descomponerlo en una serie de regiones conectadas entre sí. El método propuesto en este trabajo produce unos resultados muy destacables, aunque plantea algunos inconvenientes:

- La representación generada no siempre garantiza la correcta conectividad entre las regiones del entorno, pues bajo determinadas circunstancias pueden conectarse regiones que no están en contacto.

- El número de regiones existentes en el entorno está predeterminado en función del nivel de trabajo escogido de la estructura jerárquica piramidal, por lo que la división realizada del entorno no siempre se adecúa a la topología existente en el mismo.

- El método de selección de las regiones del entorno a partir de la estructura jerárquica piramidal no siempre conduce a una descomposición adecuada del entorno, siendo posible la pérdida de determinadas regiones si el nivel de trabajo seleccionado no es el apropiado.

- El coste temporal del proceso de generación de la representación topológica es bastante elevado, pues se basa en un proceso recursivo de estabilización de todos los niveles de la estructura.

Con el fin de solventar estos inconvenientes y mejorar la descomposición realizada del entorno en regiones se ha modificado el método original propuesto por Urdiales. Para ello se ha variado la estructura jerárquica piramidal generada mediante la incorporación de nuevos parámetros que permiten una utilización más eficiente de la misma, y se ha perfeccionado el método de selección de las distintas regiones del entorno. El nuevo proceso de generación propuesto de la estructura jerárquica piramidal es el siguiente [UBP+03]:

1. **Umbralización:** en esta fase todas aquellas celdas del mapa métrico cuya probabilidad de ocupación se encuentre por encima de un umbral U_{occ} son consideradas como espacio ocupado, asignándoles la probabilidad de ocupación C_{occ}. Aquellas celdas cuya probabilidad de ocupación se encuentre por debajo de un umbral U_{free} son consideradas como espacio libre, asignándoles la probabilidad de ocupación C_{free}. El resto de celdas cuya probabilidad de ocupación se encuentre entre los umbrales U_{occ} y U_{free} son consideradas como espacio no explorado, asignándoles la probabilidad de ocupación C_{not}.

2. **Crecimiento:** en esta fase se realiza un crecimiento de obstáculos de $Obst\_Grow$ celdas para reducir el efecto de los obstáculos poco definidos, que podrían ocasionar la conexión errónea de distintas regiones. Para ello se sigue el procedimiento descrito en el apartado 2.1.

3. **Generación:** en esta fase se genera la estructura jerárquica inicial a partir del mapa métrico umbralizado, el cual forma la base de la estructura piramidal correspondiente al nivel 0. Cada nivel l de esta pirámide es un mapa métrico formado al promediar 2x2 nodos del mapa métrico del nivel inferior $l-1$, por lo que su tamaño es la cuarta parte del tamaño del nivel inferior. El nodo promedio del nivel l se conoce como nodo padre, mientras que los nodos promediados del nivel $l-1$ se conocen como nodos hijos. Cada nodo (x, y) de la estructura piramidal en el nivel l se identifica por la terna (x, y, l), y tiene los siguientes parámetros asociados:

- Homogeneidad $H(x,y,l)$: es igual a 1 si sus cuatro nodos hijos tienen la misma probabilidad de ocupación y son homogéneos, y es igual a 0 en cualquier otro caso. Un nodo es homogéneo si su homogeneidad vale 1. La homogeneidad indica si el nodo representa a otra serie de nodos similares del nivel inferior, por lo que únicamente los nodos homogéneos pueden representar regiones en la base de la estructura piramidal formada por el mapa métrico.
- Área $A(x,y,l)$: es igual a la suma de las áreas de sus cuatro nodos hijos. El área indica el número de nodos en la base que posee la región asociada con el nodo, por lo que únicamente tiene sentido para los nodos homogéneos.
- Centroide $C(x,y,l)$: es igual al centro de masa en la base de la región asociada con el nodo. Para calcular el centro de masa se calcula la media de las coordenadas de todos los nodos en la base que forman la región asociada con el nodo, por lo que únicamente tiene sentido para los nodos homogéneos.
- Probabilidad de ocupación $P(x,y,l)$: es igual al valor promediado y umbralizado de las probabilidades de ocupación de sus cuatro nodos hijos. Para obtener el valor promediado del nodo se ponderan las probabilidades de ocupación de cada hijo con sus correspondientes áreas, tras lo cual se umbraliza a los valores C_{occ}, C_{free} o C_{not}. Si el nodo es homogéneo su probabilidad de ocupación coincide con la probabilidad de ocupación de sus cuatro nodos hijos. La probabilidad de ocupación indica si el nodo se corresponde con una zona ocupada (C_{occ}), con una zona libre (C_{free}) o con una zona no explorada (C_{not}).
- Padre $XY(x,y,l)$: si el nodo es homogéneo sus cuatro nodos hijos se enlazan directamente con él. Si el nodo no es homogéneo los cuatro nodos hijos carecen de padre, denominándose nodos huérfanos. Los enlaces entre padres e hijos son imprescindibles para propagar regiones entre los distintos niveles de la estructura piramidal.

Cuando la fase de generación ha finalizado todos los nodos huérfanos que son homogéneos representan regiones del mapa métrico, por lo que proporciona una primera descomposición del entorno en regiones. Por motivos de eficiencia únicamente se consideran regiones válidas aquellas que superan un cierto umbral de área mínima A_{min}, de forma que pequeñas imperfecciones y obstáculos poco definidos en el mapa métrico no generen regiones. Un valor del parámetro A_{min} igual a 16 suele ser adecuado en la mayoría de las aplicaciones para filtrar pequeñas imperfecciones, por lo que será el valor escogido. Es importante notar que debido a la existencia del parámetro A_{min} algunos nodos del mapa métrico pueden no pertenecer a ninguna región, si bien este hecho no altera la correcta operación del método propuesto.

Gracias a la umbralización realizada de las probabilidades de ocupación de los nodos todas las regiones están asociadas a zonas ocupadas ($P(x,y,l)$=C_{occ}), a zonas libres ($P(x,y,l)$=C_{free}) o a zonas no exploradas ($P(x,y,l)$=C_{not}). Esta descomposición realizada del entorno en regiones es similar a la técnicas de los *quad-trees*, si bien a diferencia de estos la complejidad de la descomposición no depende de la distribución de los obstáculos del entorno. Sin embargo, al igual que ocurre con la técnica de los *quadtrees* la descomposición en regiones sigue siendo fuertemente dependiente de dicha distribución,

por lo que es necesario realizar algunas etapas adicionales que reduzcan esta fuerte dependencia.

4. **Reenlazado:** en esta fase se intenta buscar un potencial padre para todos aquellos nodos huérfanos que son homogéneos, es decir, para todos aquellos nodos que representan regiones del mapa métrico. Esta fase pretende que las distintas regiones sean absorbidas por regiones mayores que sean adyacentes y del mismo tipo, contribuyendo a disminuir la dependencia de la descomposición realizada con la distribución de los obstáculos del entorno. Un nodo (x, y, l) es enlazado al nodo padre $(x', y', l+1)$ de un vecino si se cumplen las siguientes condiciones:

   - Si ambos nodos son homogéneos ($H(x, y, l)$=1 y $H(x', y', l+1)$=1), es decir, si ambos nodos representan a otro conjunto de nodos similares.
   - Si ambos nodos poseen igual probabilidad de ocupación ($P(x, y, l)$=$P(x', y', l+1)$), es decir, si representan regiones del mismo tipo.
   - Si la distancia euclídea entre los centroides de las regiones que representan no es muy elevada ($\| C(x, y, l) - C(x', y', l+1) \|_2 < Dist\_Max$), es decir, si dichas regiones están próximas entre sí.

   El parámetro $Dist\_Max$ está relacionado con la dispersión de las regiones en la base, por lo que influye en el tamaño de las mismas. Esta fase de reenlazado se realiza partiendo desde los niveles inferiores hacia los niveles superiores.

5. **Fusión:** es esta fase se intentan fusionar aquellos nodos vecinos que son huérfanos y homogéneos, es decir, aquellos nodos que representan regiones adyacentes del mismo tipo, contribuyendo así mismo a disminuir la dependencia de la descomposición realizada con la distribución de los obstáculos del entorno. Dos nodos vecinos (x_1, y_1, l) y (x_2, y_2, l) se fusionan si se cumplen las siguientes condiciones:

   - Si ambos nodos son huérfanos ($XY(x_1, y_1, l)$=NULL y $XY(x_2, y_2, l)$=NULL) y homogéneos ($H(x_1, y_1, l)$=1 y $H(x_2, y_2, l)$=1), es decir, si representan regiones del mapa métrico.
   - Si ambos nodos poseen igual probabilidad de ocupación ($P(x_1, y_1, l)$=$P(x_2, y_2, l)$), es decir, si representan regiones del mismo tipo.
   - Si la distancia euclídea entre los centroides de las regiones que representan no es muy elevada ($\| C(x_1, y_1, l) - C(x_2, y_2, l) \|_2 < Dist\_Max$), es decir, si dichas regiones están próximas entre sí.

   Esta fase de fusión se realiza partiendo desde los niveles superiores hacia los niveles inferiores.

Una vez finalizado este proceso la estructura jerárquica piramidal generada permite descomponer el entorno en una serie de regiones de forma eficiente, siendo dicha descomposición bastante robusta frente a la distribución de los obstáculos en el entorno. Las distintas regiones pueden

ser identificadas buscando en todos los niveles aquellos nodos huérfanos que son homogéneos, y propagando los enlaces de dichos nodos hacia los niveles inferiores hasta llegar a la base de la estructura piramidal, que se corresponde con el mapa métrico. Además, los distintos parámetros asociados con dichos nodos huérfanos y homogéneos indicarán las características de la región que representan:

- Área $A(x, y, l)$: indica el tamaño de la región.
- Centroide $C(x, y, l)$: indica el centro de masa de la región.
- Probabilidad de ocupación $P(x, y, l)$: indica si la región se corresponde con una zona ocupada (C_{occ}), con una zona libre (C_{free}) o con una zona no explorada (C_{not}).

La figura 6.13 ilustra este proceso completo sobre un mapa métrico muy sencillo de tamaño 8x8 celdas, aunque por simplicidad en la explicación no se va a realizar la fase de crecimiento. La figura 6.13.a representa el mapa métrico umbralizado cuya representación topológica se quiere generar. Todos los nodos de este mapa métrico umbralizado poseen un valor de probabilidad asociado a zonas libres (color blanco), a zonas ocupadas (color negro) o a zonas no exploradas (color gris).

La figura 6.13.b se corresponde con la etapa de generación, donde partiendo del nivel 0 que se corresponde con el mapa métrico de 8x8 celdas se construyen el nivel 1 de 4x4 celdas y el nivel 2 de 2x2 celdas. Los nodos no homogéneos han sido marcados con una "X", mientras que los nodos homogéneos muestran el valor de probabilidad que poseen (blanco para zonas libres, negro para zonas ocupadas o gris para zonas no exploradas). Además, los nodos huérfanos se han marcado con un círculo.

La figura 6.13.c muestra la etapa de reenlazado, indicando los nodos huérfanos y homogéneos que se han enlazado con el padre de los nodos vecinos, siempre que la distancia entre los centroides no era demasiado elevada y que representaran regiones del mismo tipo (libres, ocupadas o no exploradas). En esta figura se han etiquetado con el mismo número aquellos nodos que se han enlazado entre sí.

En la figura 6.13.d se realiza la etapa de fusión, indicando los nodos huérfanos del mismo nivel que son homogéneos que se han fusionado entre sí, siempre que la distancia entre los centroides no era demasiado elevada y que representaran regiones del mismo tipo. En la figura se han agrupado y etiquetado con una letra aquellos nodos que se han fusionado entre sí. En el nivel 1 se han fusionado todos los nodos que corresponden a la zona libre (etiquetados como "A") y todos los nodos que corresponden a la zona no explorada (etiquetada como "C"). Además, existen dos nodos que corresponden a dos zonas ocupadas (etiquetados como "B" y "D"). En el nivel 0 se han fusionado todos los nodos correspondientes a la zona ocupada superior derecha (etiquetados como "E"), todos los nodos correspondientes a la zona ocupada de la izquierda (etiquetados como "F") y todos los nodos correspondientes a la zona libre de la derecha (etiquetados como "G").

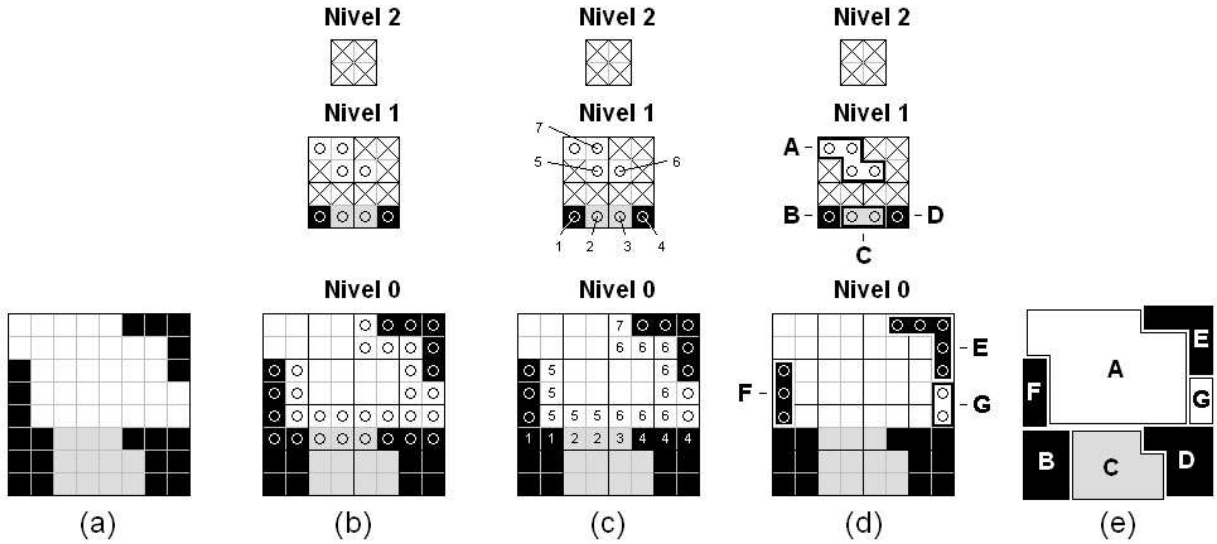

Figura 6.13: Método de generación de la representación topológica del entorno propuesto.

Por último, en la figura 6.13.e se indican las regiones finales obtenidas tras identificarlas en la representación topológica generada. Para ello hay que buscar los nodos huérfanos que son homogéneos en todos los niveles y propagarlos mediante sus enlaces hacia la base formada por el mapa métrico. En la figura se pueden identificar 8 zonas diferentes, que han sido etiquetadas de igual forma que los nodos que las originan.

Es importante remarcar que las regiones de la representación topológica pueden poseer cualquier forma. Además, la probabilidad de ocupación del nodo que representa una determinada región indica si se corresponde con una zona libre, con una zona ocupada o con una zona no explorada del entorno. Así pues, es muy sencillo discriminar aquellas regiones que son de interés para ser incorporadas posteriormente en los procesos de planificación. De hecho, en el sistema desarrollado se utilizan tanto las regiones que representan zonas libres como las regiones que representan zonas no exploradas para planificar las rutas, ignorando en dicho proceso las regiones que representan zonas ocupadas. Incluir las zonas no exploradas en el proceso de planificación permite aumentar la funcionalidad en la operación del sistema, siendo un requisito de los niveles superiores de la Jerarquía de Navegación. Así por ejemplo, es posible planificar rutas a través de zonas no exploradas o rutas para la exploración eficiente de entornos completos en el menor tiempo posible [PPU+02a, PPUS05].

Puesto que el método de generación de la representación topológica utiliza un mapa métrico, el problema de la ambigüedad que poseen las representaciones topológicas se resuelve inherentemente, pues la información métrica del entorno es preservada en la propia estructura jerárquica mediante los distintos nodos y los enlaces entre padres e hijos. Este método, pues, realiza una correcta integración métrico-topológica sin necesidad de ningún procesamiento adicional, y es capaz de extraer la representación topológica del entorno sin necesidad de que el mapa métrico sobre el cual se basa esté completo.

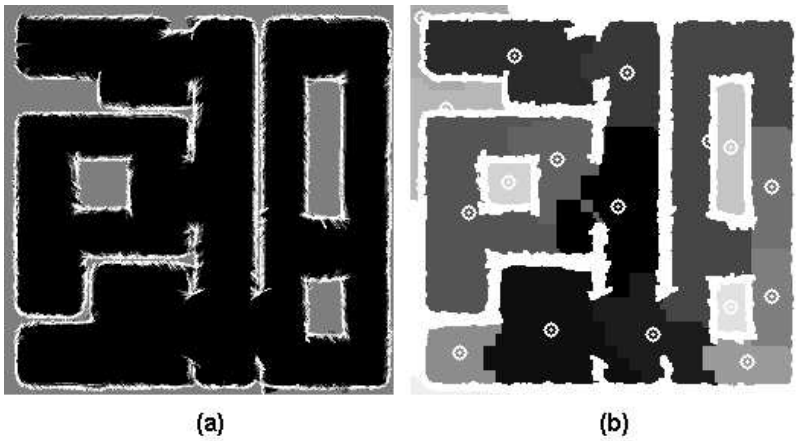

Figura 6.14: Representación topológica de un entorno: a) entorno original; b) representación topológica.

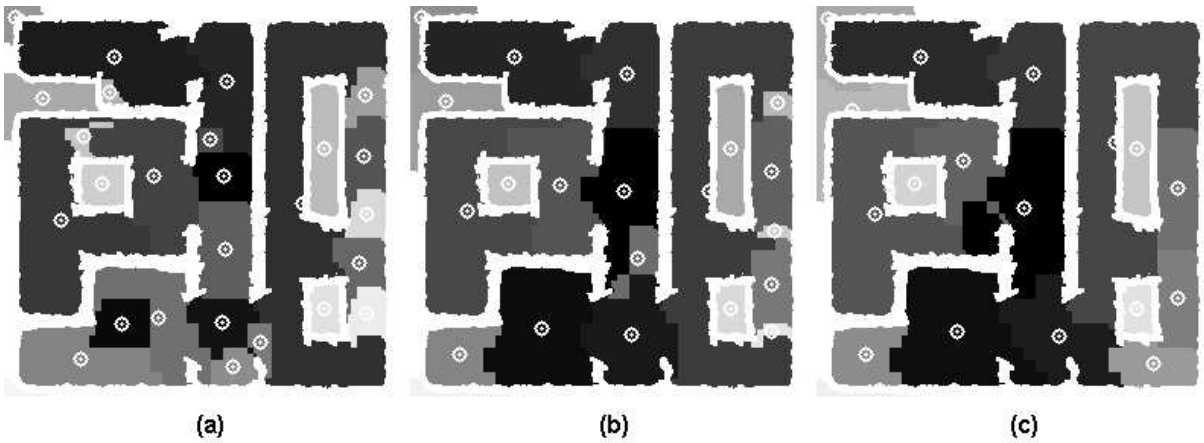

Figura 6.15: Influencia del parámetro $Dist\_Max$ en el proceso de generación de la representación topológica del entorno: a) $Dist\_Max$=20; b) $Dist\_Max$=40; c) $Dist\_Max$=60.

La figura 6.14 muestra este proceso de generación de la representación topológica mediante el método propuesto sobre un entorno real. La figura 6.14.a muestra el entorno original sobre el cual se desea realizar la representación topológica, mientras que la figura 6.14.b muestra la representación topológica obtenida utilizando un crecimiento de obstáculos de 1 celda y un parámetro $Dist\_Max$ de valor 60. En la representación topológica de la figura únicamente se han representado las zonas libres y las no exploradas, descartando las regiones correspondientes a los obstáculos del entorno. También se ha representado para cada región el centroide asociado a la misma.

La influencia del parámetro $Dist\_Max$ en el método propuesto se puede observar en la figura 6.15. Si este parámetro tiene un valor reducido el entorno es descompuesto en un número elevado de pequeñas regiones, mientras que un valor elevado origina un número reducido de grandes regiones. Empíricamente se ha verificado que un valor para este parámetro entre 40 y 60 opera correctamente en la mayoría de los casos.

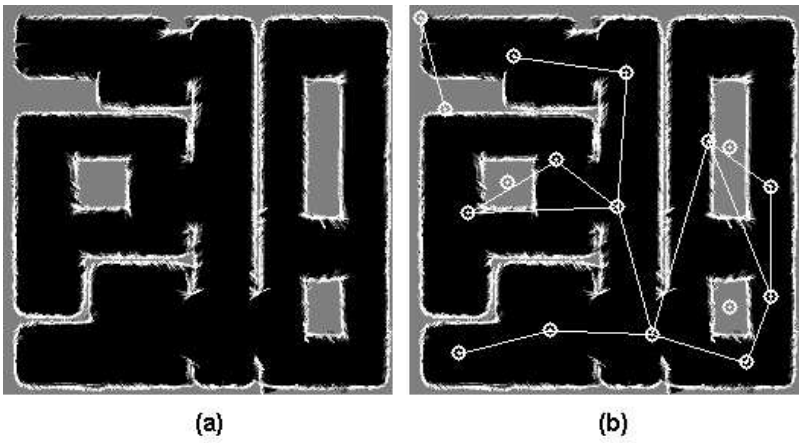

Figura 6.16: Mapa topológico de un entorno: a) entorno original; b) mapa topológico.

Por último, a partir de la representación topológica generada se extrae finalmente el mapa topológico del entorno, el cual representa no sólo las distintas regiones del entorno sino también la conectividad entre las mismas. Para ello, una vez obtenidos los nodos que representan las distintas regiones del entorno es necesario analizar su relación y conectividad. Se considera que dos nodos están conectados entre sí cuando las regiones de la base que representan están en contacto. Para verificar este hecho se comprueba si las *bounding boxes* de ambas regiones se solapan, definiendo la *bounding box* de una región como el menor rectángulo que contiene completamente dicha región. Así pues, si se produce solapamiento entre las *bounding boxes* se considera que ambas regiones están en contacto, por lo que los nodos que las representan se unen directamente en el mapa topológico. Además, el enlace que une estos nodos se etiqueta con un peso proporcional a la distancia euclídea entre los centroides de las regiones que representan. De esta forma, el mapa topológico generado posee información sobre:

- **Regiones:** cada nodo del mapa topológico representa una región del mapa métrico.
- **Conectividad:** cada enlace entre nodos del mapa topológico representa conectividad entre las regiones asociadas del mapa métrico.
- **Distancia:** el peso de cada enlace entre nodos del mapa topológico representa la distancia asociada entre los centroides de las regiones del mapa métrico.

La figura 6.16 muestra el estado final del mapa topológico generado tras realizar el estudio de la conectividad entre los nodos de la representación topológica. La figura 6.16.a muestra el entorno original, mientras que la figura 6.16.b muestra el mapa topológico generado que representa la división del entorno en regiones y la conectividad entre las mismas. En la generación del mapa topológico se ha utilizado un crecimiento de obstáculos de 1 celda y un parámetro $Dist\_Max$ de valor 60.

Para concluir con este apartado se van a analizar las prestaciones temporales del proceso de generación del mapa topológico del entorno propuesto. Obviamente estas prestaciones dependen

Pentium 4 - 2.4 GHz - 512 MB RAM				
	128x128 celdas	256x256 celdas	512x512 celdas	1024x1024 celdas
Umbralización	64.32 ± 0.21 μs	230.7 ± 0.6 μs	912 ± 3 μs	3.737 ± 0.012 ms
Crecimiento	1.07 ± 0.03 μs	1.12 ± 0.03 μs	1.196 ± 0.020 μs	1.29 ± 0.03 μs
Generación	775 ± 4 μs	5.487 ± 0.008 ms	23.478 ± 0.023 ms	94.1 ± 0.3 ms
Reenlazado	82.95 ± 0.24 μs	1.4156 ± 0.0015 ms	5.00 ± 0.04 ms	19.4 ± 0.4 ms
Fusión	48.24 ± 0.13 μs	2.62 ± 0.03 ms	11.40 ± 0.03 ms	46.65 ± 0.06 ms
Conectividad	201.9 ± 0.5 μs	1.3025 ± 0.0017 ms	4.654 ± 0.006 ms	17.986 ± 0.010 ms
Total	1.173 ± 0.004 ms	11.06 ± 0.03 ms	45.45 ± 0.04 ms	181.9 ± 0.6 ms
Pentium 3 - 1.0 GHz - 256 MB RAM				
	128x128 celdas	256x256 celdas	512x512 celdas	1024x1024 celdas
Umbralización	339.0 ± 0.9 μs	1.329 ± 0.008 ms	5.813 ± 0.003 ms	26.16 ± 0.05 ms
Crecimiento	1.85 ± 0.08 μs	3.00 ± 0.03 μs	3.45 ± 0.05 μs	4.8 ± 0.3 μs
Generación	7.19 ± 0.03 ms	49.69 ± 0.07 ms	204.84 ± 0.11 ms	834.8 ± 1.6 ms
Reenlazado	850.5 ± 1.5 μs	10.87 ± 0.03 ms	39.31 ± 0.03 ms	156.0 ± 0.6 ms
Fusión	428 ± 2 μs	25.31 ± 0.04 ms	107.24 ± 0.08 ms	439.9 ± 0.8 ms
Conectividad	1.215 ± 0.013 ms	11.922 ± 0.022 ms	43.72 ± 0.05 ms	175.4 ± 0.5 ms
Total	10.02 ± 0.04 ms	99.12 ± 0.08 ms	400.93 ± 0.13 ms	1.6323 ± 0.0019 s
Pentium 3 - 500 MHz - 192 MB RAM				
	128x128 celdas	256x256 celdas	512x512 celdas	1024x1024 celdas
Umbralización	372.9 ± 1.7 μs	1.472 ± 0.010 ms	6.215 ± 0.021 ms	29.5 ± 0.3 ms
Crecimiento	2.11 ± 0.04 μs	3.13 ± 0.04 μs	4.43 ± 0.04 μs	6.64 ± 0.11 μs
Generación	10.0 ± 0.7 ms	54.16 ± 0.08 ms	237.5 ± 1.3 ms	971 ± 6 ms
Reenlazado	1.103 ± 0.014 ms	17.64 ± 0.04 ms	56.5 ± 0.6 ms	218.6 ± 1.5 ms
Fusión	445.1 ± 1.2 μs	28.48 ± 0.06 ms	126.2 ± 0.6 ms	516 ± 3 ms
Conectividad	2.2 ± 0.3 ms	17.69 ± 0.03 ms	66.2 ± 0.5 ms	260.4 ± 1.4 ms
Total	14.1 ± 0.8 ms	119.45 ± 0.11 ms	493 ± 3 ms	1.996 ± 0.009 s

Tabla 6.3: Prestaciones temporales del proceso de generación del mapa topológico del entorno propuesto.

claramente del tamaño del entorno y de los recursos de la máquina empleada para generar la representación topológica, por lo que se ha repetido la misma prueba para distintos tamaños de entorno y para distintos tipos de máquinas: 128x128, 256x256, 512x512 y 1024x1024 celdas, utilizando en cada caso una máquina de altas prestaciones (Pentium 4 - 2.4 GHz - 512 MB RAM), una máquina de prestaciones medias (Pentium 3 - 1.0 GHz - 256 MB RAM) y una máquina de bajas prestaciones (Pentium 3 - 500 MHz - 192 MB RAM). Evidentemente el tiempo de procesamiento depende del número de celdas empleadas en el crecimiento de obstáculos, utilizando en todos los casos 1 celda. El tiempo medio de procesamiento empleado en cada una de las etapas del método propuesto se muestra en la tabla 6.3, donde los errores se han calculado sobre un intervalo de confianza del 95%.

La tabla 6.3 revela que el proceso de generación del mapa topológico puede llegar a ser bastante costoso. De hecho, en condiciones extremas con un mapa métrico de tamaño 1024x1024 celdas y utilizando una máquina de bajas prestaciones el retardo medio del proceso de generación del

mapa topológico está alrededor de los 2 s, lo que implica que a una velocidad del agente de unos 300 mm/s se recorren aproximadamente 60 cm antes de generar dicho mapa topológico. Este resultado pone de manifiesto que el proceso completo de generación del mapa topológico lleva a cabo un procesamiento bastante profundo del conocimiento que se posee del entorno. Es importante matizar que en el sistema desarrollado el mapa métrico utilizado posee un tamaño de 256x256 celdas, por lo que incluso utilizando una máquina de bajas prestaciones el retardo medio está alrededor de los 120 ms, es decir, un desplazamiento del agente de unos 4 cm a una velocidad de 300 mm/s. Este valor es mucho más asequible para el sistema bajo desarrollo, poniendo de manifiesto que salvo en el caso de poseer mapas métricos muy grandes el proceso de generación del mapa topológico es bastante rápido.

3.3 Cálculo de rutas

Tras generar el mapa topológico del entorno es necesario calcular una ruta que determine las regiones a atravesar para alcanzar el destino final. Para ello hay que utilizar algún planificador que sobre el mapa topológico obtenga un conjunto de nodos que describan la ruta a seguir, teniendo en cuenta que los nodos representan regiones del entorno. Este proceso se puede llevar a cabo fácilmente gracias a que el mapa topológico posee información no sólo de las regiones del entorno y su conectividad, sino también de la distancia relativa entre ellas a través de los enlaces que unen los distintos nodos del mapa topológico.

Puesto que el número de nodos del entorno es mucho menor que el número de celdas del mapa métrico, el proceso de planificación de rutas opera generalmente sobre una cantidad de datos bastante reducida, disminuyendo considerablemente el tiempo requerido para realizar dicha planificación. Gracias a este hecho el planificador de rutas que se ha aplicado sobre este mapa topológico se basa en el algoritmo de **Dijkstra** [Dij59], que es un planificador óptimo muy eficiente con una complejidad del orden $O(n^2)$ para n nodos. No obstante, si el número de nodos del mapa topológico crece considerablemente el algoritmo de Dijkstra puede no ser el más adecuado para realizar el proceso de búsqueda, debido al excesivo aumento de su coste computacional y temporal. En dicho caso se pueden aplicar otro tipo de estrategias de búsqueda más apropiadas como pueden ser los algoritmos genéticos [PPUS05].

La figura 6.17 muestra el proceso de cálculo de la ruta sobre el mapa topológico. La figura 6.17.a representa el entorno sobre el que se desea realizar la planificación de rutas, así como la posición actual del agente (marcada con un círculo) y el destino final (marcado con una "X"). En la figura 6.17.b se representa el mapa topológico calculado sobre dicho entorno, identificando además los nodos asociados a las regiones que contienen la posición actual del agente y el destino final. En la generación del mapa topológico se ha utilizado un crecimiento de obstáculos de 1 celda y un parámetro $Dist\_Max$ de valor 60. Por último, la figura 6.17.c muestra los nodos que forman la ruta calculada, describiendo las regiones que hay que atravesar para alcanzar el destino final desde la posición actual.

Por último, se van a analizar las prestaciones temporales del proceso de cálculo de rutas propuesto. Obviamente estas prestaciones dependen claramente del número de nodos y complejidad

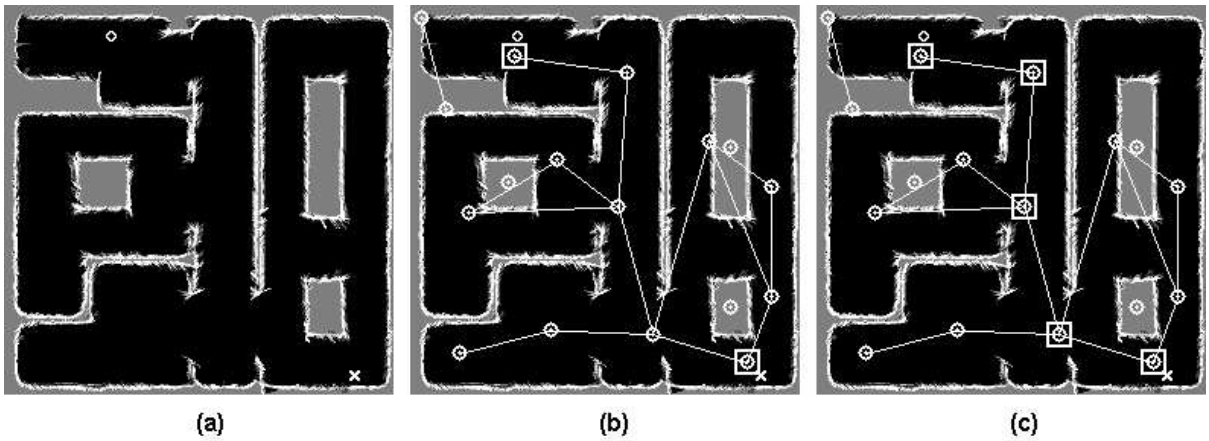

Figura 6.17: Método de planificación de rutas propuesto: a) entorno original; b) mapa topológico; c) cálculo de la ruta.

Pentium 4 - 2.4 GHz - 512 MB RAM				
	128x128 celdas	256x256 celdas	512x512 celdas	1024x1024 celdas
Total	4.1 ± 0.3 μs	7.99 ± 0.04 μs	8.05 ± 0.06 μs	8.76 ± 0.05 μs
Pentium 3 - 1.0 GHz - 256 MB RAM				
	128x128 celdas	256x256 celdas	512x512 celdas	1024x1024 celdas
Total	17.8 ± 0.7 μs	36.01 ± 0.13 μs	38.08 ± 0.19 μs	42.35 ± 0.14 μs
Pentium 3 - 500 MHz - 192 MB RAM				
	128x128 celdas	256x256 celdas	512x512 celdas	1024x1024 celdas
Total	22.16 ± 0.14 μs	39.24 ± 0.15 μs	42.7 ± 0.4 μs	45.6 ± 0.9 μs

Tabla 6.4: Prestaciones temporales del proceso de planificación de rutas propuesto.

del mapa topológico y de los recursos de la máquina empleada para implementar el planificador, por lo que se ha repetido la misma prueba para distintos tamaños de entorno y para distintos tipos de máquinas: 128x128, 256x256, 512x512 y de 1024x1024 celdas, utilizando en cada caso una máquina de altas prestaciones (Pentium 4 - 2.4 GHz - 512 MB RAM), una máquina de prestaciones medias (Pentium 3 - 1.0 GHz - 256 MB RAM) y una máquina de bajas prestaciones (Pentium 3 - 500 MHz - 192 MB RAM). La relación entre el tamaño del entorno y el número de nodos del mapa topológico no es en absoluto directa, pero evidentemente dicho mapa topológico suele poseer mayor número de nodos cuanto mayor sea el tamaño del entorno. El tiempo medio de procesamiento empleado se muestra en la tabla 6.4, donde los errores se han calculado sobre un intervalo de confianza del 95%.

Los resultados de la tabla 6.4 indican que el proceso de planificación de rutas es extremadamente rápido, debido sin lugar a dudas a la compactación realizada por el mapa topológico al reducir considerablemente la cantidad de información a procesar. El retardo medio de la descomposición del camino no supera en ningún caso los 50 μs, pudiendo considerarse pues prácticamente despreciable.

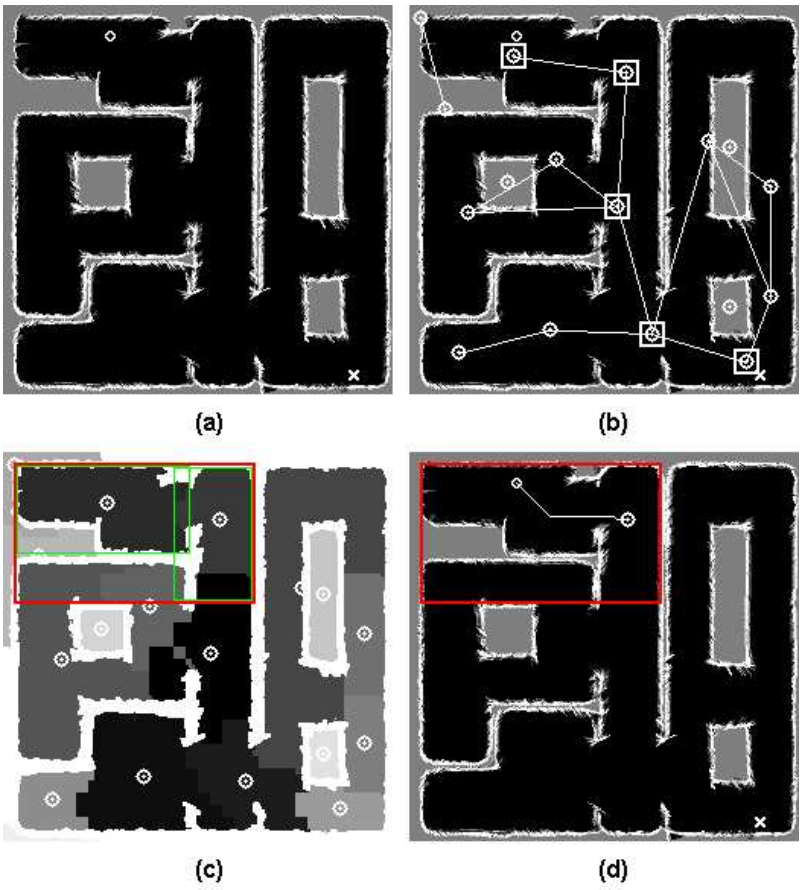

Figura 6.18: Integración de la planificación de rutas con la planificación de caminos: a) entorno original; b) ruta calculada; c) *bounding box* de la zona del mapa métrico a procesar; d) cálculo del camino para alcanzar la siguiente región de la ruta.

3.4 Integración en el sistema de la planificación de rutas

Como ya se ha comentado en el apartado 2.3 una adecuada integración de los procesos de planificación con los comportamientos reactivos constituye un factor crítico en cualquier sistema híbrido. En este apartado se va a abordar el análisis detallado de la operación temporal del proceso de planificación de rutas así como su interacción con el resto del sistema.

El planificador de rutas de la Capa de Planificación amplía las capacidades deliberadas del sistema, actuando directamente sobre el planificador de caminos descrito en el apartado 2. Una vez generada la ruta que describe las regiones a atravesar para alcanzar el destino final es necesario obtener un camino libre de obstáculos para navegar a lo largo de las regiones de la ruta. Con el fin de acelerar este proceso el planificador de rutas tan sólo le proporciona al planificador de caminos el centroide de la próxima región a visitar y la *bounding box* de la zona del mapa métrico que comprende la región actual y la siguiente región a visitar, que será la región sobre la cual se realice el proceso de planificación de caminos. Reduciendo el tamaño del mapa métrico sobre el cual se realiza la planificación de caminos se aumenta la velocidad de dicho proceso, independientemente del tamaño del mapa métrico que represente el entorno.

La figura 6.18 muestra el proceso de integración del planificador de rutas con el planificador de caminos descrito. La figura 6.18.a muestra el entorno original sobre el que se desea realizar la planificación de rutas, así como la posición actual del agente (marcada con un círculo) y el destino final (marcado con una "X"). La figura 6.18.b muestra los nodos que forman la ruta calculada, describiendo las regiones que hay que atravesar para alcanzar el destino final. En la figura 6.18.c se muestra la zona del mapa métrico a procesar para calcular el camino, compuesta por la *bounding box* que engloba la región actual y la siguiente región a visitar. Por último, la figura 6.18.d muestra el camino calculado por el planificador de caminos sobre la zona del mapa métrico indicada para alcanzar la siguiente región de la ruta. En este cálculo del camino se ha empleado un crecimiento de obstáculos de 5 celdas.

Esta forma de operar permite una integración muy eficiente entre los procesos de planificación deliberados presentes en el sistema: la planificación de caminos y la planificación de rutas. El proceso de navegación del sistema combinando la operación de la planificación de caminos con la planificación de rutas se realiza según el siguiente esquema:

1. El usuario introduce un destino final para ser alcanzado, el cual es inmediatamente enviado tanto a la Capa de Navegación como a la Capa de Planificación del sistema.

2. El planificador de rutas genera el mapa topológico del entorno y calcula una ruta para alcanzar la región correspondiente al destino final. Tras calcular la ruta, el planificador de rutas envía al planificador de caminos la posición de la siguiente región de la ruta a alcanzar, así como la zona del mapa métrico que contiene la región actual y la siguiente región.

3. Inicialmente el planificador de caminos procesa el mapa métrico completo. Una vez que el planificador de rutas ha finalizado su procesamiento y dispone de una ruta, el planificador de caminos procesa la zona del mapa métrico que comprende la región actual y la siguiente región a visitar. Tras calcular un camino libre de obstáculos para alcanzar la siguiente región de la ruta, el planificador de caminos lo descompone y envía el siguiente destino intermedio a la Capa de Navegación.

4. Inicialmente la Capa de Navegación se dirige directamente hacia el destino final. Una vez que el planificador de caminos ha finalizado su procesamiento y dispone de un camino descompuesto, la Capa de Navegación comienza a dirigirse sucesivamente al siguiente destino intermedio con el fin de completar el camino calculado.

5. Si se detecta un cambio significativo en el mapa métrico durante el proceso de navegación, el planificador de rutas y/o el planifidador de caminos son relanzados de nuevo para calcular una nueva ruta y/o un nuevo camino más acordes con las nuevas circunstancias del entorno. Mientras tanto, la Capa de Navegación sigue navegando según la ruta y/o el camino anteriormente propuestos.

Se describen a continuación algunas conclusiones importantes extraídas del análisis detallado de este modo de operación:

- Al enviar directamente el destino final tanto a la Capa de Navegación como a la Capa de Planificación se permite que el agente comience a navegar inmediatamente sin necesidad de esperar a que los procesos de planificación finalicen. El principal inconveniente de este esquema radica en el hecho de que la navegación reactiva puede dirigir al agente en una dirección no adecuada inicialmente. No obstante, puesto que los procesos de planificación son bastante rápidos no resulta tan costoso deshacer el camino recorrido en el caso de optar por una dirección errónea, frente a la ventaja que supone tener al agente en movimiento si se ha optado por una dirección correcta.

- Como se ha descrito en el apartado 4.3 del capítulo 3, la ruta y/o el camino calculado se mantienen hasta que se detectan cambios significativos en el entorno. Para ello, se computan el número de celdas del mapa métrico que pasan de libres a ocupadas o viceversa [2], y se considera que el entorno ha variado sustancialmente cuando este número supera un cierto umbral. Por motivos de flexibilidad se han utilizado dos umbrales distintos, el umbral U_{topo} para indicar un cambio significativo del entorno que requiere la generación de un nuevo mapa topológico y de una nueva ruta, y el umbral U_{path} descrito anteriormente en el apartado 2.3 para indicar un cambio significativo del entorno que requiere la generación de un nuevo camino. Con este esquema se optimiza la operación del sistema, pues únicamente se realiza la planificación cuando se considera necesaria, manteniendo la ruta y/o el camino calculado hasta que son lo suficientemente obsoletos por los cambios del entorno.

- Cuando los cambios en el entorno requieren que se recalculen una nueva ruta y/o un nuevo camino la Capa de Navegación sigue navegando según la ruta y/o el camino anteriormente calculados, presentando la ventaja de que el agente no necesita pararse mientras finalizan los nuevos procesos de planificación. El principal inconveniente, de nuevo, radica en el hecho de que puede dirigirse al agente en una dirección equivocada. Sin embargo, se ha preferido mantener este esquema debido a que los procesos de planificación son muy rápidos, por lo que no es tan costoso deshacer la distancia recorrida en el caso de optar por una dirección errónea, frente a la ventaja de no detener el agente si la dirección es acertada.

Es importante notar que el proceso de planificación de rutas sólo interacciona con el proceso de planificación de caminos. Así pues, la adecuada integración entre los procesos de planificación deliberados y los comportamientos reactivos del sistema se siguen realizando a través del planificador de caminos, tal y como se ha descrito en el apartado 2.3.

[2] Recuérdese que tal y como se ha descrito en el apartado 2.1 todas aquellas celdas del mapa métrico cuya probabilidad de ocupación se encuentre por encima de un valor P_{obst} se consideran obstáculos, y las que se encuentren por debajo de dicho valor se consideran espacio libre.

4 Comparativa entre los niveles de planificación

Los procesos de planificación implementados en la Capa de Planificación del sistema poseen distintas características y propiedades. Si bien el proceso de planificación de caminos proporciona el nivel básico de procesamiento deliberado en el sistema, el proceso de planificación de rutas permite optimizar su operación y obtener nuevas capacidades de navegación. Obviamente, el nivel de planificación requerido en cada momento dependerá de los objetivos de navegación perseguidos.

Se realiza a continuación una comparativa entre las características más importantes manifestadas por cada uno de estos procesos de planificación.

4.1 Planificación de caminos

El proceso de planificación de caminos implementa el nivel de Navegación Planificada del sistema, el cual desarrolla el nivel de Respuesta Inducida por Reconocimiento de la Jerarquía de Navegación. Posee importantes ventajas desde el punto de vista del proceso de navegación del agente, entre las cuales caben destacar:

- La trayectoria final recorrida por el agente tiene una longitud cuasi-óptima al poseer la longitud Manhattan mínima.
- La trayectoria final recorrida es bastante suave y minimiza los cambios bruscos de dirección, reduciendo el problema del deslizamiento.
- La trayectoria final recorrida mantiene una distancia de seguridad predeterminada con todos los obstáculos del entorno.

No obstante, la planificación de caminos también posee una serie de inconvenientes y limitaciones, entre las cuales cabe destacar las siguientes:

- Posee una complejidad lineal con el tamaño del mapa métrico, por lo que sus prestaciones temporales se pueden degradar considerablemente en entornos grandes.
- Utiliza el mapa métrico como base de su procesamiento, el cual representa el entorno con una reducida capacidad de abstracción. Así pues, no es posible desarrollar comportamientos de navegación de más alto nivel como planificación de caminos visitando varias ubicaciones o exploración eficiente de entornos completos.

4.2 Planificación de rutas

El proceso de planificación de rutas implementa el nivel de Navegación Topológica del sistema, el cual desarrolla los niveles de Navegación Topológica y de Navegación por Reconocimiento

de la Jerarquía de Navegación. Incorpora importantes ventajas al proceso de planificación de caminos del sistema, entre las que cabe destacar:

- Al utilizar una representación más compacta del entorno permite almacenar entornos mucho mayores, por lo que es posible planificar rutas hacia destinos que se encuentren más allá de la representación métrica almacenada por el sistema.

- Permite proporcionar respuestas más rápidas del proceso de navegación, pues el planificador de rutas selecciona una zona del mapa métrico sobre la que debe operar el planificador de caminos, reduciendo la cantidad de datos que debe procesar este último. Esta característica lo hace especialmente atractivo para su empleo en entornos grandes, pues en estas circunstancias la complejidad de la planificación de caminos no depende del tamaño del mapa métrico, sino del tamaño de la zona a procesar, que es función únicamente de la complejidad del entorno.

- Utiliza el mapa topológico como base de su procesamiento, el cual representa el entorno con una mayor capacidad de abstracción. Así pues, es posible desarrollar comportamientos de navegación complejos como establecimiento de rutas que atraviesen múltiples emplazamientos simultáneamente minimizando la distancia recorrida o la exploración eficiente de entornos completos [PPUS05].

Es importante hacer notar que el proceso de navegación utilizando el planificador de rutas es más rápido que el proceso de navegación utilizando el planificador de caminos. Es cierto que el proceso completo de generación del mapa topológico a partir del mapa métrico, cálculo de la ruta y del camino necesario para alcanzar la siguiente región es más lento que el proceso de cálculo del camino a seguir sobre el mapa métrico completo.

No obstante, hay que recordar que en el sistema los distintos módulos operan concurrentemente y de forma asíncrona. De esta forma, aunque el planificador de rutas esté operando el planificador de caminos no espera a que dicho procesamiento finalice, sino que comienza a operar inmediatamente en la generación de un camino. Inicialmente el planificador de caminos utiliza el mapa métrico completo, pero una vez que la ruta es generada y determinada la región del mapa métrico a ser procesada el planificador de caminos ya no vuelve a procesar el mapa métrico completo, por lo que su operación será más rápida. Si es necesario calcular otra nueva ruta por cambios sustanciales en el entorno el planificador de caminos no espera hasta que dicha ruta sea calculada, sino que sigue operando con la ruta anterior hasta que la nueva esté disponible.

Por lo tanto, es evidente que al utilizar la planificación de rutas se restringe la zona del mapa métrico empleada por el planificador de caminos, reduciendo la cantidad de datos que este último procesa y el tiempo empleado en esta planificación. Este esquema combina una gran eficiencia en el modo de operación del sistema con una actualización continua de los procesos de planificación, por lo que se garantiza que siempre se utiliza el modelo más actualizado que se posee del entorno.

No obstante, la planificación de rutas también posee algunos inconvenientes y limitaciones, siendo la más importante la siguiente:

- Al visitar regiones sucesivamente, representadas por sus centroides, el recorrido final que describe el agente no es tan óptimo, pues hay que visitar el centroide de cada región antes de alcanzar la siguiente.

Es evidente que ambos niveles de planificación proporcionan características complementarias, por lo que una adecuada integración de ambos procesos contribuye a aumentar la flexibilidad en la capacidad de navegación del sistema.

5 Implementación de la Capa de Planificación

La Capa de Planificación es la responsable de generar los planes adecuados que permitan alcanzar el destino final de una forma segura y eficiente mientras se dirige adecuadamente al sistema reactivo.

5.1 Módulo *PathPlanning*

La planificación de caminos de la Capa de Planificación se va a implementar mediante el módulo *PathPlanning*, encargado de calcular un camino libre de obstáculos hasta el destino final, aplicando el método descrito en el apartado 2. Para ello, este módulo debe implementar las siguientes funciones:

- **Planificación:** cálculo sobre el mapa métrico del camino libre de obstáculos a seguir para alcanzar el destino final.
- **Seguimiento:** descomposición del camino a seguir en un conjunto de destinos intermedios a seguir sucesivamente de forma reactiva.

El interfaz que define los datos de entrada y salida que este módulo comparte con el resto de módulos es el siguiente:

- **Datos de entrada:**
  - **Destino final:** generado por el módulo *IFUser* si el sistema se encuentra en el nivel de Navegación Planificada o por el módulo *MapTopo* si el sistema se encuentra en el nivel de Navegación Topológica.
  - **Posición del agente:** generada por el módulo *IFRobot*.
  - **Mapa métrico:** generada por el módulo *MapMetric*.

  - **Bounding box:** generada por el módulo *MapTopo*.

- **Datos de salida:**

  - **Camino calculado:** utilizado por el módulo *IFUser*.
  - **Próximo destino intermedio:** utilizado por los módulos *Navigation* y *IFUser*.

5.2 Módulo *MapTopo*

La planificación de rutas de la Capa de Planificación se va a implementar mediante el módulo *MapTopo*, encargado de calcular una ruta hasta el destino final, aplicando el método descrito en el apartado 3. Para ello, este módulo debe implementar las siguientes funciones:

- **Representación topológica:** generación del mapa topológico que representa el entorno donde el agente desarrolla su actividad mediante un conjunto de regiones conectadas entre sí.
- **Planificación:** cálculo de la ruta a seguir para alcanzar el destino final.

El interfaz que define los datos de entrada y salida que este módulo comparte con el resto de módulos es el siguiente:

- **Datos de entrada:**

  - **Destino final:** generado por el módulo *IFUser* si el sistema se encuentra en el nivel de Navegación Topológica.
  - **Posición del agente:** generada por el módulo *IFRobot*.
  - **Mapa métrico:** generado por el módulo *MapMetric*.

- **Datos de salida:**

  - **Mapa topológico:** utilizado por el módulo *IFUser*.
  - **Ruta calculada:** utilizada por el módulo *IFUser*.
  - **Próxima región:** utilizada por el módulo *PathPlanning*.
  - **Bounding box:** utilizada por el módulo *PathPlanning*.

6 Conclusiones

En el presente capítulo se ha abordado el diseño de la Capa de Planificación del sistema, responsable de desarrollar los procesos de planificación deliberados necesarios en todo esquema

híbrido con el fin de aumentar la eficiencia del sistema. Existen distintos niveles de planificación posibles, según el grado de abstracción que se persiga en el proceso de navegación. A mayor grado de abstracción, los resultados obtenidos en el proceso de planificación son inevitablemente más generales pero permiten un mayor grado de procesamiento para realizar operaciones más complejas. En el sistema propuesto se han implementado dos niveles de planificación distintos: un proceso de planificación de caminos y un proceso de planificación de rutas.

La **planificación de caminos** presenta el nivel de planificación básico en el sistema, y se basa en el cálculo de un camino libre de obstáculos a través del entorno que permita al agente alcanzar el destino final mediante la navegación reactiva. Utiliza una representación métrica del entorno completo sobre el cual el agente desempeña su actividad, por lo que permite optimizar el camino calculado según determinados criterios mientras dirige adecuadamente al sistema reactivo evitando las trampas de mínimos locales. Obviamente existen distintas alternativas a la hora de implementar un esquema de planificación de caminos, optándose finalmente por el método de los **Campos Potenciales** mediante una propagación de ondas. Este método exhibe unas prestaciones excelentes, proporcionando caminos cortos, suaves y seguros, a la vez que posibilita una muy sencilla descomposición del camino obtenido para realizar un adecuado seguimiento del mismo.

La **planificación de rutas** presenta el nivel de planificación más avanzado en el sistema, al descomponer el entorno en regiones conectadas entre sí y calcular una ruta con las regiones que es necesario atravesar para alcanzar el destino final. Utiliza una representación topológica del entorno, la cual posee un mayor nivel de abstracción y permite obtener comportamientos más complejos como generar rutas sobre grandes entornos mayores que los representados por el mapa métrico, trazar rutas para visitar diferentes ubicaciones simultáneamente o realizar la exploración eficiente de entornos completos. Una vez generada la ruta adecuada se basa en el nivel de planificación de caminos para dirigirse a las distintas regiones del entorno sucesivamente. Existen distintas alternativas a la hora de generar una representación topológica del entorno, escogiéndose **una estructura jerárquica piramidal** al disponer de una representación métrica previamente generada.

Ambos niveles de planificación presentan características complementarias, por lo que su combinación permite aumentar la funcionalidad del sistema. El nivel de planificación de caminos permite obtener resultados más eficientes pero puede llegar a ser excesivamente lento en grandes entornos. Por su parte, el nivel de planificación de rutas proporciona resultados más rápidos en grandes entornos y manifiesta capacidades de navegación más avanzadas, aunque las trayectorias finales obtenidas suelen ser más largas.

Por último, se ha descrito la implementación de la Capa de Planificación mediante los módulos *PathPlanning* y *MapTopo*, responsables de desarrollar la planificación de caminos y la planificación de rutas respectivamente. Las pruebas realizadas demuestran así mismo que ambos procesos de planificación son bastante rápidos.

Capítulo 7

Resultados

Una vez concluida la implementación del sistema es necesario comprobar su correcto funcionamiento. Los resultados obtenidos durante este proceso de validación no sólo permite detectar posibles errores en el desarrollo del sistema, sino también caracterizar las prestaciones finales obtenidas por el mismo. En este capítulo se presentan y analizan los diferentes resultados obtenidos en la realización de diversas pruebas sobre el sistema desarrollado.

Este capítulo se ha organizado como se indica a continuación. En el apartado 1 se presenta el entorno de pruebas que se va a utilizar para analizar el funcionamiento del sistema, describiendo los recursos y elementos empleados. En el apartado 2 se muestran las pruebas realizadas sobre el estilo de la arquitectura *DLA*, mostrando las prestaciones obtenidas con los diferentes esquemas propuestos: distribuido básico, distribuido síncrono y local síncrono. En el apartado 3 se presentan las pruebas realizadas sobre la estructura de la arquitectura *DLA*, mostrando la capacidad de navegación del sistema en cada uno de los niveles de navegación desarrollados: Navegación Reactiva, Navegación Planificada y Navegación Deliberada. Estas pruebas se llevarán a cabo tanto en entornos simulados como en entornos reales, con el fin de caracterizar el funcionamiento del sistema y la influencia de los posibles errores introducidos durante su operación en el mundo real. Por último, en el apartado 4 se describen las principales conclusiones extraídas a lo largo del presente capítulo.

1 Entorno de pruebas

Antes de proceder a describir las pruebas realizadas para comprobar el correcto funcionamiento de la arquitectura *DLA* se va a describir el entorno sobre el cual se han realizado estas pruebas, enumerando los recursos utilizados en el desarrollo de las mismas. La descripción del entorno de pruebas es importante, puesto que las prestaciones finales obtenidas en el sistema dependen directamente de los recursos utilizados en la implementación del mismo.

La figura 7.1 muestra el entorno de pruebas utilizado, formado por una serie de máquinas distribuidas sobre una red de área local. Como agente autónomo móvil se van a emplear un

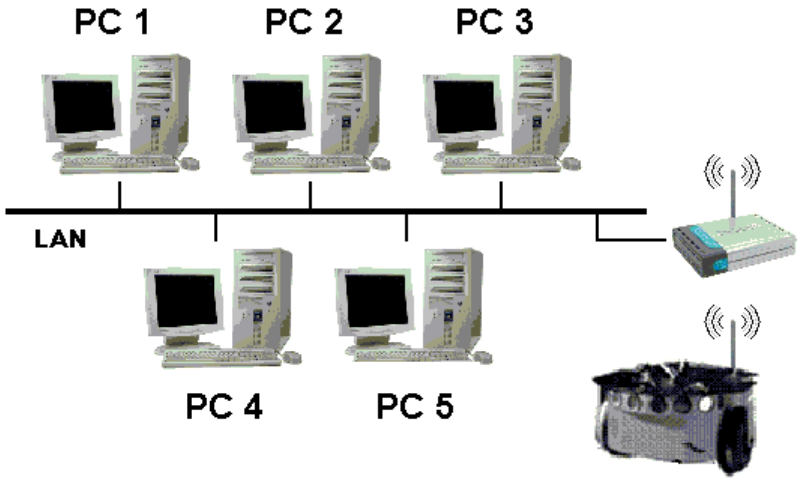

Figura 7.1: Entorno utilizado en las pruebas

Máquina	Procesador	Memoria	Sistema Operativo	Prestaciones
PC 1	Pentium 4 - 2.4 GHz	512 MB	Red Hat 9.0	Altas
PC 2	Pentium 4 - 2.4 GHz	512 MB	Red Hat 9.0	Altas
PC 3	Pentium 3 - 1.0 Ghz	256 MB	Red Hat 9.0	Medias
PC 4	Pentium 3 - 500 MHz	192 MB	Red Hat 9.0	Bajas
PC 5	Pentium 3 - 500 MHz	192 MB	Red Hat 9.0	Bajas

Tabla 7.1: Recursos utilizados en las pruebas

simulador para agentes robóticos y una plataforma robótica *Pioneer P2AT*. El objetivo del simulador es simplificar algunas pruebas donde se intentan analizar parámetros que no tienen que ver con el agente en sí. El simulador empleado es el correspondiente a la plataforma robótica *Nomad 200*, proporcionado por *Nomadics* y desarrollado para operar sobre *Linux*. Sobre dicho simulador se ha configurado un agente con 16 sensores sonar y un sistema de posicionamiento basado en odometría. La plataforma robótica *Pioneer P2AT* está equipada con 8 sensores sonar *Polaroid*, una brújula, un sistema de posicionamiento basado en odometría y una conexión inalámbrica mediante un punto de acceso para conectarse con la red anteriormente citada.

Para realizar un análisis más profundo de las pruebas se han escogido y utilizado máquinas con distintos recursos de las disponibles en la red empleada: máquinas de altas prestaciones, máquinas de prestaciones medias y máquinas de bajas prestaciones. La tabla 7.1 resume los recursos utilizados en las pruebas y enumera las prestaciones de cada una de las máquinas empleadas.

Sobre este entorno de pruebas se van a distribuir los distintos elementos necesarios para implementar el sistema bajo desarrollo. Estos elementos son los siguientes:

- **Arquitectura *DLA*:** constituye el núcleo del sistema. Está formada por el servidor central de la arquitectura *DLAServer*, descrito en el apartado 3.2 del capítulo 3.

- **Sistema de navegación:** compuesto por los distintos algoritmos de procesamiento necesarios para implementar el sistema de navegación en entornos dinámicos no estructurados. Está formado por las capas y módulos descritos en el apartado 4.2.1 del capítulo 3: la Capa Física formada por el módulo *IFRobot*, la Capa de Navegación formada por el módulo *Navigation*, la Capa de Representación formada por el módulo *MapMetric*, la Capa de Planificación formada por los módulos *PathPlanning* y *MapTopo*, y la Capa de Comando formada por el módulo *IFUser*.
- **Servidor del sistema *CBR*:** proporciona los comandos de navegación reactiva para la evitación de obstáculos. Está formado por el servidor *CBRServer*, descrito en el apartado 3.1 del capítulo 5.
- **Servidor hardware del robot:** desarrolla el interfaz para el control del agente autónomo móvil. Está formado por el servidor *RobotServer*, descrito en el apartado 4.2.2 del capítulo 3.

Respecto a los parámetros que se han utilizado en el sistema se muestran en la tabla 7.2, agrupados según los módulos que los utilizan. Para cada parámetro se indica el valor empleado en las pruebas realizadas, especificando brevemente su significado y mostrando el capítulo y el apartado de la Tesis donde se describen con mayor detalle.

Una vez presentado el entorno de pruebas que se va a utilizar se describen a continuación las distintas pruebas realizadas para comprobar la operación del sistema.

2 Estilo de la arquitectura *DLA*

En este apartado se pretenden analizar las prestaciones obtenidas por los distintos esquemas de la arquitectura *DLA*, descritos en el apartado 3.2 del capítulo 3: **esquema distribuido básico**, **esquema distribuido síncrono** y **esquema local síncrono**.

Uno de los aspectos más importantes al realizar cualquier tipo de pruebas radica en la correcta elección de algún parámetro que muestre las variaciones en las prestaciones obtenidas a lo largo de las diferentes pruebas. Puesto que se pretende implementar un sistema de navegación en entornos dinámicos no estructurados sobre un agente autónomo móvil, un parámetro interesante de las prestaciones finales obtenidas por el sistema es el retardo en generar un comando de movimiento para el agente a partir de los estímulos captados del entorno. Así pues, el parámetro que se va a utilizar para cuantificar las prestaciones del sistema es la **latencia**, definida como el tiempo transcurrido desde que se obtienen los datos de entrada, que en este caso es la medida de los sensores y la posición del agente, hasta que se dispone de los datos de salida, que en este caso son los comandos de movimiento adecuados para realizar la navegación por parte del agente.

Puesto que el sistema desarrollado pertenece a la categoría de sistemas híbridos que integran procesos de planificación deliberados con comportamientos reactivos, se han definido dos parámetros de latencia distintos en el sistema por su interés:

IFRobot			
Parámetro	**Valor**	**Descripción**	**Apartado**
U_{sec}	20 cm	Distancia de seguridad para detener al agente	Cap. 4 - 1.1
Navigation			
Parámetro	**Valor**	**Descripción**	**Apartado**
U_{adap}	6000	Distancia para adaptar el caso devuelto	Cap. 5 - 2.4.2
C_{soft}	$2.3 \cdot 10^{-2}$	Coeficiente del factor de suavidad	Cap. 5 - 2.4.3
C_{dist}	$2.3 \cdot 10^{-2}$	Coeficiente del factor de distancia	Cap. 5 - 2.4.3
C_{sec}	$5.8 \cdot 10^{-5}$	Coeficiente del factor de seguridad	Cap. 5 - 2.4.3
K_{soft}	1	Coeficiente de ponderación del factor de suavidad	Cap. 5 - 2.4.3
K_{dist}	1	Coeficiente de ponderación del factor de distancia	Cap. 5 - 2.4.3
K_{sec}	1	Coeficiente de ponderación del factor de seguridad	Cap. 5 - 2.4.3
U_{disp}	50	Distancia para captura de nuevos casos	Cap. 5 - 2.4.4
$Dist\_Cluster$	500	Distancia mínima entre clases	Cap. 5 - 2.5
U_{prox}	80 cm	Distancia para evitación de obstáculos	Cap. 5 - 3.1
G_{attrac}	1	Coeficiente de ponderación de la fuerza atractiva	Cap. 5 - 3.1
G_{rep}	4	Coeficiente de ponderación de la fuerza repulsiva	Cap. 5 - 3.1
MapMetric			
Parámetro	**Valor**	**Descripción**	**Apartado**
Num_Col	256	Número de columnas del mapa métrico	Cap. 4 - 2.1
Num_Row	256	Número de filas del mapa métrico	Cap. 4 - 2.1
$Size\_Cell$	4 cm	Tamaño de la celda del mapa métrico	Cap. 4 - 2.1
2β	10^o	Anchura del lóbulo principal del sensor sonar	Cap. 4 - 2.1
P_{occ}	20	Incremento de probabilidad para celdas ocupadas	Cap. 4 - 2.1
P_{free}	10	Reducción de probabilidad para celdas libres	Cap. 4 - 2.1
U_{path}	200	Celdas modificadas para planificación de caminos	Cap. 4 - 2.2
U_{topo}	200	Celdas modificadas para planificación de rutas	Cap. 4 - 2.2
PathPlanning			
Parámetro	**Valor**	**Descripción**	**Apartado**
P_{obst}	60	Umbral de probabilidad para obstáculos	Cap. 6 - 2.1
$Obst\_Grow$	5	Celdas utilizadas para el crecimiento de obstáculos	Cap. 6 - 2.1
MapTopo			
Parámetro	**Valor**	**Descripción**	**Apartado**
U_{occ}	40	Umbral de probabilidad para celdas ocupadas	Cap. 6 - 3.2
U_{free}	60	Umbral de probabilidad para celdas libres	Cap. 6 - 3.2
C_{occ}	100	Nivel de probabilidad para celdas ocupadas	Cap. 6 - 3.2
C_{free}	0	Nivel de probabilidad para celdas libres	Cap. 6 - 3.2
C_{not}	50	Nivel de probabilidad para celdas no exploradas	Cap. 6 - 3.2
$Obst\_Grow$	1	Celdas utilizadas para el crecimiento de obstáculos	Cap. 6 - 3.2
A_{min}	16	Número mínimo de celdas para formar una región	Cap. 6 - 3.2
$Dist\_Max$	60	Distancia máxima entre regiones	Cap. 6 - 3.2

Tabla 7.2: Parámetros empleados en la implementación del sistema.

- **Latencia reactiva:** correspondiente al retado asociado a los comportamientos reactivos de la Capa de Navegación. El sistema reactivo debe ser rápido en sus respuestas, puesto que debe reaccionar lo antes posible a los estímulos de entrada para evitar situaciones críticas y de peligro que puedan desembocar en colisiones. Así pues, esta latencia debe ser lo más baja posible para mantener la integridad del sistema.

- **Latencia total:** correspondiente al retardo total asociado al sistema, compuesto por los procesos de planificación deliberados de la Capa de Planificación y por los comportamientos reactivos de la Capa de Navegación. Esta medida pone de manifiesto las prestaciones totales del sistema, indicando el tiempo empleado por el mismo en generar los planes adecuados ante un comando de entrada mediante la Capa de Planificación y mover al agente autónomo de acuerdo a dichos planes mediante la Capa de Navegación. Esta latencia no es tan crítica como la latencia reactiva, aunque obviamente es deseable tener valores tan bajos como sea posible.

Como se ha descrito en el apartado 4.1 del capítulo 3 el sistema implementado permite desarrollar distintos niveles de navegación, por lo que las pruebas realizadas van a poner de manifiesto las prestaciones del sistema en cada uno de estos niveles:

- **Navegación Reactiva:** indicado en las pruebas con el término "**Nav**", hace referencia al nivel de navegación reactiva.

- **Navegación Planificada:** indicado en las pruebas con el término "**Path**", hace referencia al nivel de navegación deliberada que usa planificación de caminos.

- **Navegación Topológica:** indicado en las pruebas con el término "**Topo**", hace referencia al nivel de navegación deliberada que usa planificación de rutas.

Cada uno de estos niveles aumenta progresivamente el número de módulos activos en el sistema al utilizar niveles de navegación más avanzados, por lo que es de esperar que la latencia también aumente progresivamente. Es interesante notar que en el caso de Navegación Reactiva no se realiza ningún tipo de planificación, por lo que ningún módulo de la Capa de Planificación estará activo. Así pues, en estas circunstancias la latencia reactiva y la latencia total del sistema deben coincidir.

Por último, indicar que en las pruebas realizadas sobre el estilo de la arquitectura *DLA* se pretenden comparar las prestaciones obtenidas entre los distintos esquemas disponibles, independientemente del tipo de agente utilizado. Así pues, para simplificar el desarrollo de las mismas, se va a emplear el simulador de agentes robóticos proporcionado por *Nomadics* en lugar de la plataforma robótica *Pioneer P2AT*.

Se procede a continuación a describir y analizar las pruebas realizadas sobre los distintos esquemas de la arquitectura *DLA*, indicando que en todas las pruebas los errores se han calculado sobre un intervalo de confianza del 95%.

Elemento	Máquina	Prestaciones
DLAServer	PC 2	Altas
IFRobot	PC 1	Altas
Navigation	PC 1	Altas
MapMetric	PC 3	Medias
PathPlanning	PC 3	Medias
MapTopo	PC 3	Medias
IFUser	PC 5	Bajas
CBRServer	PC 1	Altas
RobotServer	PC 4	Bajas
Simulador	PC 4	Bajas

Tabla 7.3: Distribución empleada en la prueba sobre el esquema distribuido básico de la arquitectura *DLA*

2.1 Esquema distribuido básico

Se pretende en primer lugar analizar las prestaciones obtenidas mediante el esquema distribuido básico de la arquitectura *DLA*. Aunque ya se realizó una comparativa en dicho apartado entre el esquema distribuido básico de la arquitectura *DLA* y el esquema propuesto por Dulimarta bajo condiciones extremas de tráfico, se va a realizar otro análisis comparativo entre ambos, pero esta vez sobre el sistema de navegación real implementado.

La distribución de los distintos elementos del sistema y las máquinas empleadas en esta prueba es la que se muestra en la tabla 7.3. Se han intentado asignar inteligentemente los distintos elementos del sistema entre los recursos disponibles. El servidor central de la arquitectura *DLAServer* es un elemento crítico, puesto que sus prestaciones determinan las prestaciones del sistema total, por lo que se ha ubicado en una máquina de altas prestaciones. Igualmente, la Capa Física formada por el módulo *IFRobot* y la Capa de Navegación formada por el módulo *Navigation* constituyen elementos críticos del sistema, por ser necesarios para implementar el sistema reactivo responsable de garantizar la integridad del agente. Es por ello por lo que ambas capas se han ubicado en una máquina de altas prestaciones. Como parte del sistema reactivo el servidor *CBRServer* también constituye un elemento crítico, ubicándose igualmente en una máquina de altas prestaciones. La Capa de Comando formada por el módulo *IFUser* desarrolla el interfaz de usuario que acepta comandos y muestra la información del agente, por lo que no es un elemento crítico y ha sido ubicado en una máquina de bajas prestaciones. Por su parte, la Capa de Representación formada por el módulo *MapMetric* y la Capa de Planificación formada por los módulos *PathPlanning* y *MapTopo* son necesarios para implementar los procesos de planificación deliberados, que aun no siendo críticos es deseable que tengan unos tiempos de respuesta bajos, por lo que han sido ubicados en una máquina de prestaciones medias. Por último, el simulador del agente robótico y el servidor hardware del robot *RobotServer* tampoco son elementos críticos, pues controlan un agente físico que opera a menor velocidad que el sistema de navegación por la propia inercia que posee, por lo que también han sido ubicados en una máquina de bajas prestaciones.

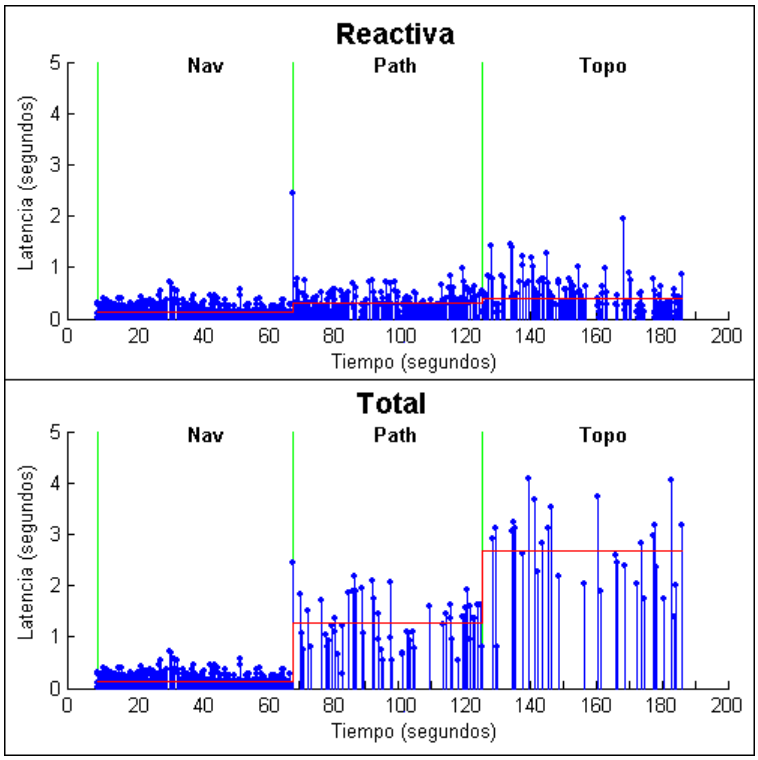

Figura 7.2: Prestaciones obtenidas por el esquema propuesto por Dulimarta

Esquema	Latencia	Nav	Path	Topo
Dulimarta	Reactiva	$122 \pm 10\ ms$	$310 \pm 30\ ms$	$390 \pm 50\ ms$
	Total	$122 \pm 10\ ms$	$1270 \pm 130\ ms$	$2700 \pm 300\ ms$

Tabla 7.4: Prestaciones obtenidas por el esquema propuesto por Dulimarta

La figura 7.2 muestra las prestaciones obtenidas por el esquema propuesto por Dulimarta en los distintos niveles de navegación. Cada nivel de navegación se ha mantenido durante 60 segundos aproximadamente, y se han medido la latencia reactiva y la latencia total del sistema en cada uno de estos niveles de navegación. Cada medida se corresponde con un comando de movimiento enviado al agente autónomo, y el valor de la medida indica en qué instante se adquirieron los datos que han originado dicho comando de movimiento. Así por ejemplo, alrededor del segundo 170 correspondiente al nivel de Navegación Topológica hay una medida con una latencia reactiva de unos 2 segundos aproximadamente. Esto indica que en ese preciso momento se envió al agente un comando de movimiento cuyos datos se obtuvieron unos 2 segundos antes, alrededor del segundo 168. Por ello la latencia reactiva del sistema es de aproximadamente 2 segundos en ese instante de tiempo. Sobre cada período completo de tiempo correspondiente a un nivel de navegación se ha calculado y mostrado la media en las medidas de latencia obtenidas. Así pues, la densidad de medidas obtenidas es un valor indicativo de la frecuencia de ejecución del ciclo básico del sistema, mientras que la media es un valor indicativo del retardo total del sistema en reaccionar frente a nuevos estímulos de entrada. La tabla 7.4 muestra los valores numéricos de la media calculada para la latencia reactiva y para la latencia total.

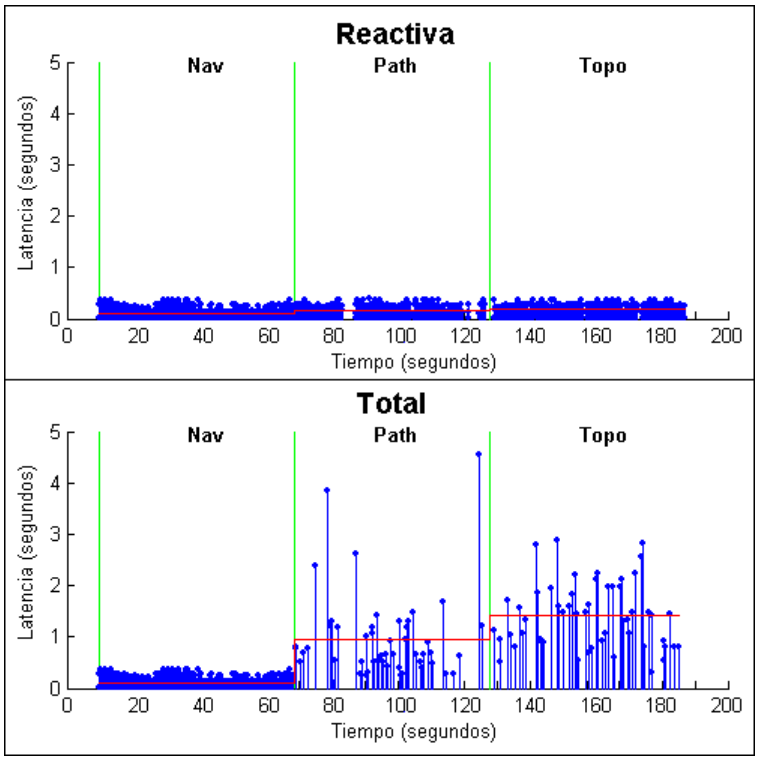

Figura 7.3: Prestaciones obtenidas por el esquema distribuido básico de la arquitectura *DLA*

Esquema	Latencia	Nav	Path	Topo
Distribuido básico	Reactiva	$102 \pm 7\ ms$	$171 \pm 12\ ms$	$198 \pm 11\ ms$
	Total	$102 \pm 7\ ms$	$960 \pm 240\ ms$	$1410 \pm 180\ ms$

Tabla 7.5: Prestaciones obtenidas por el esquema distribuido básico de la arquitectura *DLA*

Los resultados obtenidos indican que tanto la latencia reactiva como la latencia total aumentan según se utilizan niveles de navegación más avanzados, al aumentar progresivamente el número de módulos activos en el sistema. Así mismo se puede observar que en el nivel de Navegación Reactiva tanto la latencia reactiva como la latencia total del sistema coinciden, puesto que la Capa de Planificación en tal caso permanece inactiva. En el caso peor, correspondiente obviamente al nivel de Navegación Topológica donde todos los módulos están activos, la latencia reactiva es de unos 400 ms, lo que implica que a una velocidad del agente de unos 300 mm/s se recorren aproximadamente 12 cm antes de reaccionar frente a un nuevo estímulo. En el caso peor la latencia total es de unos 2.7 s, por lo que a una velocidad del agente de unos 300 mm/s se recorren aproximadamente 81 cm antes de generar los planes adecuados ante un nuevo comando.

La figura 7.3 muestra las prestaciones obtenidas por el esquema distribuido básico de la arquitectura *DLA* en los distintos niveles de navegación, mientras que la tabla 7.5 muestra los valores numéricos de la media calculada para la latencia reactiva y para la latencia total. La distribución de los distintos elementos del sistema y las máquinas empleadas en esta prueba es la misma que la realizada en la prueba sobre el esquema propuesto por Dulimarta, mostrado en la tabla 7.3.

En el caso peor la latencia reactiva es de unos 200 ms, lo que implica que a una velocidad del agente de unos 300 mm/s se recorren aproximadamente 6 cm antes de reaccionar frente a un nuevo estímulo. En el caso peor la latencia total es de unos 1.4 s, por lo que a una velocidad del agente de unos 300 mm/s se recorren aproximadamente 42 cm antes de generar los planes adecuados ante un nuevo comando.

Los resultados obtenidos corroboran plenamente los previamente extraídos en el apartado 3.2.1 del capítulo 3, donde se realizó una comparativa bajo condiciones de tráfico extremas entre el esquema distribuido básico de la arquitectura *DLA* y el esquema propuesto por Dulimarta. Como se puede observar, el esquema *DLA* es claramente superior al esquema propuesto por Dulimarta, reduciendo considerablemente los retardos del sistema, lo que permite que el agente reaccione con mayor rapidez tanto a nuevos estímulos captados por los sensores como a nuevos comandos introducidos por el usuario. Otra conclusión que se puede extraer es que los retardos mostrados por el esquema distribuido básico de la arquitectura *DLA* son bastante aceptables, pues a una velocidad del agente de 300 mm/s únicamente se recorren unos 6 cm antes de reaccionar frente a nuevos estímulos.

2.2 Esquema distribuido síncrono

Con la incorporación del esquema distribuido síncrono se tiene la capacidad de activar y desactivar selectivamente distintos módulos en el sistema. En el esquema distribuido básico todos los módulos se encuentran activos simultáneamente, si bien sólo procesan datos aquellos que se necesitan para el nivel de navegación seleccionado. Los módulos no necesarios se encuentran en un bucle de espera, consumiendo recursos y generando tráfico en el sistema que ralentizan la operación del resto de módulos y aumentan el tiempo de servicio del servidor central de la arquitectura *DLAServer*. Es de esperar, pues, que el esquema distribuido síncrono mejore los tiempos de respuesta al desactivar por completo los módulos que no son necesarios en el mismo.

La distribución de los distintos elementos del sistema y las máquinas empleadas en esta prueba es la que se muestra en la tabla 7.6, que es la misma distribución utilizada en la prueba del esquema distribuido básico de la arquitectura *DLA*.

La figura 7.4 muestra la frecuencia de ejecución de todos los módulos del sistema en el esquema distribuido básico, mientras que la figura 7.5 muestra dicha frecuencia de ejecución en el esquema distribuido síncrono. En dichas figuras se ha incluido una medida cada vez que un módulo es ejecutado, por lo que la densidad de medidas es un indicativo de la frecuencia de ejecución de un módulo determinado.

En el esquema distribuido básico todos los módulos se están ejecutando continuamente, si bien sólo algunos de ellos procesan datos útiles en función del nivel de navegación seleccionado como se ha comentado anteriormente. En el esquema distribuido síncrono únicamente los módulos que son necesarios se están ejecutando en el sistema, encontrándose el resto inactivos.

De acuerdo a la implementación de la Capa de Planificación descrita en el apartado 4.2.1 del

Elemento	Máquina	Prestaciones
DLAServer	PC 2	Altas
IFRobot	PC 1	Altas
Navigation	PC 1	Altas
MapMetric	PC 3	Medias
PathPlanning	PC 3	Medias
MapTopo	PC 3	Medias
IFUser	PC 5	Bajas
CBRServer	PC 1	Altas
RobotServer	PC 4	Bajas
Simulador	PC 4	Bajas

Tabla 7.6: Distribución empleada en la prueba sobre el esquema distribuido síncrono de la arquitectura *DLA*

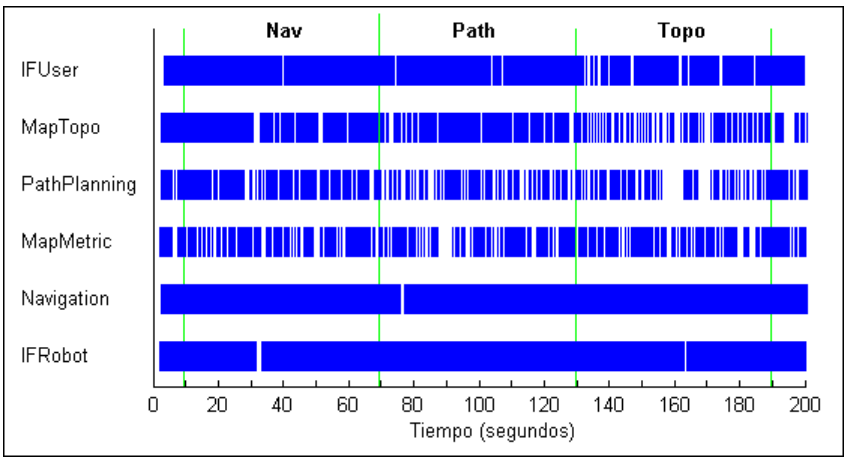

Figura 7.4: Frecuencia de ejecución de los módulos en el esquema distribuido básico de la arquitectura *DLA*

capítulo 3 esta es ejecutada únicamente cuando se detectan cambios significativos en el entorno que invaliden los planes previamente calculados. Es importante recordar que los cambios significativos se detectan según el número de celdas que sean modificadas en el mapa métrico que se posee del entorno, existiendo dos parámetros independientes para relanzar la planificación de caminos (U_{path}) y la planificación de rutas (U_{topo}). Obviamente estos parámetros controlan la frecuencia de ejecución de los módulos deliberados. La figura 7.6 muestra el efecto de reducir estos parámetros al mínimo, iniciando un nuevo ciclo de la Capa de Planificación cada vez que se detecta alguna modificación en el mapa métrico. Como se puede apreciar respecto a la figura 7.5 la Capa de Planificación tiene una mayor frecuencia de ejecución en este caso, aunque los distintos módulos siguen quedando inactivos cuando no son necesarios. Gracias a estos parámetros es posible ajustar muy fácilmente la sensibilidad y validez de los planes calculados frente a cambios en el entorno.

La figura 7.7 muestra las prestaciones obtenidas por el esquema distribuido síncrono de la arquitectura *DLA* en los distintos niveles de navegación, mientras que la tabla 7.7 muestra los valores

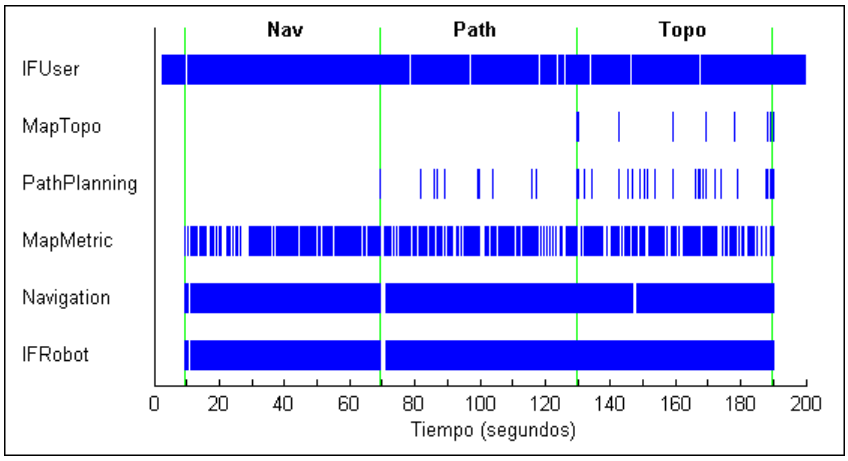

Figura 7.5: Frecuencia de ejecución de los módulos en el esquema distribuido síncrono de la arquitectura *DLA*

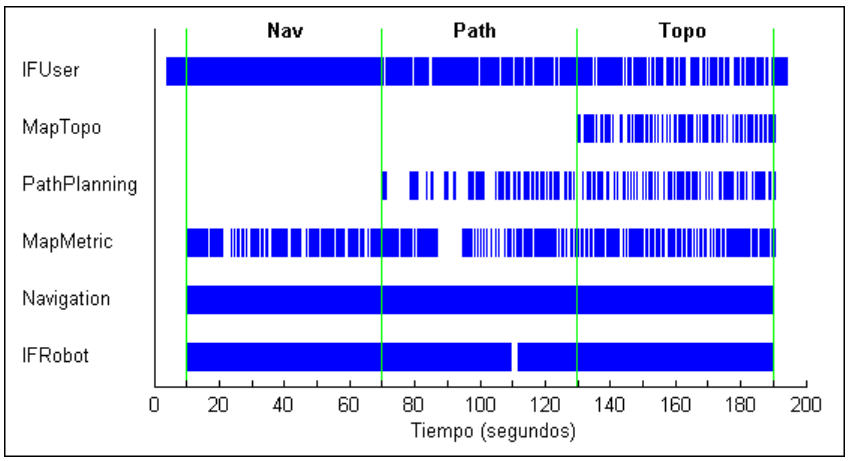

Figura 7.6: Frecuencia de ejecución de los módulos en el esquema distribuido síncrono de la arquitectura *DLA* con un tiempo de actualización menor

numéricos de la media calculada para la latencia reactiva y para la latencia total.

En el caso peor la latencia reactiva es de unos 175 ms, lo que implica que a una velocidad del agente de unos 300 mm/s se recorren aproximadamente 5 cm antes de reaccionar frente a un nuevo estímulo. En el caso peor la latencia total es de unos 1.3 s, por lo que a una velocidad del agente de unos 300 mm/s se recorren aproximadamente 39 cm antes de generar los planes adecuados ante un nuevo comando.

Comparando estos resultados con los obtenidos en el esquema distribuido básico de la figura 7.3 y de la tabla 7.3 se puede concluir que el nuevo esquema mejora ligeramente las prestaciones del sistema. Es importante recordar que la mejora obtenida en las prestaciones depende directamente de la distribución de los recursos empleados, por lo que otras distribuciones más críticas que utilicen menos recursos son susceptibles de mostrar mejoras mucho más importantes en las latencias del esquema distribuido síncrono respecto del esquema distribuido básico. Sin embargo, el esquema distribuido síncrono es evidentemente más eficiente que el esquema distribuido

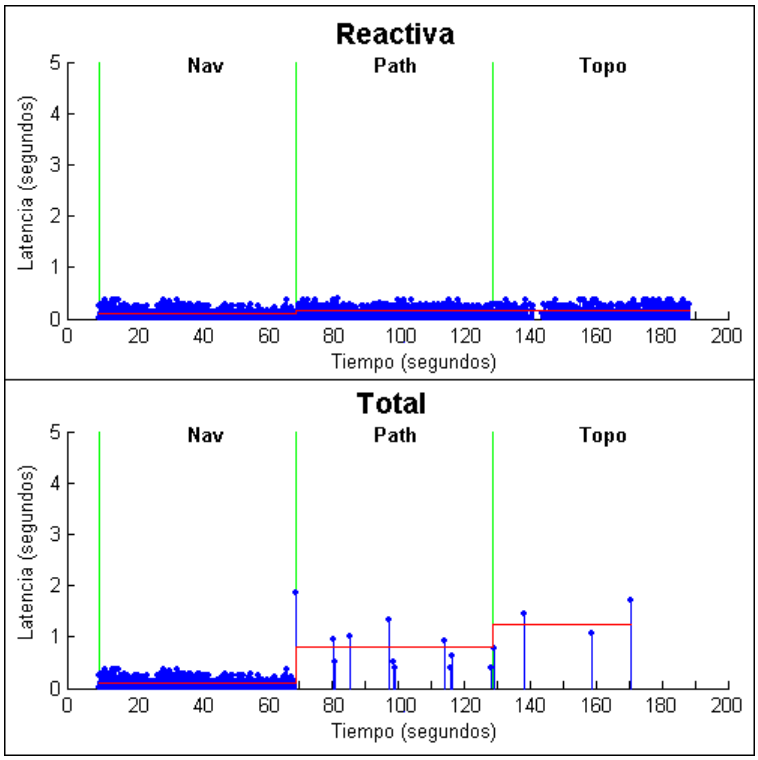

Figura 7.7: Prestaciones obtenidas por el esquema distribuido síncrono de la arquitectura *DLA*

Esquema	Latencia	Nav	Path	Topo
Distribuido síncrono	Reactiva	$98 \pm 7\ ms$	$163 \pm 9\ ms$	$175 \pm 11\ ms$
	Total	$98 \pm 7\ ms$	$800 \pm 300\ ms$	$1300 \pm 600\ ms$

Tabla 7.7: Prestaciones obtenidas por el esquema distribuido síncrono de la arquitectura *DLA*

básico al optimizar el uso de los recursos disponibles, por lo que será el empleado finalmente en el sistema real.

Por último, el esquema distribuido síncrono permite no sólo optimizar eficientemente el uso de los recursos disponibles, sino también optimizar dinámicamente la distribución de los mismos, tal como se comentó en el apartado 3.2.2 del capítulo 3. En este sentido, se ha realizado otra prueba donde los módulos son distribuidos dinámicamente en el sistema para utilizar así de forma más eficiente los recursos disponibles.

La distribución de los distintos elementos del sistema y las máquinas empleadas en esta prueba es la que se muestra en la tabla 7.8. En la situación inicial todos los módulos del sistema se encuentran ubicados en una máquina con prestaciones medias. Tras un período de tiempo los módulos *Navigation*, *PathPlanning* y *MapTopo* se redistribuyen a una máquina de altas prestaciones. Es importante recordar que no se produce una redistribución real del módulo, sino la desactivación de los módulos en una máquina y la activación en otra, tal y como se comentó en el apartado 3.2.2 del capítulo 3.

La figura 7.8 muestra las prestaciones obtenidas por el esquema distribuido síncrono de la ar-

Elemento	Máquina inicial	Prestaciones	Máquina final	Prestaciones
DLAServer	PC 2	Altas	PC 2	Altas
IFRobot	PC 3	Medias	PC 3	Medias
Navigation	PC 3	Medias	PC 1	Altas
MapMetric	PC 3	Medias	PC 3	Medias
PathPlanning	PC 3	Medias	PC 1	Altas
MapTopo	PC 3	Medias	PC 1	Altas
IFUser	PC 3	Medias	PC 3	Medias
CBRServer	PC 1	Altas	PC 1	Altas
RobotServer	PC 4	Bajas	PC 4	Bajas
Simulador	PC 4	Bajas	PC 4	Bajas

Tabla 7.8: Distribución empleada en la prueba sobre el esquema distribuido síncrono de la arquitectura *DLA* con distribución dinámica de módulos

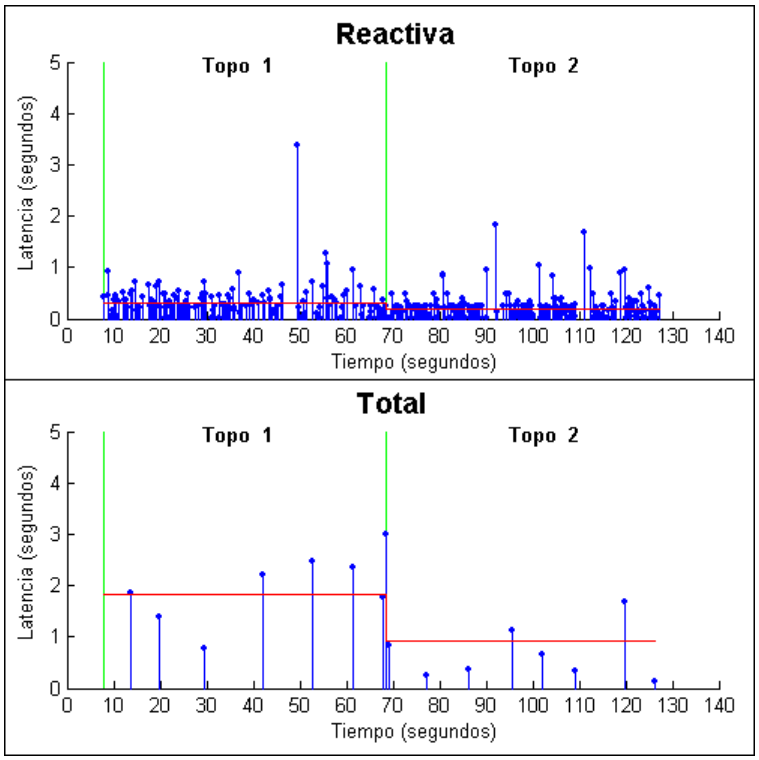

Figura 7.8: Prestaciones obtenidas por el esquema distribuido síncrono de la arquitectura *DLA* con distribución dinámica de módulos

Esquema	Latencia	Topo 1	Topo 2
Distribuido síncrono (dinámico)	Reactiva	$320 \pm 50\ ms$	$180 \pm 30\ ms$
	Total	$1800 \pm 600\ ms$	$900 \pm 700\ ms$

Tabla 7.9: Prestaciones obtenidas por el esquema distribuido síncrono de la arquitectura *DLA* con distribución dinámica de módulos

Elemento	Máquina	Prestaciones
DLAServer	PC 2	Altas
IFRobot	PC 1	Altas
Navigation	PC 1	Altas
MapMetric	PC 1	Altas
PathPlanning	PC 1	Altas
MapTopo	PC 1	Altas
IFUser	PC 1	Altas
CBRServer	PC 1	Altas
RobotServer	PC 4	Bajas
Simulador	PC 4	Bajas

Tabla 7.10: Distribución empleada en la prueba sobre el esquema distribuido síncrono de la arquitectura *DLA* en una máquina de altas prestaciones

quitectura *DLA* con distribución dinámica de módulos en en nivel de Navegación Topológica, mientras que la tabla 7.9 muestra los valores numéricos de la media calculada para la latencia reactiva y para la latencia total. Se ha etiquetado como "Topo 1" la situación inicial antes de la distribución de los módulos y como "Topo 2" la situación final después de la distribución de los módulos. Se ha realizado la prueba únicamente sobre el nivel de Navegación Topológica porque se corresponde con el caso peor donde todos los módulos del sistema están activos. Como se puede observar la latencia reactiva y la latencia total mejora considerablemente una vez realizada la distribución de los módulos entre los recursos disponibles, como era lógico esperar.

Esta última prueba abre una nueva línea de investigación consistente en la monitorización continua de la carga soportada por cada una de las máquinas disponibles en el sistema, de forma que se distribuyan automáticamente los módulos entre ellas con el fin de optimizar el uso de los recursos disponibles.

2.3 Esquema local síncrono

En algunas circunstancias es necesario ubicar todos los módulos del sistema sobre una misma máquina, como por ejemplo cuando se quiere implementar un agente autónomo móvil que opere en entornos exteriores, donde la existencia de enlaces inalámbricos está bastante limitada. El esquema local síncrono pretende optimizar el funcionamiento del sistema bajo estas circunstancias extremas, pues en esta situación donde todos los módulos se ubican en la misma máquina implementar un esquema distribuido carece totalmente de sentido, degradando significativamente las prestaciones finales del sistema al tener que interaccionar los distintos módulos a través de un agente externo que no es necesario bajo una distribución local.

La distribución de los distintos elementos del sistema y las máquinas empleadas en esta prueba es la que se muestra en la tabla 7.10. Todos los módulos del sistema se han distribuido sobre una única máquina, en este caso de altas prestaciones. Para verificar el mejor funcionamiento del esquema local síncrono frente al esquema distribuido síncrono en las circunstancias extremas

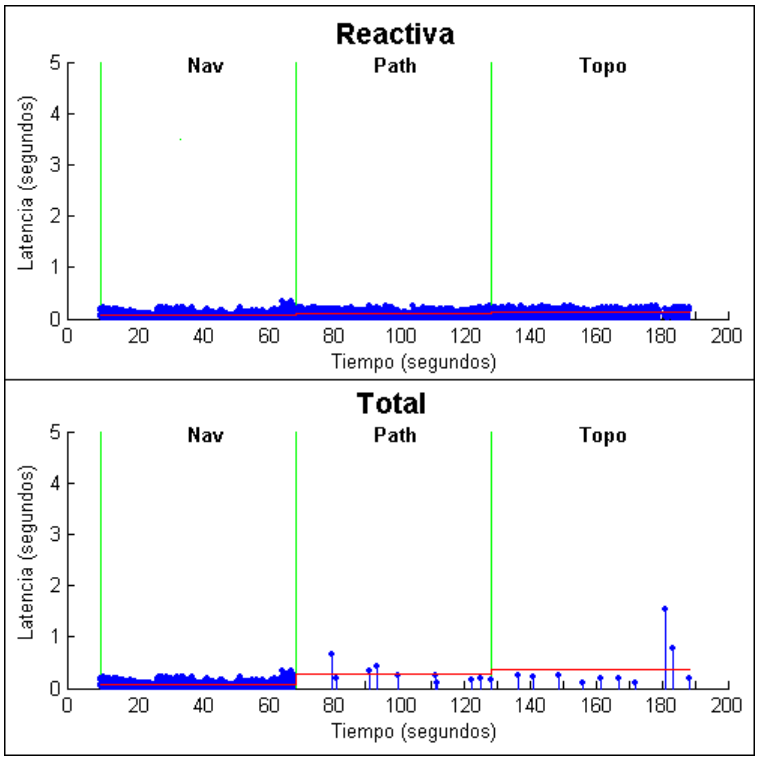

Figura 7.9: Prestaciones obtenidas por el esquema distribuido síncrono de la arquitectura *DLA* en una máquina de altas prestaciones

Esquema	Latencia	Nav	Path	Topo
Distribuido síncrono (altas prestaciones)	Reactiva	70 ± 3 ms	104 ± 4 ms	129 ± 5 ms
	Total	70 ± 3 ms	290 ± 130 ms	400 ± 300 ms

Tabla 7.11: Prestaciones obtenidas por el esquema distribuido síncrono de la arquitectura *DLA* en una máquina de altas prestaciones

citadas anteriormente, se van a realizar dos pruebas sobre la distribución mostrada en la tabla 7.10, una interconectando los módulos a través del esquema distribuido síncrono y otra interconectando los módulos a través del esquema local síncrono.

La figura 7.9 muestra las prestaciones obtenidas por el esquema distribuido síncrono de la arquitectura *DLA* en los distintos niveles de navegación, mientras que la tabla 7.11 muestra los valores numéricos de la media calculada para la latencia reactiva y para la latencia total.

En el caso peor la latencia reactiva es de unos 130 ms, lo que implica que a una velocidad del agente de unos 300 mm/s se recorren aproximadamente 4 cm antes de reaccionar frente a un nuevo estímulo. En el caso peor la latencia total es de unos 400 ms, por lo que a una velocidad del agente de unos 300 mm/s se recorren aproximadamente 12 cm antes de generar los planes adecuados ante un nuevo comando.

Comparando estos resultados con los obtenidos en el esquema distribuido síncrono de la figura 7.7 y de la tabla 7.7 se puede concluir que la nueva distribución mejora considerablemente las

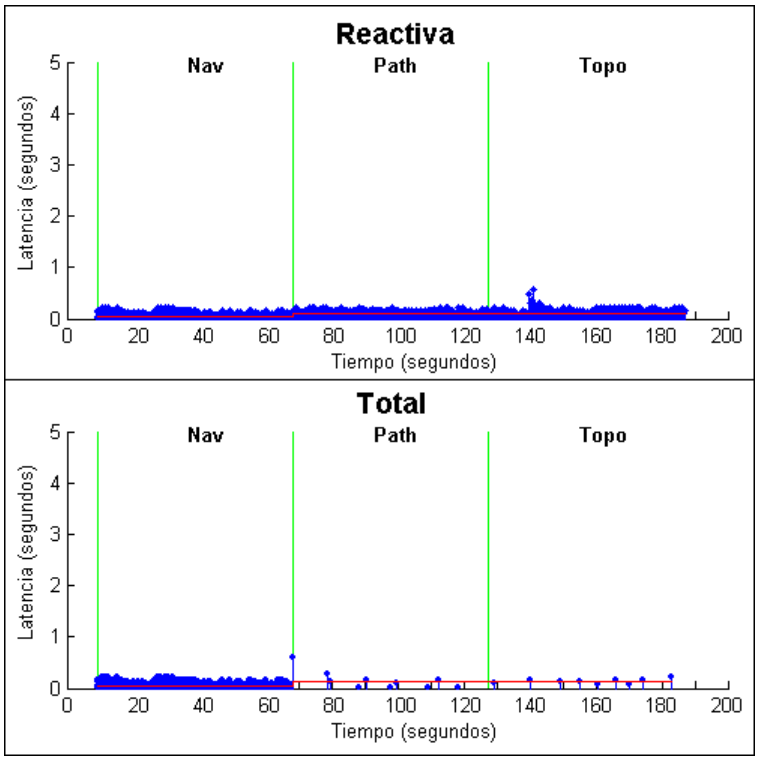

Figura 7.10: Prestaciones obtenidas por el esquema local síncrono de la arquitectura *DLA* en una máquina de altas prestaciones

Esquema	Latencia	Nav	Path	Topo
Local síncrono (altas prestaciones)	Reactiva	54 ± 3 *ms*	89 ± 4 *ms*	105 ± 6 *ms*
	Total	54 ± 3 *ms*	120 ± 80 *ms*	140 ± 40 *ms*

Tabla 7.12: Prestaciones obtenidas por el esquema local síncrono de la arquitectura *DLA* en una máquina de altas prestaciones

prestaciones del sistema. Este hecho se debe a que todos los módulos se han ubicado en una máquina de altas prestaciones, frente a la situación del esquema distribuido síncrono donde los módulos se ubicaron en varias máquinas con distintas prestaciones. Así pues, aunque todos los módulos se han ubicado en la misma máquina, los recursos disponibles del sistema han sido mejor distribuidos, por lo que las prestaciones del mismo mejoran.

Para optimizar el funcionamiento del sistema se va a emplear el esquema local síncrono en lugar del esquema distribuido síncrono sobre la misma distribución mostrada en la tabla 7.10. La figura 7.10 muestra las prestaciones obtenidas por el esquema local síncrono de la arquitectura *DLA* en los distintos niveles de navegación, mientras que la tabla 7.12 muestra los valores numéricos de la media calculada para la latencia reactiva y para la latencia total.

En el caso peor la latencia reactiva es de unos 100 *ms*, lo que implica que a una velocidad del agente de unos 300 mm/s se recorren aproximadamente 3 *cm* antes de reaccionar frente a un nuevo estímulo. En el caso peor la latencia total es de unos 140 *ms*, por lo que a una velocidad del agente de unos 300 mm/s se recorren aproximadamente 4 *cm* antes de generar los planes

Elemento	Máquina	Prestaciones
DLAServer	PC 2	Altas
IFRobot	PC 5	Bajas
Navigation	PC 5	Bajas
MapMetric	PC 5	Bajas
PathPlanning	PC 5	Bajas
MapTopo	PC 5	Bajas
IFUser	PC 5	Bajas
CBRServer	PC 1	Altas
RobotServer	PC 4	Bajas
Simulador	PC 4	Bajas

Tabla 7.13: Distribución empleada en la prueba sobre el esquema distribuido síncrono de la arquitectura *DLA* en una máquina de bajas prestaciones

adecuados ante un nuevo comando.

Comparando estos resultados con los obtenidos en el esquema distribuido síncrono de la figura 7.9 y de la tabla 7.9 se puede apreciar que la nueva distribución mejora considerablemente las prestaciones del sistema, por lo que el esquema local síncrono es el óptimo a utilizar en aquellas circunstancias extremas donde todos los módulos deban ser ubicados en la misma máquina.

Es importante recordar que las prestaciones obtenidas dependen directamente de los recursos empleados. Si se repite la prueba anterior pero utilizando una máquina de bajas prestaciones en lugar de una máquina de altas prestaciones para distribuir los módulos la mejora obtenida en las prestaciones es mucho más significativa. La distribución de los distintos elementos del sistema y las máquinas empleadas en esta prueba es la que se muestra en la tabla 7.13. Todos los módulos del sistema se han distribuido sobre una única máquina en este caso de bajas prestaciones.

Bajo estas circunstancias, la figura 7.11 muestra las prestaciones obtenidas por el esquema distribuido síncrono de la arquitectura *DLA* en los distintos niveles de navegación, mientras que la tabla 7.14 muestra los valores numéricos de la media calculada para la latencia reactiva y para la latencia total.

En el caso peor la latencia reactiva es de unos 570 ms, lo que implica que a una velocidad del agente de unos 300 mm/s se recorren aproximadamente 17 cm antes de reaccionar frente a un nuevo estímulo. En el caso peor la latencia total es de unos 2.7 s, por lo que a una velocidad del agente de unos 300 mm/s se recorren aproximadamente 81 cm antes de generar los planes adecuados ante un nuevo comando.

Estos resultados ponen de manifiesto que no siempre es mejor distribuir todos los módulos del sistema en una misma máquina, dependiendo obviamente de las prestaciones de la máquina utilizada. Comparando estos resultados con los obtenidos en el esquema distribuido síncrono de la figura 7.7 y de la tabla 7.7 se observa que la nueva distribución empeora considerablemente las prestaciones del sistema. Este hecho se debe a que ahora todos los módulos se han ubicado en una máquina de bajas prestaciones, frente a la situación del esquema distribuido síncrono

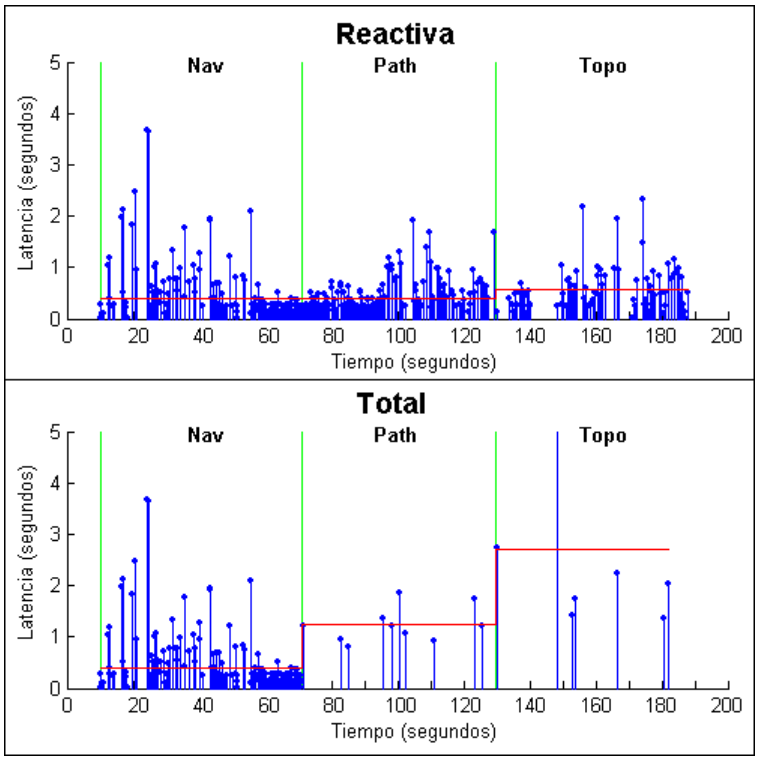

Figura 7.11: Prestaciones obtenidas por el esquema distribuido síncrono de la arquitectura *DLA* en una máquina de bajas prestaciones

Esquema	Latencia	Nav	Path	Topo
Distribuido síncrono (bajas prestaciones)	Reactiva	$390 \pm 80\ ms$	$400 \pm 50\ ms$	$570 \pm 90\ ms$
	Total	$390 \pm 80\ ms$	$1230 \pm 240\ ms$	$2700 \pm 2000\ ms$

Tabla 7.14: Prestaciones obtenidas por el esquema distribuido síncrono de la arquitectura *DLA* en una máquina de bajas prestaciones

donde los módulos se ubicaron en varias máquinas con distintas prestaciones. Puesto que los recursos disponibles han sido peor distribuidos las prestaciones del sistema empeoran.

Aun en esta situación tan desfavorable, se va a optimizar el funcionamiento del sistema empleando de nuevo el esquema local síncrono en lugar del esquema distribuido síncrono sobre la misma distribución mostrada en la tabla 7.13. La figura 7.12 muestra las prestaciones obtenidas por el esquema local síncrono de la arquitectura *DLA* en los distintos niveles de navegación, mientras que la tabla 7.15 muestra los valores numéricos de la media calculada para la latencia reactiva y para la latencia total.

En el caso peor la latencia reactiva es de unos 165 ms, lo que implica que a una velocidad del agente de unos 300 mm/s se recorren aproximadamente 5 cm antes de reaccionar frente a un nuevo estímulo. En el caso peor la latencia total es de unos 240 ms, por lo que a una velocidad del agente de unos 300 mm/s se recorren aproximadamente 7 cm antes de generar los planes adecuados ante un nuevo comando.

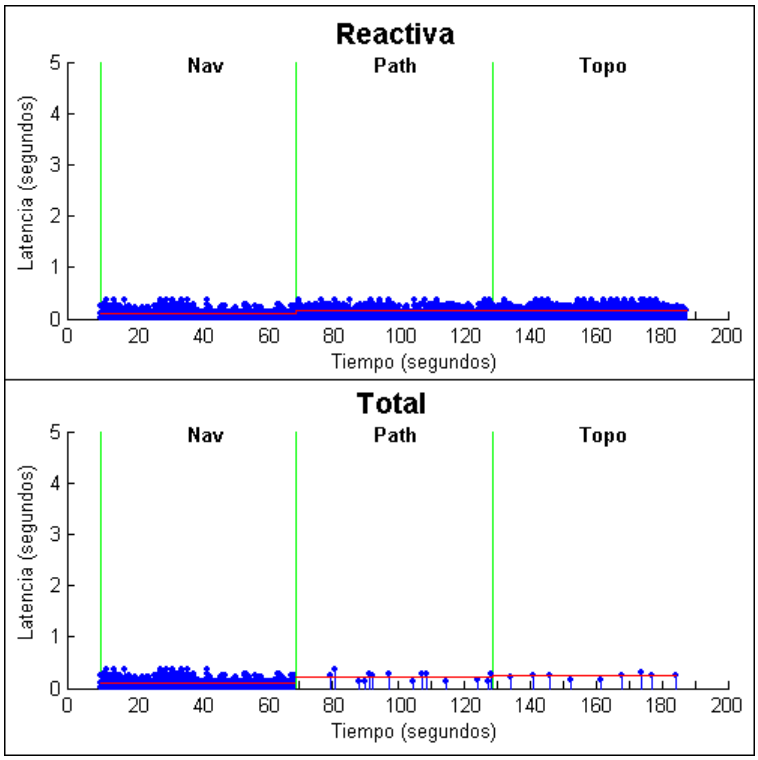

Figura 7.12: Prestaciones obtenidas por el esquema local síncrono de la arquitectura *DLA* en una máquina de bajas prestaciones

Esquema	**Latencia**	**Nav**	**Path**	**Topo**
Local síncrono (bajas prestaciones)	Reactiva	$94 \pm 7\ ms$	$159 \pm 10\ ms$	$165 \pm 11\ ms$
	Total	$94 \pm 7\ ms$	$220 \pm 40\ ms$	$240 \pm 40\ ms$

Tabla 7.15: Prestaciones obtenidas por el esquema local síncrono de la arquitectura *DLA* en una máquina de bajas prestaciones

Estos tiempos son muy superiores a los obtenidos bajo la misma distribución mediante el esquema distribuido síncrono, por lo que se corrobora el hecho de que el esquema local síncrono es el óptimo a utilizar en las situaciones extremas donde todos los módulos deban ser ubicados en la misma máquina.

Si se comparan las prestaciones obtenidas por el esquema distribuido síncrono (módulos distribuidos a lo largo de distintas máquinas, figura 7.7 y tabla 7.7) con las prestaciones obtenidas por el esquema local síncrono (todos los módulos ubicados en la misma máquina, figura 7.12 y tabla 7.15) puede parecer que el esquema local síncrono es preferible aun en el caso de utilizar una máquina de bajas prestaciones. Es importante matizar que la mejora de las prestaciones en tales circunstancias depende enormemente del sistema implementado y de los recursos disponibles. En sistemas donde el número de módulos sea más elevado puede que una perspectiva local presente peores prestaciones que una perspectiva distribuida. Además, la perspectiva distribuida puede ser necesaria si se pretenden utilizar distintas plataformas en el desarrollo de los módulos, como puede ser *Linux* y *Windows*.

Es por ello por lo que se han mantenido los distintos esquemas implementados en la arquitectura *DLA*, ya que el esquema distribuido síncrono optimiza el funcionamiento en sistemas distribuidos mientras que el esquema local síncrono optimiza el funcionamiento en sistema locales.

3 Estructura de la arquitectura *DLA*

En este apartado se pretende analizar la capacidad de navegación obtenida en el sistema mediante la estructura definida en el mismo. Para ello se van a realizar diferentes pruebas sobre los distintos niveles de navegación implementados para desarrollar la Jerarquía de Navegación: Navegación Reactiva, Navegación Planificada y Navegación Topológica. Se analizan a continuación los resultados obtenidos en la realización de estas pruebas.

3.1 Navegación Reactiva

El nivel de Navegación Reactiva constituye el nivel de navegación básica del sistema, y es responsable de navegar evitando la colisión con los obstáculos fijos y/o móviles del entorno hasta alcanzar el destino final. Este nivel, pues, debe garantizar la integridad del sistema evitando situaciones de riesgo y reaccionando adecuadamente ante situaciones inesperadas. Según la Jerarquía de Navegación descrita en el apartado 2 del capítulo 1 este nivel de navegación se corresponde con el grupo de Navegación Local, el cual desarrolla los niveles más básicos de navegación: Búsqueda, Seguimiento, Apuntado y Guiado.

El nivel de Navegación Reactiva es desarrollado por el esquema *CBR* propuesto en el apartado 2 del capítulo 5. Para comprobar el correcto funcionamiento del sistema se van a desarrollar diferentes pruebas sobre dicho esquema: pruebas en entornos simulados, pruebas en entornos reales, pruebas en entornos mixtos y pruebas en entornos con obstáculos móviles. En las distintas pruebas realizadas se han empleado mapas métricos para mostrar los resultados obtenidos, si bien es importante destacar que dichos mapas no han sido utilizados por la Capa de Navegación. En estos mapas las zonas libres se muestran en negro, las zonas ocupadas en blanco y las zonas no exploradas en gris.

3.1.1 Entornos simulados

La utilización de entornos simulados es una práctica habitual en el desarrollo de sistemas complejos, no sólo por facilidad de depuración sino también por seguridad y por repetitividad de las pruebas. Una vez el sistema ha sido satisfactoriamente analizado bajo un entorno simulado se procede a la verificación final de su funcionamiento en el campo de aplicación real donde va a desarrollar su función.

El comportamiento del sistema en entornos simulados ya ha sido profundamente analizado al realizar el diseño del esquema *CBR*, donde se ha utilizado un simulador para poder analizar

con mayor detalle los resultados obtenidos y para garantizar la repetitividad de las diferentes pruebas. En concreto, en el apartado 2 del capítulo 5 se han realizado las siguientes pruebas:

- **Aprendizaje por observación:** se han realizado diversas pruebas para determinar la influencia del aprendizaje por observación en la operación del sistema, realizando pruebas con entrenamiento mediante Campos Potenciales (figura 5.7), pruebas con entrenamiento humano (figura 5.9) y pruebas con entrenamiento mixto (figura 5.10).
- **Aprendizaje por experiencia:** se han realizado diversas pruebas para determinar la influencia del aprendizaje por experiencia en la operación del sistema, realizando pruebas sin utilizar ningún tipo de entrenamiento previo (figura 5.11) y pruebas tras un proceso de entrenamiento inicial (figura 5.13).
- **Optimización de la base de casos:** se han realizado diversas pruebas para determinar la influencia del proceso de optimización de la base de casos en la operación del sistema, realizando pruebas en función del tamaño final de la base de casos optimizada (figura 5.15) y pruebas en función de los distintos parámetros de eficiencia del sistema (figuras 5.16 y 5.17).

Se considera, pues, que el comportamiento básico del sistema ya ha sido lo suficientemente verificado en entornos simulados, por lo que no se va a realizar ninguna prueba adicional y se remite al lector a los resultados obtenidos en el citado apartado 2 del capítulo 5.

3.1.2 Entornos reales

Tras comprobar el correcto funcionamiento del sistema en un entorno simulado es necesario proceder a la verificación del mismo en el entorno final donde va a desarrollar su actividad. Este tipo de pruebas son importantes para determinar la influencia que tienen sobre el sistema los posibles errores que caracterizan al mundo real, errores no presentes en entornos simulados.

Una de las fuentes de error más comunes en aplicaciones que utilizan sensores sonar se encuentra en las frecuentes lecturas erróneas que caracterizan la operación de dichos sensores. De hecho, en los entornos reales suelen coexistir una gran cantidad de materiales distintos (paredes, madera, cartón, metal, ...), cada uno de los cuales exhibe un comportamiento típico ante el haz de ultrasonidos del sensor sonar. Aquellos materiales con menores coeficientes de reflexión producen lecturas sonar más ruidosas, y dependiendo del ángulo de incidencia del haz incluso pueden producir lecturas erróneas, apareciendo medidas espúreas que detecten falsos obstáculos o que no detecten obstáculos existentes. Es por ello por lo que es necesario medir la sensibilidad del sistema ante este tipo de situaciones.

Prueba 1: Operación en entornos reales

La primera prueba trata simplemente de verificar el funcionamiento básico de la Capa de Navegación en entornos reales. Para ello se va a realizar un sencillo entrenamiento mediante apren-

Figura 7.13: Entrenamiento mediante varios operadores humanos de la Capa de Navegación.

dizaje por observación de algunas situaciones sencillas, verificando posteriormente si la operación del sistema es correcta.

La figura 7.13 muestra la situación utilizada para entrenar al sistema *CBR* de la Capa de Navegación. El entrenamiento consiste en guiar al agente a través de un obstáculo tanto por la izquierda como por la derecha, situación repetida por tres operadores humanos distintos. Durante el proceso de entrenamiento no se impuso ningún tipo de restricción a los recorridos realizados, por lo que los distintos operadores humanos podían seguir trayectorias poco óptimas si lo deseaban, con largos recorridos, oscilaciones, proximidad a los obstáculos y cambios bruscos de dirección.

Una vez finalizado el entrenamiento se realizó el proceso de optimización de la base de casos sobre las seis rutas de entrenamiento realizadas, obteniendo 159 casos en la base de casos final. Durante es proceso de optimización se fijó la misma importancia para cada uno de los criterios de evaluación: suavidad, distancia y seguridad (K_{soft}=K_{dist}=K_{sec}=1 en la expresión 5.9 del capítulo 5). No se realizó un entrenamiento más exhaustivo con mayor número de obstáculos para comprobar la operación del sistema ante situaciones desconocidas.

Seguidamente se configuró un entorno de prueba sencillo con dos obstáculos bastante separados entre sí y se comprobó si el agente era capaz de alcanzar el destino final. La figura 7.14 muestra el entorno utilizado en esta prueba, así como la posición actual del agente, el destino final (marcado con una "X") y la trayectoria seguida por el mismo. En esta figura la posición de los distintos obstáculos ha sido añadida artificialmente para proporcionar una mayor claridad en la exposición de las pruebas.

El análisis detallado de esta prueba corrobora algunas conclusiones interesantes. Inicialmente el agente se dirige en línea recta hacia el destino final, pero en su recorrido se encuentra un primer obstáculo. Debido al proceso de entrenamiento esta situación con un obstáculo es similar a la que posee el sistema en su base de casos, por lo que utilizando esta experiencia el agente comienza a girar suavemente para esquivar el obstáculo detectado. Tras avanzar un poco detecta un nuevo obstáculo, apareciendo pues dos obstáculos en la zona de influencia del agente. Esta situación con dos obstáculos no aparece contemplada en el proceso de entrenamiento, por lo que constituye una situación desconocida y el sistema debe generar nuevos casos adaptados para resolverla. Durante este proceso de adaptación se producen algunos cambios de dirección un

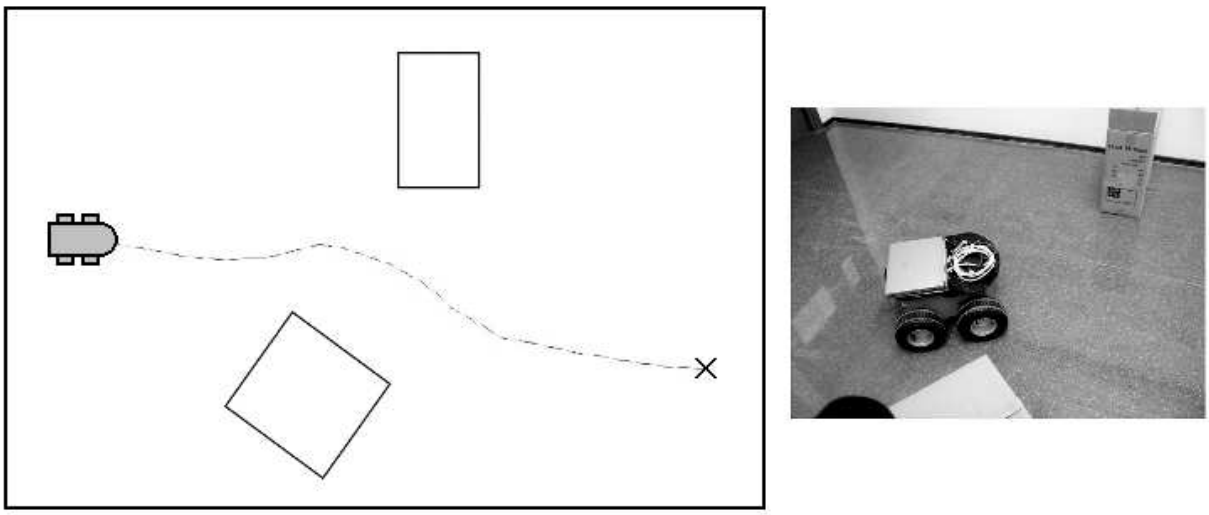

Figura 7.14: Operación en un entorno real con dos obstáculos separados.

tanto bruscos. Mientras el agente sigue avanzando entre los dos obstáculos va generando nuevos casos adaptados, necesarios para resolver esta situación desconocida. Tras seguir avanzando uno de los obstáculos desaparece del área de influencia del agente, por lo que se retorna a la situación donde únicamente se detecta un obstáculo, situación de nuevo conocida del proceso de entrenamiento. Así pues, el agente vuelve a girar suavemente hasta que finalmente se supera y se puede encaminar en línea recta hacia el destino final.

Durante la realización de este prueba el sistema generó numerosos casos adaptados, correspondientes a la situación donde confluían dos obstáculos dentro de la zona de influencia del agente. Estos nuevos casos adaptados fueron incorporados en el sistema como aprendizaje por experiencia, de forma que para futuras pruebas fuese conocida la situación de atravesar dos obstáculos. Tras utilizar el proceso de optimización de la base de casos con este nuevo conocimiento derivado de los casos adaptados el sistema incorporó 42 nuevos casos.

Prueba 2: Comparación con los Campos Potenciales

Esta segunda prueba trata de comparar el funcionamiento del esquema *CBR* propuesto con el esquema de los Campos Potenciales en entornos reales. Para ello se va a forzar una situación que no pueda ser resuelta mediante los Campos Potenciales y se va a aplicar el sistema *CBR* para verificar si dicha situación es resuelta satisfactoriamente.

Tras realizar la prueba de la figura 7.14 y almacenar la nueva experiencia adquirida mediante el proceso de aprendizaje por experiencia se configuró un entorno más complejo, ubicando dos obstáculos muy próximos entre sí. Este entorno fue diseñado para que el esquema de los Campos Potenciales descrito en el apartado 1.1.1 del capítulo 5 no fuese capaz de superar satisfactoriamente dicha situación, para lo cual se fue reduciendo progresivamente la separación entre ambos obstáculos hasta que dicho esquema no pudo atravesarlos. Recuérdese que la imposibilidad de

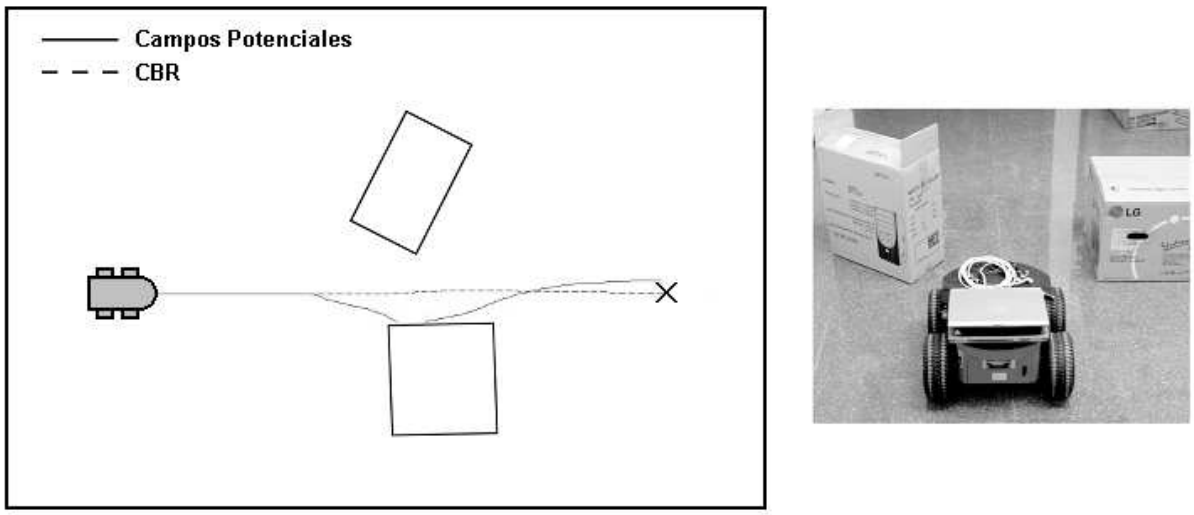

Figura 7.15: Operación en un entorno real con dos obstáculos cercanos.

atravesar obstáculos cercanos es uno de los típicos inconvenientes del esquema de los Campos Potenciales. Sin embargo, para permitir que el agente atravesase estos obstáculos se ajustaron los parámetros de los Campos Potenciales reduciendo considerablemente la fuerza de repulsión que generaban los obstáculos del entorno. Esta situación no es deseada, pues permite que el agente se aproxime más a los obstáculos aumentando el riesgo de colisión. No obstante, se permitió por motivos de comparación con el esquema *CBR* propuesto. Es preciso indicar que para realizar esta prueba se tuvo que deshabilitar el sistema de alarma del módulo *IFRobot*, pues si no el agente se hubiese detenido cuando la distancia a cualquier obstáculo se encontrase por debajo del umbral de seguridad U_{sec}, tal y como se ha descrito en el apartado 1.1 del capítulo 4.

La figura 7.15 muestra el entorno utilizado en esta prueba, así como la posición actual del agente, el destino final (marcado con una "X") y la trayectoria seguida por el mismo tanto para el esquema *CBR* como para el esquema de los Campos Potenciales. Como se puede apreciar, el destino final no coincide exactamente en ambas trayectorias, debido al problema de la repetitividad de las pruebas en entornos reales. En esta figura la posición de los distintos obstáculos ha sido añadida artificialmente para proporcionar una mayor claridad en la exposición de las pruebas.

Los resultados de esta prueba ponen de manifiesto que mientras el esquema basado en Campos Potenciales ha producido una colisión con los obstáculos del entorno, el comportamiento del esquema *CBR* ha sido plenamente satisfactorio. Este hecho se debe a que la experiencia acumulada en el sistema contemplaba situaciones con dos obstáculos, que fueron las adquiridas mientras se realizaba la prueba 1 (figura 7.14). Por lo tanto, gracias a esta experiencia el agente realizó pequeños cambios de dirección con una trayectoria final bastante suave. Es necesario recordar que la colisión obtenida por el esquema de los Campos Potenciales ha sido forzada intencionadamente al reducir excesivamente la fuerza de repulsión generada por los obstáculos del entorno, y se ha permitido para mostrar las ventajas del esquema *CBR* propuesto. En realidad, ajustando convenientemente los parámetros del esquema de los Campos Potenciales esta colisión

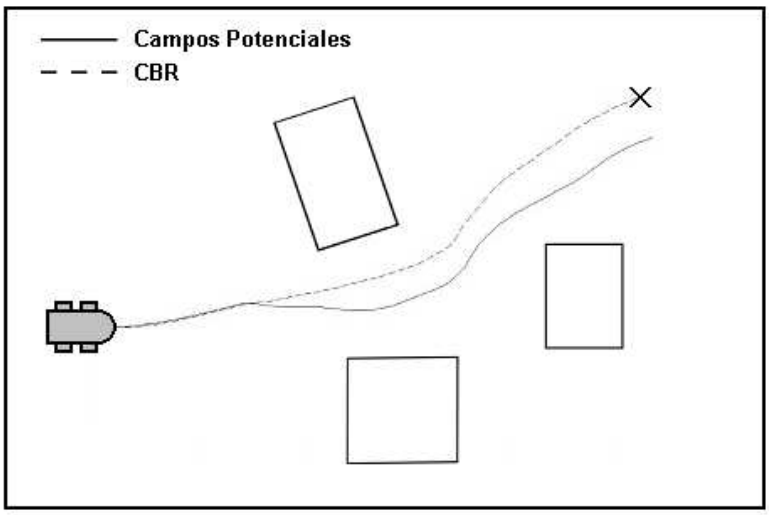

Figura 7.16: Operación en un entorno real con tres obstáculos.

no se hubiese producido, si bien es cierto que tampoco se hubiesen atravesado los dos obstáculos.

Prueba 3: Operación en entornos complejos

Aunque tanto la prueba 1 (figura 7.14) como la prueba 2 (figura 7.15) han mostrado la correcta operación del sistema, los entornos utilizados eran relativamente sencillos, con únicamente dos obstáculos. Se quiere comprobar la operación del sistema ante situaciones más complejas de resolver, para lo cual se va a diseñar un entorno algo más complejo.

La figura 7.16 muestra el entorno utilizado en esta prueba, así como la posición actual del agente, el destino final (marcado con una "X") y la trayectoria seguida por el mismo tanto para el esquema *CBR* como para el esquema de los Campos Potenciales. Se puede observar que el destino final no coincide exactamente en ambas trayectorias, debido al problema de la repetitividad de las pruebas en entornos reales. En esta figura la posición de los distintos obstáculos ha sido añadida artificialmente para proporcionar una mayor claridad en la exposición de las pruebas. Si bien el agente posee experiencia suficiente para manejar situaciones con dos obstáculos, este nuevo escenario vuelve a plantear una configuración nueva y desconocida. Puesto que los obstáculos en esta prueba están relativamente separados, se ha vuelto a ajustar correctamente el esquema de los Campos Potenciales para evitar posibles situaciones de colisión, aumentando para ello la fuerza de repulsión generada por los obstáculos del entorno. También se ha vuelto a habilitar el sistema de alarma del módulo *IFRobot*.

La trayectoria generada mediante el esquema *CBR* es más suave y directa que la trayectoria generada con el esquema de los Campos Potenciales. Este hecho se debe a que el esquema *CBR* permite una navegación más próxima a los obstáculos del entorno si dicha situación está contenida en la experiencia que posee el sistema, lo que permite que los dos primeros obstáculos sean atravesados siguiendo una trayectoria más suave gracias a dicha experiencia. Los cambios

bruscos de dirección que se producen en la trayectoria con el esquema *CBR* se deben a la necesaria adaptación de casos generada al enfrentarse el sistema con situaciones desconocidas, circunstancia que puede ser mejorada con un mayor nivel de entrenamiento.

A la vista de los resultados obtenidos con la realización de estas pruebas se puede concluir que la operación del esquema *CBR* en entornos reales sigue siendo satisfactoria, tanto para situaciones conocidas como para situaciones desconocidas. Su funcionamiento además presenta una gran flexibilidad al permitir un entrenamiento concreto de diferentes situaciones de navegación, adquiriendo una mayor experiencia progresivamente y mostrando mejores prestaciones que el esquema de los Campos Potenciales. Por último, remarcar que los resultados obtenidos ponen de manifiesto que los posibles errores en las medidas de los sensores sonar son adecuadamente gestionados por el sistema, debido en parte a que estos errores se encuentran presentes en el propio proceso de entrenamiento, por lo que son convenientemente absorbidos por la experiencia del sistema.

3.1.3 Entornos mixtos

El esquema *CBR* utilizado para implementar la Capa de Navegación mejora sus prestaciones conforme aumenta el conocimiento adquirido, ya que al aumentar la experiencia se reducen el número de situaciones desconocidas. Este conocimiento es incorporado en el sistema mediante un sistema de aprendizaje dividido en dos niveles: aprendizaje por observación tras un proceso de entrenamiento y aprendizaje por experiencia tras añadir los nuevos casos adaptados.

Si bien el aprendizaje por experiencia se deriva de la propia operación del sistema, un adecuado conocimiento inicial derivado de un aprendizaje por observación puede mejorar enormemente el funcionamiento del esquema *CBR*. Es este sentido, la utilización de un simulador puede simplificar enormemente la adquisición de este conocimiento inicial, pues resulta mucho más sencillo realizar un aprendizaje controlado de distintas configuraciones básicas bajo simulación que en un entorno real, donde suele ser complejo controlar todos los obstáculos que pueden influir en el sistema o disponer de determinados entornos o configuraciones.

Si bien el empleo de un simulador puede incorporar mayor flexibilidad en el sistema, no es evidente que se garantice su correcta operación, debido a los frecuentes errores de las lecturas sonar en un entorno real. En un aprendizaje basado en entornos reales estos errores están presentes y pueden ser convenientemente absorbidos y almacenados en el conocimiento del sistema. Sin embargo, en un aprendizaje en entornos simulados estos errores no aparecen, por lo que estas circunstancias no son almacenadas en la experiencia del agente y su posterior aparición en un entorno real podrían ocasionar funcionamientos incorrectos en el sistema.

Con el fin de verificar la capacidad de operación del sistema en entornos reales tras un proceso de aprendizaje simulado se va a realizar otro conjunto de pruebas. Para facilitar la exposición de los resultados se han utilizado mapas métricos, si bien es importante recordar que no son empleados durante el proceso de navegación. Sobre dichos mapas métricos se ha superpuesto la trayectoria seguida por el agente, marcando con círculos grises las posiciones donde ha sido

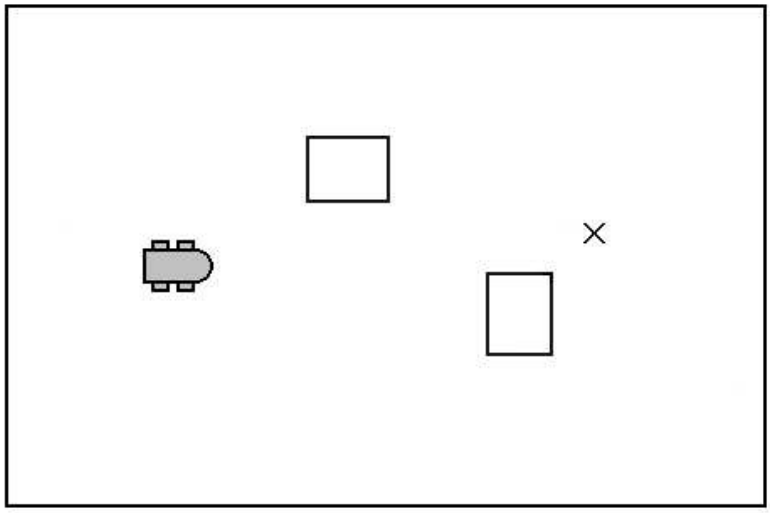

Figura 7.17: Entornos real con dos obstáculos cercanos.

utilizado el sistema *CBR* y con círculos blancos las posiciones donde los casos devueltos han tenido que ser adaptados.

Prueba 1: Operación en entornos reales con entrenamiento simulado

La primera prueba trata simplemente de verificar el funcionamiento básico de la Capa de Navegación en entornos reales tras un proceso de entrenamiento simulado. Para ello se va a realizar un sencillo proceso de aprendizaje por observación de algunas situaciones sencillas, verificando posteriormente si la operación del sistema es correcta.

La figura 7.17 muestra el entorno utilizado en esta prueba, compuesto por dos obstáculos cercanos, indicando también la posición actual del agente y el destino final (marcado con una "X").

En primer lugar se va a realizar un proceso de entrenamiento simulado mediante el esquema de los Campos Potenciales, según muestra la figura 7.18. De todas estas trayectorias únicamente la trayectoria de la figura 7.18.a se ha utilizado para adquirir los casos significativos necesarios para materializar el aprendizaje por observación.

Una vez realizado el proceso de aprendizaje por observación se va a completar el conocimiento del sistema mediante una etapa adicional de aprendizaje por experiencia. Para ello se realizan distintas trayectorias a lo largo de los entornos simulados de la figura 7.19, mientras se adquieren los nuevos casos adaptados que también formarán parte del conocimiento inicial del sistema.

Tras adquirir todos los casos de interés se realiza el proceso de optimización de la base de casos para obtener el conocimiento de partida del sistema. Es importante notar que todo este conocimiento ha sido derivado por completo del aprendizaje realizado en un entorno simulado, basado en los Campos Potenciales.

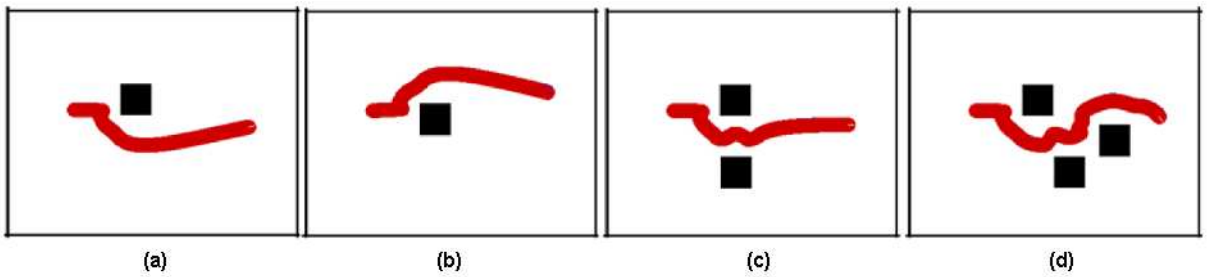

Figura 7.18: Aprendizaje por observación a partir del esquema de los Campos Potenciales: a) un obstáculo a la izquierda; b) un obstáculo a la derecha; c) dos obstáculos; d) tres obstáculos.

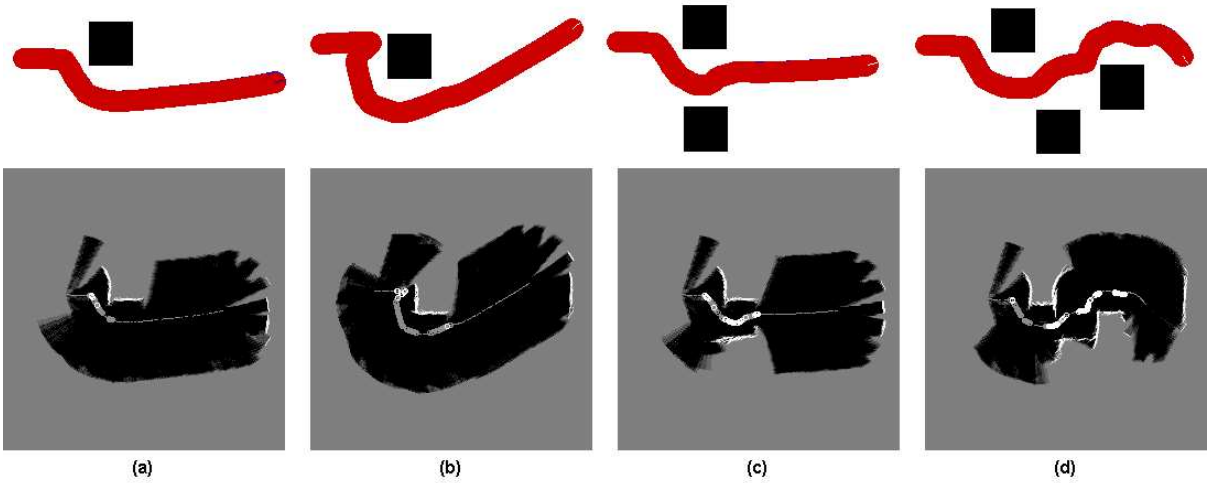

Figura 7.19: Operación del sistema *CBR* con aprendizaje por observación a partir del esquema de los Campos Potenciales: a) un obstáculo a la izquierda; b) un obstáculo a la derecha; c) dos obstáculos; d) tres obstáculos.

El resultado del proceso de navegación llevado a cabo por el agente se muestra en la figura 7.20, donde se puede apreciar en primer lugar que los mapas métricos obtenidos en entornos reales son bastante más difusos que los mapas métricos obtenidos en entornos simulados, debido a los distintos errores en las lecturas sonar. Para facilitar la exposición de los resultados se ha representado en otra imagen la posición de los obstáculos con la trayectoria final recorrida durante la prueba. El agente reproduce los casos almacenados en la base de casos (círculos grises) hasta que encuentra una nueva situación no contenida en la experiencia del sistema (círculos blancos), que se corresponde con la zona comprendida entre ambos obstáculos. En dicha zona se generan la mayoría de nuevos casos adaptados.

Se obtienen resultados similares si en lugar de utilizar un entrenamiento basado en los Campos Potenciales se utiliza un entrenamiento basado en un operador humano. Para ello se realiza el proceso de entrenamiento que muestra la figura 7.21. De todas estas trayectorias únicamente las trayectorias de las figuras 7.21.a y 7.21.b se han utilizado para adquirir los casos significativos necesarios para materializar el aprendizaje por observación.

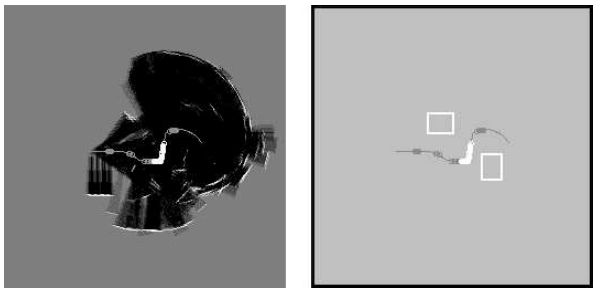

Figura 7.20: Operación en un entorno real con aprendizaje simulado basado en los Campos Potenciales.

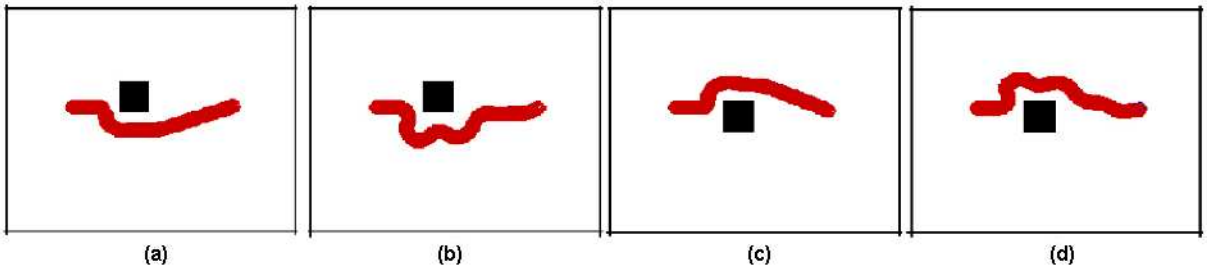

Figura 7.21: Aprendizaje por observación a partir de un operador humano: a-b) un obstáculo a la izquierda; c-d) un obstáculo a la derecha.

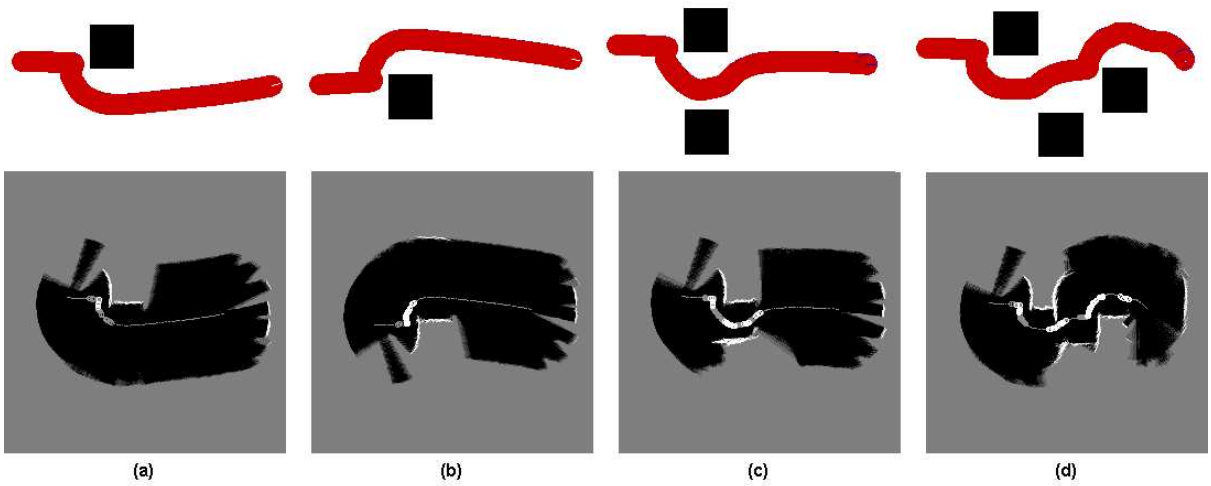

Figura 7.22: Operación del sistema *CBR* con aprendizaje por observación a partir de un operador humano: a) un obstáculo a la izquierda; b) un obstáculo a la derecha; c) dos obstáculos; d) tres obstáculos.

Una vez realizado este entrenamiento se describen algunas trayectorias en el entorno simulado de la figura 7.22, mientras se adquieren los nuevos casos adaptados para incorporar en el sistema una etapa de aprendizaje por experiencia.

Tras adquirir todos los casos se realiza el proceso de optimización de la base de casos para obtener el conocimiento de partida del sistema. Es importante notar de nuevo que todo este

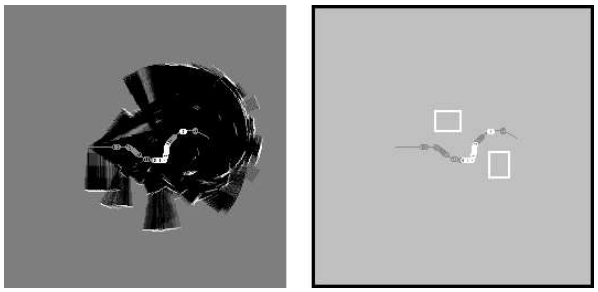

Figura 7.23: Operación en un entorno real con aprendizaje simulado basado en un operador humano.

conocimiento ha sido extraído del aprendizaje realizado en un entorno simulado, basado en este caso en un operador humano.

El resultado del proceso de navegación en este caso se muestra en la figura 7.23. Los resultados obtenidos son muy similares a los mostrados en la figura 7.20, demostrando así mismo el correcto funcionamiento del sistema. De nuevo el agente reproduce los casos almacenados en la base de casos (círculos grises) hasta que encuentra una nueva situación no contenida en la experiencia del sistema (círculos blancos), que se corresponde con la zona comprendida entre ambos obstáculos, generando en dicha zona la mayoría de nuevos casos adaptados.

La prueba mostrada en la figura 7.23 pone de manifiesto un efecto interesante, pues en la zona final del recorrido se adaptaron algunos casos sin necesidad aparente de ello. Del análisis detallado de los resultados de esta prueba se pudo constatar que estas adaptaciones se debieron a algunos errores de las lecturas sonar, ocasionando la generación de nuevos casos adaptados aun cuando el conocimiento del sistema es similar a la situación actual. De hecho, extendiendo este análisis detallado también a la prueba mostrada en la figura 7.20 permitió observar que a lo largo de las trayectorias realizadas en ambas pruebas se presentaron situaciones donde los errores de las lecturas sonar ocasionaban la no detección de algunos obstáculos del entorno.

Estos son los tipos de errores presentes en entornos reales que no se manifiestan en los entornos simulados. Sin embargo, estas pruebas ponen de manifiesto que aun en estas circunstancias la operación del sistema es correcta. Este hecho se debe a la ejecución continua del sistema de navegación reactiva, lo que permite filtrar estos errores esporádicos al tratarse de casos aislados que no duran el tiempo suficiente para inducir cambios significativos en la trayectoria del agente.

Prueba 2: Operación en entornos complejos

Si bien la prueba 1 (figuras 7.20 y 7.23) ha puesto de manifiesto la correcta operación del sistema en entornos reales con aprendizaje simulado, las situaciones que se presentaron eran bastante sencillas de resolver. En esta prueba se quiere verificar la operación del sistema ante situaciones reales algo más complejas desde el punto de vista de la navegación.

La figura 7.24 muestra el entorno utilizado en esta prueba, compuesta por un entorno en forma de pasillo ancho, indicando también la posición actual del agente y el destino final (marcado con una "X").

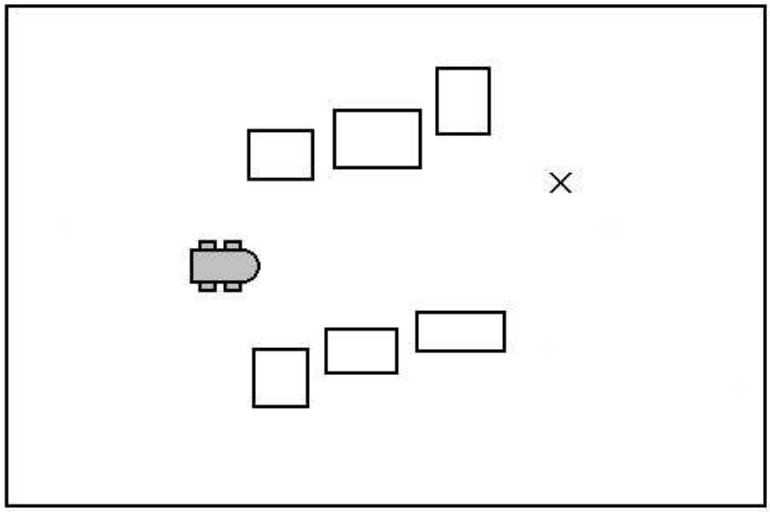

Figura 7.24: Entorno real con un pasillo ancho.

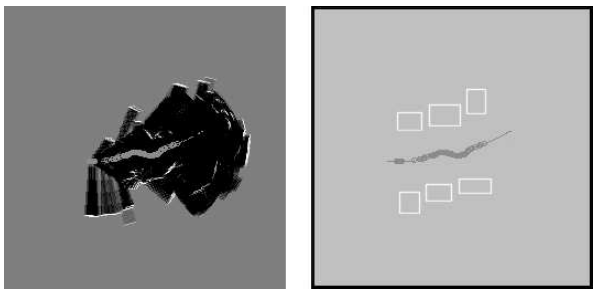

Figura 7.25: Operación en un entorno real con un pasillo ancho.

El entrenamiento inicial de esta prueba va a consistir en la trayectoria realizada en la figura 7.18.a y en las trayectorias realizadas en las figuras 7.21.a y 7.21.b. Tras este entrenamiento se realizan las trayectorias de las figuras 7.19 y 7.22 respectivamente, adquiriendo los nuevos casos adaptados durante este proceso de navegación. Seguidamente se realiza el proceso de optimización de la base de casos para formar el conocimiento inicial del sistema.

El resultado del proceso de navegación se muestra en la figura 7.25, donde todas las situaciones encontradas durante el proceso de navegación son conocidas (círculos grises), por lo que no se generan nuevos casos adaptados. Este hecho se debe a que el pasillo diseñado se dispuso en una configuración bastante ancha, por lo que el agente pudo atravesarlo utilizando el conocimiento contenido en la base de casos mediante una combinación de situaciones donde se detectaron dos y tres obstáculos.

Para comprobar la operación del sistema en un entorno complejo sin experiencia previa se va a modificar ligeramente el entorno de prueba. La figura 7.26 muestra un nuevo entorno en forma de pasillo estrecho, indicando también la posición actual del agente y el destino final (marcado con una "X").

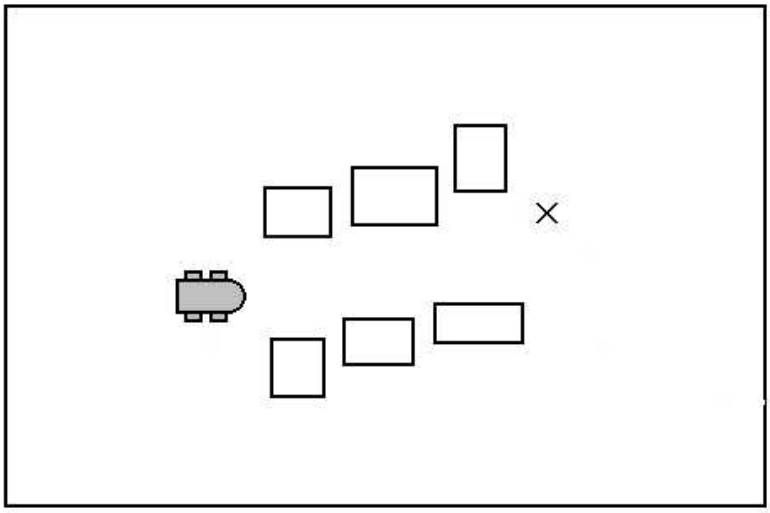

Figura 7.26: Entorno real con un pasillo estrecho.

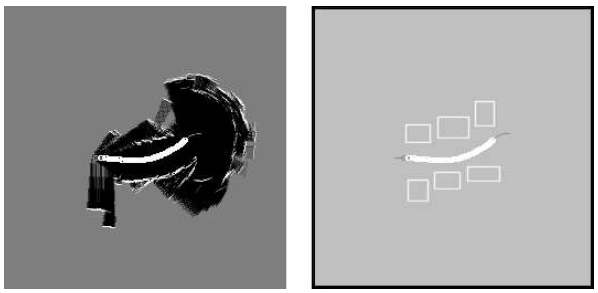

Figura 7.27: Operación en un entorno real con un pasillo estrecho.

El resultado de esta prueba se muestra en la figura 7.27, y en este caso el agente ha tenido que adaptar prácticamente todos los casos devueltos, ya que esta nueva situación no está contenida en la experiencia del sistema. A pesar de ello la trayectoria final recorrida es bastante suave, pues el agente prácticamente carece de margen de maniobra para operar.

Obviamente todos estos nuevos casos adaptados pueden ser incorporados en el conocimiento del sistema para ir aumentando la experiencia del mismo, siendo capaz de resolver situaciones similares en un futuro. Si se incorporan estos nuevos casos en el sistema optimizando adecuadamente la base de casos y se vuelve a repetir la misma prueba se obtiene el resultado mostrado en la figura 7.28. Ahora todas las situaciones afrontadas eran conocidas por el sistema, por lo que no se ha generado ningún nuevo caso adaptado. Este hecho refuerza la idea de que el sistema es capaz de perfeccionar su funcionamiento conforme afronta un mayor número de situaciones durante su operación.

Como principal conclusión de estas pruebas se puede afirmar que si bien la operación en entornos reales difiere de la operación en entornos simulados, es posible realizar un aprendizaje previo simulado para posteriormente utilizar el sistema en situaciones reales. No obstante, como se

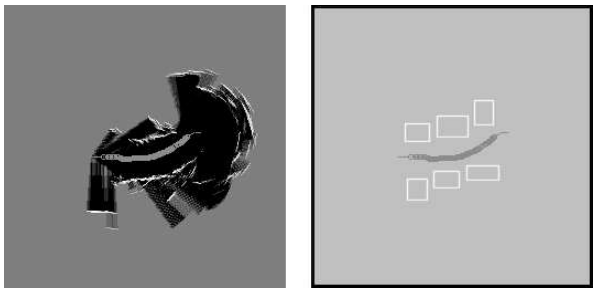

Figura 7.28: Operación en un entorno real con un pasillo estrecho con experiencia previa.

ha derivado del análisis de las pruebas, debido a los errores de las lecturas sonar se generan algunos casos adaptados que sí se corresponden con un conocimiento contenido en el sistema. Sin embargo, debido a que estos errores son circunstanciales y no se mantienen en el tiempo mientras el agente navega son rápidamente sustituidos por otros casos adecuados cuando se proporcionan lecturas correctas. Es necesario recordar que estos nuevos casos adaptados debidos a lecturas erróneas pueden ser convenientemente integrados en el sistema mediante el proceso de optimización de la base de casos implementado y descrito en el apartado 2.5 del capítulo 5, por lo que el sistema es capaz de perfeccionar el conocimiento inicial añadiendo el nuevo conocimiento derivado de la navegación en entornos reales.

3.1.4 Entornos reales con obstáculos móviles

La operación del sistema en presencia de obstáculos móviles está contemplada implícitamente en el esquema reactivo de navegación implementado. De hecho, un esquema reactivo considera únicamente la información instantánea que el agente capta del entorno, por lo que no existe diferencia alguna entre los obstáculos fijos y móviles, generándose ininterrumpidamente los comandos de navegación adecuados en función de la posición instantánea de todos los obstáculos del entorno.

La única diferencia apreciable entre los obstáculos fijos y los obstáculos móviles se encuentra en la generación del mapa métrico del entorno analizada en el apartado 2 del capítulo 4. En este tipo de mapas, los obstáculos fijos son claramente representados al generar sucesivas lecturas sonar que detectan el obstáculo en la misma posición, por lo que la probabilidad de ocupación aumenta con cada nueva lectura y al final el obstáculo se define claramente. Sin embargo, un obstáculo móvil varía su posición, por lo que la probabilidad de ocupación de las celdas que ocupa aumenta y disminuye con cada nueva lectura, hasta que finalmente desaparece al abandonar el rango de detección de los sensores sonar. Este comportamiento es totalmente correcto y deseable, pues los obstáculos que desaparecen del entorno deben desaparecer también del mapa que lo representa.

Llegados a este punto, no parece interesante repetir las pruebas realizadas hasta el momento con la simple incorporación de obstáculos móviles en el entorno, pues como se acaba de comentar al no existir diferencia alguna entre ambos tipos de obstáculos desde el punto de vista del sistema de navegación reactiva los resultados obtenidos serían los mismos que las derivados de las pruebas anteriores. Sin embargo, el esquema de navegación propuesto mediante *CBR* posee

una capacidad de operación muy flexible, pues como se ha demostrado anteriormente es posible incorporar distintos comportamientos de navegación en función del entrenamiento realizado. Para demostrar esta versatilidad se van a realizar otro conjunto de pruebas más complejas en las que no sólo se van a incluir obstáculos móviles en el entorno, sino también un sencillo sistema de navegación de tipo cooperativo con prioridades donde puedan coexistir múltiples agentes. Para realizar las pruebas, no obstante, se va a utilizar un único agente, mientras el otro agente lo simulará una persona.

Es importante matizar que con estas pruebas únicamente se pretende demostrar la flexibilidad del sistema *CBR* implementado a la hora de configurar los distintos comportamientos de navegación reactiva desarrollados. En ningún momento se pretende implementar un sistema completo de operación multiagente, ya que estos sistemas de navegación poseen su propia problemática y suelen ser bastante más complejos de implementar [MCM99]

Prueba 1: Operación en entornos multiagente

El entrenamiento del sistema se va a realizar mediante un operador humano, el cual guiará al agente por el entorno hasta alcanzar el destino final mientras evita los obstáculos del entorno. No obstante, con este tipo de entrenamiento el agente únicamente aprende a evitar los obstáculos, ya sean fijos o móviles, por lo que siempre se alejará de ellos intentando mantener una trayectoria segura sin colisiones. Si el obstáculo que se presenta es móvil, simplemente tratará de esquivarlo y adaptará su trayectoria al mismo. Parece evidente que en presencia de múltiples agentes este esquema resulta claramente insuficiente, pues en tales situaciones suelen existir reglas que intentan coordinar el comportamiento de cada agente con el fin de establecer una estrategia de navegación común más eficiente. La navegación basada en *CBR* permite implementar un sencillo esquema de navegación multiagente imponiendo reglas de prioridades que cedan el paso a los obstáculos móviles que aparezcan por la derecha, en un esquema parecido a las normas de circulación. De esta forma es posible compartir un mismo entorno por varios agentes navegando simultáneamente.

Para implementar este esquema de navegación multiagente es preciso incluir en la experiencia del sistema la información relativa a las distintas situaciones de prioridad donde el agente debe detenerse, que en el esquema propuesto se reducen a aquellas situaciones donde el agente encuentra obstáculos móviles en la parte frontal derecha. Para ello se ha añadido una nueva funcionalidad en el proceso de entrenamiento, de forma que el operador humano pueda detener al agente cuando deba ceder el paso. Para incorporar este nuevo conocimiento en el sistema ha sido necesario modificar ligeramente la representación del caso realizada en el apartado 2.2 del capítulo 5, incluyendo una nueva característica en el vector que define el caso para indicar si el agente debe detenerse o no tal y como se indica a continuación:

$$caso = [\alpha_{dest}, S_1, S_2, .., S_N, \alpha_{dir}, Stop] \tag{7.1}$$

donde α_{dest} representa la dirección en la que se encuentra el destino final respecto a la dirección de avance del agente, S_i representa la lectura del sensor i, N representa el número de sensores

que posee el agente, α_{dir} representa la dirección de avance del agente propuesta como solución y *Stop* es un *flag* que indica si el agente debe detenerse ante la configuración del entorno detectada.

Sin embargo, esta estrategia tal y como se describe presenta un serio inconveniente: si un obstáculo fijo aparece en la parte frontal derecha del agente este puede quedar permanentemente bloqueado. Para resolver esta situación es necesario incorporar algún mecanismo temporal que indique si el obstáculo detectado es fijo o móvil. La solución adoptada monitoriza la evolución temporal de aquellos obstáculos que están en la parte frontal derecha del agente, de forma que si un determinado obstáculo no se mueve durante una cantidad de tiempo determinada se considera como un obstáculo fijo y se sobrepasa.

Por motivos prácticos, además, no se detiene inmediatamente al agente tras detectar una situación donde deba ceder el paso. La estrategia adoptada se basa en reducir progresivamente la velocidad del agente, de forma que tras detectar unas pocas situaciones consecutivas en las que se deba ceder el paso el agente se detenga completamente. Esta solución se ha adoptado para prevenir por un lado que el sistema se detenga innecesariamente por lecturas sonar erróneas, y para aumentar la eficiencia del esquema al permitir un pequeño período de tiempo en el que el obstáculo móvil puede abandonar la zona frontal derecha del agente, reanudando este último su navegación sin necesidad de detenerse por completo.

Así pues, el esquema finalmente implementado es el que se describe a continuación:

1. Gracias al período de entrenamiento, cuando el agente detecta un obstáculo en su parte frontal derecha se activa el mecanismo de ceder el paso.

2. Cuando el agente identifica una posible situación en la que debe ceder el paso reduce progresivamente su velocidad, hasta detenerse por completo tras unas pocas lecturas.

3. Tras detenerse se activa un temporizador, que estará habilitado mientras se mantenga la potencial situación de ceder el paso.

4. Si el obstáculo detectado es un obstáculo móvil abandonará la parte frontal derecha del agente tras un período de tiempo, por lo que este continuará su movimiento. Si el obstáculo detectado es un obstáculo fijo el agente estará detenido hasta que el temporizador supere un umbral U_{time} establecido. Pasado dicho intervalo de tiempo, el agente continuará su movimiento y esquivará el obstáculo.

Las pruebas realizadas con el esquema propuesto han sido llevadas a cabo en un entorno donde los pasillos eran lo suficientemente anchos como para permitir el paso de dos agentes en paralelo. Como ocurría en las pruebas anteriormente realizadas, los distintos mapas presentados no han sido utilizados en el proceso de navegación, y son empleados para facilitar la exposición de los resultados.

La figura 7.29 muestra el proceso de entrenamiento llevado a cabo. En todos los casos el agente se dirigió por un pasillo hacia una intersección, a la vez que una persona entraba en

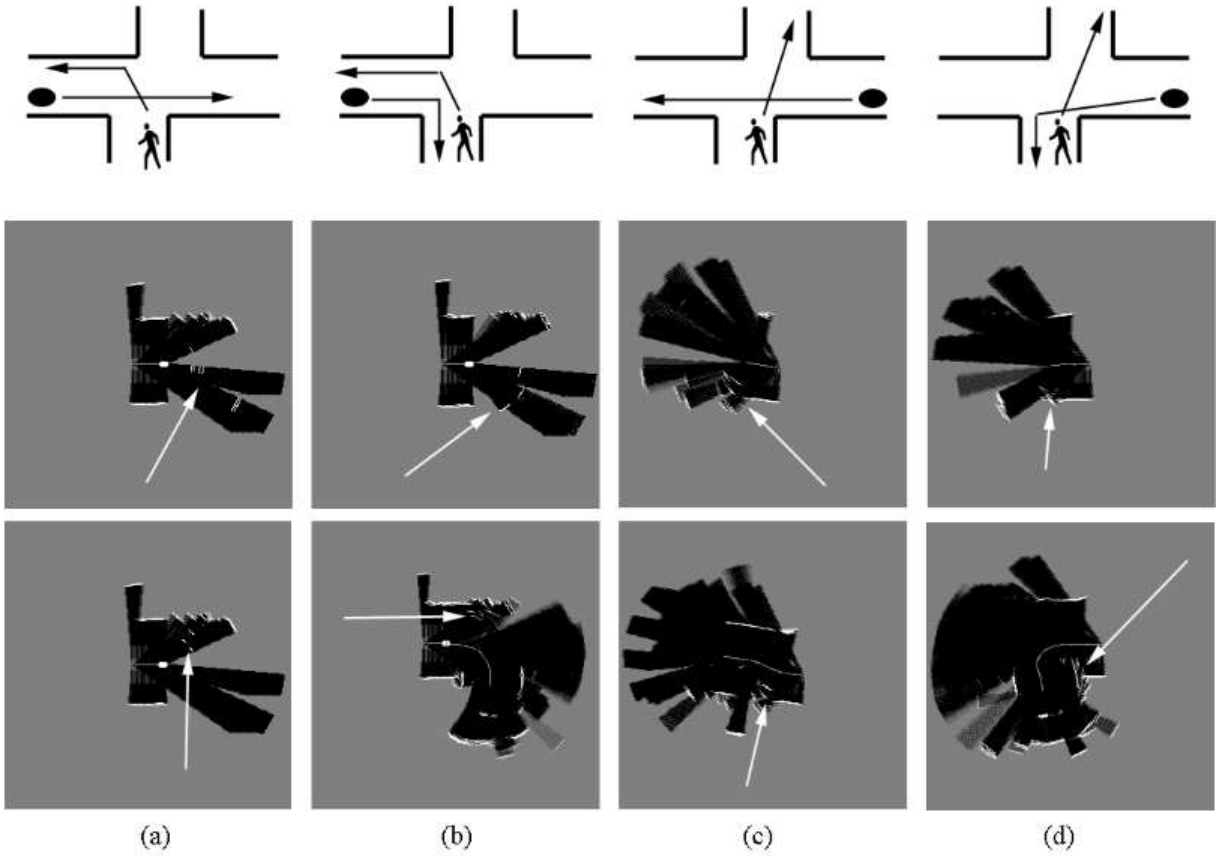

Figura 7.29: Entrenamiento en entornos multiagente: a) la persona tiene prioridad; b) la persona tiene prioridad; c) el agente tiene prioridad; d) el agente tiene prioridad.

dicha intersección procedente de otro pasillo. Junto con cada trayectoria de entrenamiento se presentan dos mapas métricos, que se corresponden con el instante inicial antes de resolver la situación de ceder el paso y con el instante final tras haber resuelto dicha situación. Sobre los mapas métricos se ha representado a la persona mediante una flecha blanca, pues como ya se ha comentado anteriormente los obstáculos móviles no dejan una constancia clara en los mapas métricos, tan solo una traza dispersa que tiende a eliminarse con sucesivas lecturas.

En las situaciones de las figuras 7.29.a y 7.29.b el agente fue detenido por el operador humano para ceder el paso a la persona. Esta situación donde el agente se detiene se ha marcado en las trayectorias realizadas mediante un círculo blanco. En las situaciones de las figuras 7.29.c y 7.29.d, sin embargo, el agente tiene preferencia, por lo que no fue detenido por el operador humano ante la persona que se aproximaba por la izquierda. Es importante notar que este entrenamiento no sólo le enseña al agente a detectar las situaciones donde debe ceder el paso, sino también la forma de esquivar los obstáculos presentes en el entorno.

Tras adquirir todos los casos de interés de la etapa de entrenamiento se realiza el proceso de optimización de la base de casos, generando el conocimiento inicial del sistema. Para agrupar correctamente los casos disponibles se ha modificado ligeramente el algoritmo de optimización

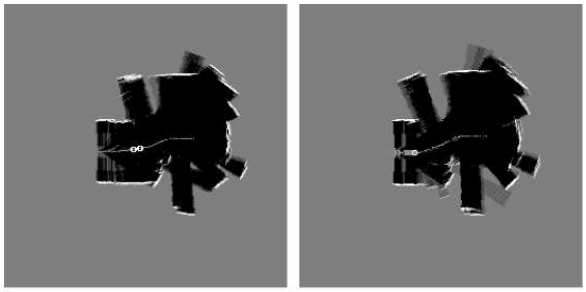

Figura 7.30: Operación en entornos multiagente con obstáculos fijos: a) navegación lejos de la pared; b) navegación cerca de la pared.

descrito en el apartado 2.5 del capítulo 5, agrupando por separado aquellos casos que tenían desactivo el *flag Stop* por una parte y aquellos casos que tenían activo el *flag Stop* por otra. Si no se hubiese hecho así se corría el riesgo de que se agrupasen casos donde el agente debía detenerse con otros casos cercanos donde el agente no debía detenerse, produciendo un resultado incoherente en la agrupación. Esta característica demuestra que el proceso de optimización de la base de casos diseñado es bastante versátil, pues se pueden agrupar los casos del sistema en función de la experiencia que representan.

Una primera prueba consistió en verificar que el nuevo esquema operaba correctamente esquivando los obstáculos fijos del entorno. La figura 7.30 se corresponde con una situación donde se ha colocado una caja de cartón en la intersección para que el agente la detecte en la parte frontal derecha, de forma que tenga que detenerse y cederle el paso. La figura 7.30.a muestra cómo el agente ha detectado la caja y se ha detenido en la ubicación marcada con un círculo blanco, pero transcurrido el período de tiempo U_{time} continúa su movimiento y esquiva la caja. En la figura 7.30.b se ha repetido la prueba, pero se ha situado al agente cerca de la pared para que la detecte como un obstáculo en la parte frontal derecha y se detenga. Se puede observar que el agente comienza a reducir su velocidad bastante antes de lo que lo hizo en la prueba 7.30.a, situación marcada en esta prueba con círculos grises. Finalmente se detiene a una distancia mayor de la caja ubicada en la intersección, y transcurrido el período de tiempo U_{time} continúa su movimiento y de nuevo esquiva la caja.

Una segunda prueba consistió en verificar el nuevo esquema en una situación donde el agente debía detenerse. En este caso se reprodujo la prueba de la figura 7.29.a realizada en la etapa de entrenamiento. La figura 7.31 muestra los resultados obtenidos en dicha prueba.

En este caso la persona, representada por la flecha blanca, es detectada por el agente en su parte frontal derecha, por lo que este último se detiene en la posición marcada con un círculo blanco. Una vez la persona ha sobrepasado al agente ya no es detectada en la parte frontal derecha del mismo, por lo que es agente continúa su movimiento hacia el destino final. Durante su operación el agente ha reproducido correctamente la secuencia de entrenamiento de la figura 7.29.a.

Otra prueba realizada consistió en verificar la operación del sistema cuando el agente no debía detenerse, para lo cual se reprodujo la prueba de la figura 7.29.c realizada en la etapa de

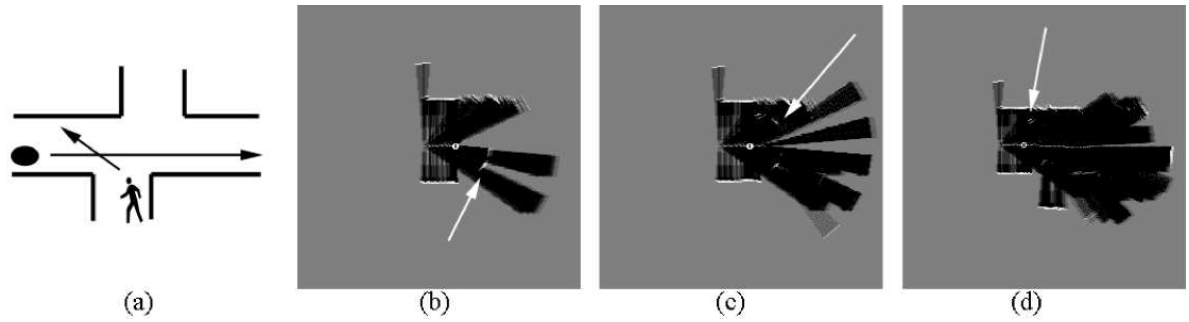

Figura 7.31: Operación en entornos multiagente con una persona con prioridad: a) configuración; b) detención del movimiento del agente; c) reanudación del movimiento del agente; d) llegada al destino final.

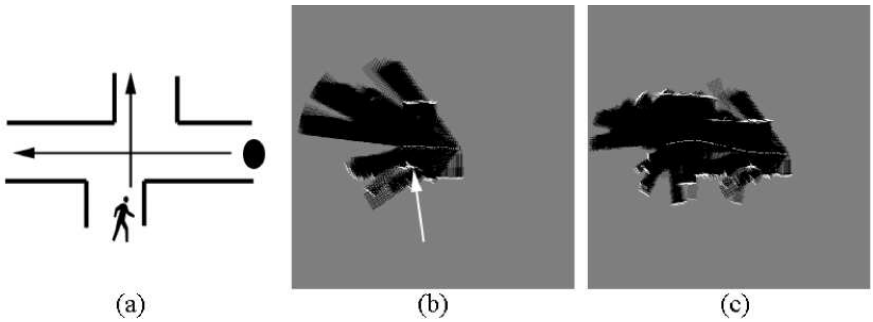

Figura 7.32: Operación en entornos multiagente con una persona sin prioridad: a) configuración; b) detección del movimiento de la persona; c) llegada al destino final.

entrenamiento. La figura 7.32 muestra en esta ocasión los resultados obtenidos.

En esta ocasión el agente continúa navegando sin detenerse cuando se detecta la persona, representada de nuevo con una flecha blanca. No obstante, el agente ha tenido que variar ligeramente su trayectoria para evitar la colisión con la persona que ha aparecido por su izquierda. De nuevo el agente ha reproducido correctamente la secuencia de entrenamiento de la figura 7.29.c.

Una última prueba realizada consistió en plantear una nueva situación no resuelta en el proceso de entrenamiento, para ver si el sistema es capaz de operar correctamente ante nuevas situaciones. La figura 7.33 muestra la configuración de esta prueba, que no forma parte del proceso de entrenamiento de la figura 7.29.

En esta ocasión tanto el agente como la persona deben pasar por el mismo pasillo, si bien el agente debe detenerse y cederle el paso a la persona al aparecer esta por su derecha. Según muestra la figura 7.33.b la persona, representada por una flecha blanca, es detectada por el agente antes de que este último comience a girar para introducirse en el nuevo pasillo, por lo que detiene su movimiento en la posición marcada por un círculo blanco. Cuando la persona supera al agente y desaparece de la parte frontal derecha del mismo reanuda su camino, como se muestra en la figura 7.33.c. Tras introducirse el agente en el nuevo pasillo la persona camina delante del agente, y como muestra la figura 7.33.d el agente no vuelve a detenerse más.

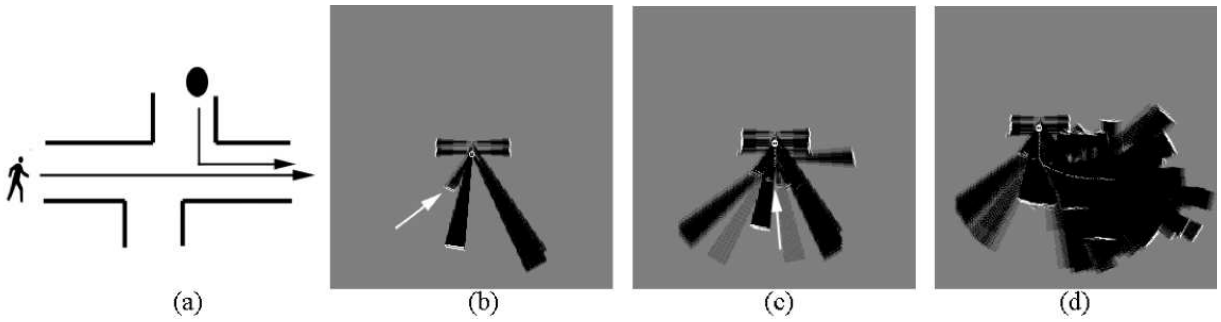

Figura 7.33: Operación en entornos multiagente con una persona con prioridad: a) configuración; b) detención del movimiento del agente; c) reanudación del movimiento del agente; d) llegada al destino final.

Estas pruebas ponen de manifiesto la enorme versatilidad del sistema reactivo de navegación implementado mediante el esquema *CBR*, pues con el adecuado proceso de entrenamiento es posible configurar numerosos comportamientos de navegación en el sistema.

3.2 Navegación planificada

El nivel de Navegación Planificada constituye el nivel de navegación intermedio del sistema, y es responsable de proporcionar un camino libre de obstáculos para alcanzar el destino final. Este nivel, pues, utiliza un proceso de planificación para optimizar bajo algún criterio el camino a seguir por el agente, evitando además las posibles trampas de mínimos locales en las que pueda quedar atrapado el nivel de Navegación Reactiva. Según la Jerarquía de Navegación descrita en el apartado 2 del capítulo 1 este nivel de navegación se corresponde con el nivel de Respuesta Inducida del grupo de Búsqueda de Caminos.

El nivel de Navegación Planificada es desarrollado por el esquema de planificación de caminos propuesto en el apartado 2 del capítulo 6. Para comprobar el correcto funcionamiento del sistema se van a desarrollar diferentes pruebas sobre dicho esquema: pruebas para demostrar la mayor eficiencia en la navegación aportada por el proceso de planificación, pruebas en entornos simulados y pruebas en entornos reales. Al igual que en las pruebas anteriores, en los mapas métricos utilizados las zonas libres se muestran en negro, las zonas ocupadas en blanco y las zonas no exploradas en gris.

3.2.1 Planificación de caminos

El nivel de Navegación Planificada incrementa la capacidad de navegación del sistema al generar un camino libre de obstáculos para alcanzar el destino final, camino que se calcula bajo algún criterio de optimización. Como se ha comentado en el apartado 2 del capítulo 6 el proceso de planificación de caminos propuesto se basa en un método de propagación de ondas, capaz de proporcionar caminos cortos, seguros, suaves y sencillos de seguir. Este proceso de planificación permite dirigir intencionadamente al sistema reactivo en la consecución de objetivos más óptimos

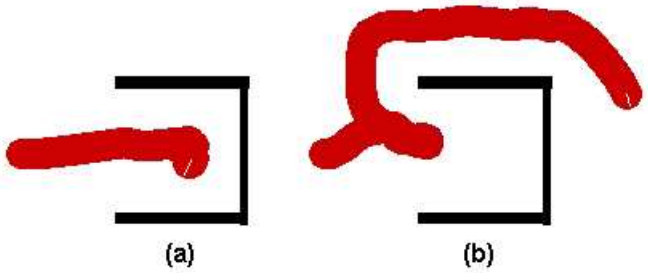

Figura 7.34: Operación del sistema ante un mínimo local: a) sin planificación; b) con planificación.

y eficientes.

El nivel de Navegación Planificada también permite resolver algunos problemas importantes del nivel de Navegación Reactiva, ya que como todo esquema reactivo este último depende únicamente de la percepción instantánea que el agente capta del entorno inmediato, por lo que está sometido a trampas de mínimos locales que pueden bloquear el avance del agente. Para demostrar este hecho se ha llevado a cabo una sencilla prueba en la que se ha configurado el entorno con una típica distribución de mínimo local, como se muestra en la figura 7.34.

La figura 7.34.a muestra la operación del sistema en este sencillo entorno bajo el nivel de Navegación Reactiva. Como se puede apreciar el sistema no es capaz de resolver adecuadamente dicha situación, pues al intentar avanzar el agente hacia el objetivo evitando la colisión con los obstáculos circundantes queda atrapado en el entorno diseñado. Aunque no es directamente apreciable en la figura, el agente ha recorrido numerosas veces la última etapa de la trayectoria, donde intenta sobrepasar la pared del fondo por la derecha y por la izquierda alternativamente. Un proceso de planificación adecuado como el propuesto en esta Tesis es capaz de detectar esta situación y dirigir al agente en la dirección adecuada para salir de la situación de bloqueo planteada. La figura 7.34.b muestra la operación del sistema en este entorno bajo el nivel de Navegación Planificada.

Esta sencilla prueba permite poner de manifiesto la importante mejora obtenida en la operación del sistema tras incorporar un adecuado proceso de planificación. Es por ello por lo que los sistemas híbridos que combinan adecuadamente una planificación eficiente con un esquema reactivo son actualmente el paradigma por excelencia en el desarrollo de sistemas robóticos, tal y como se analizó en el apartado 2 del capítulo 2.

3.2.2 Entornos simulados

Se va a proceder a continuación a verificar el comportamiento del sistema completo en el nivel de Navegación Planificada. Para permitir una mayor exploración del modo de operación se va a utilizar en primer lugar un entorno simulado, pasando posteriormente a un entorno real para comprobar la influencia de las lecturas erróneas de los sensores sonar en el correspondiente

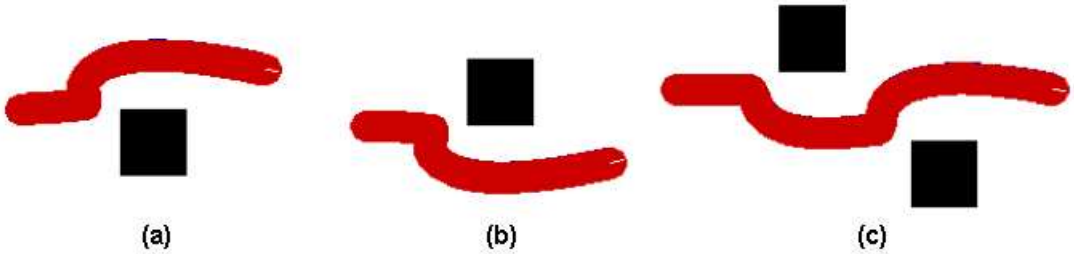

Figura 7.35: Entrenamiento del sistema reactivo en las pruebas del nivel de Navegación Planificada: a) un obstáculo a la derecha; b) un obstáculo a la izquierda; c) dos obstáculos.

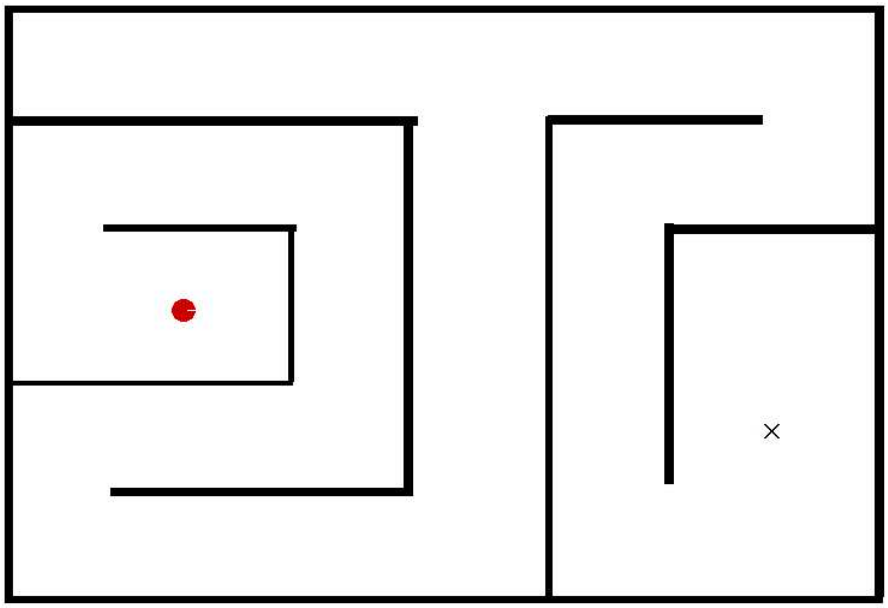

Figura 7.36: Entorno simulado para la prueba del nivel de Navegación Planificada.

proceso de planificación.

Puesto que el nivel de Navegación Planificada se basa en el nivel de Navegación Reactiva, antes de realizar ningún tipo de prueba es necesario llevar a cabo un entrenamiento del sistema reactivo. Este entrenamiento ha sido realizado por un operador humano con los comportamientos básicos que se muestran en la figura 7.35, utilizando para ello un entorno simulado con unas pocas situaciones, en concreto con uno y dos obstáculos únicamente. Tras aplicar el mecanismo de optimización sobre los casos adquiridos en las trayectorias de la figura 7.35 se obtiene la base de casos inicial que contiene el conocimiento de partida del sistema.

Para poner de manifiesto toda la capacidad de navegación incorporada en el sistema con el proceso de planificación desarrollado se va a configurar un gran entorno bastante complejo, conteniendo numerosas trampas de mínimos locales. La figura 7.36 muestra el entorno utilizado en esta prueba, así como la posición actual del agente y el destino final (marcado con una "X"). Es importante notar que este tipo de pruebas tan complejas son posibles gracias a la utilización

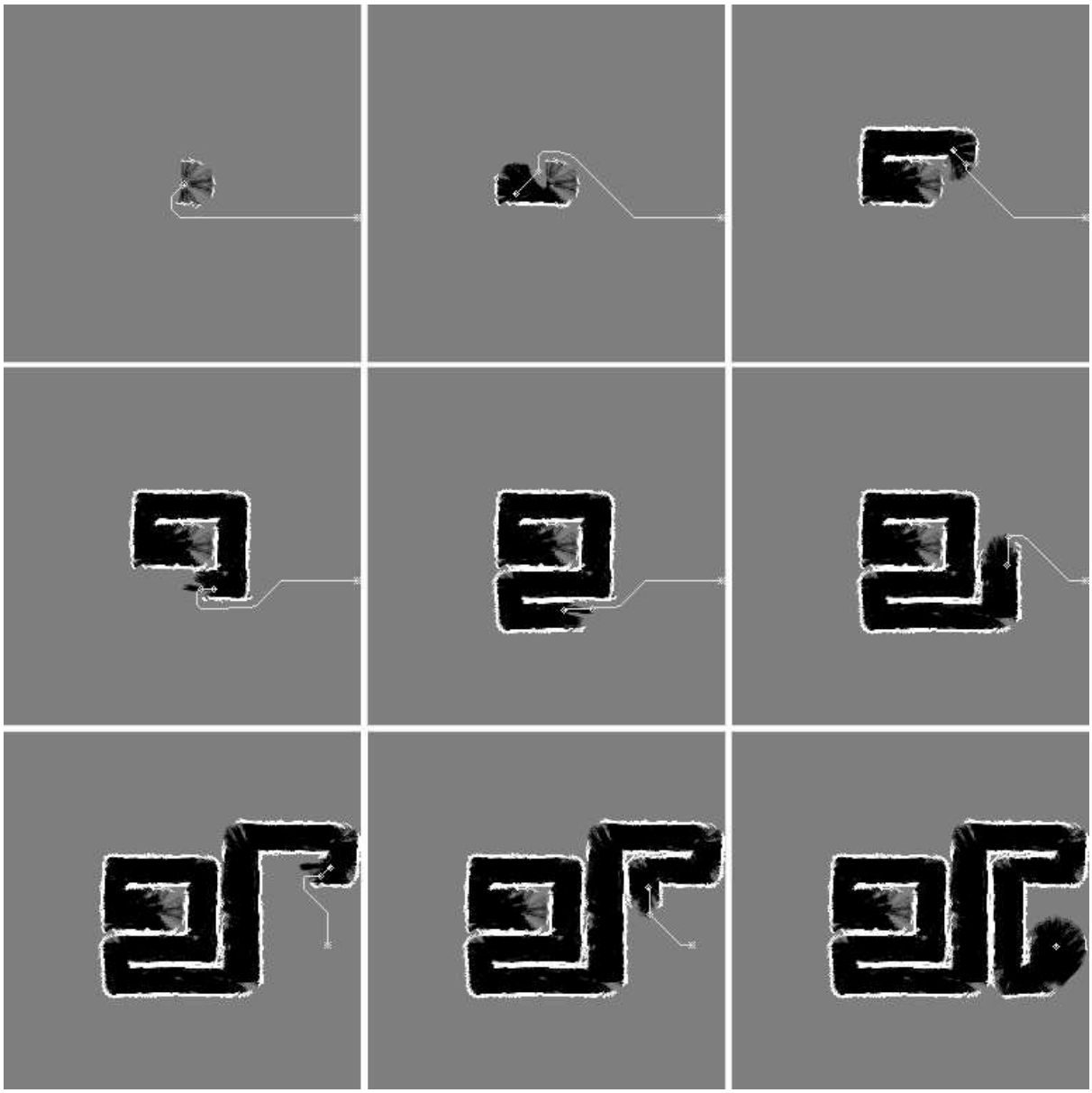

Figura 7.37: Etapas del proceso de navegación en la prueba del nivel de Navegación Planificada en entornos simulados.

de un simulador, pues resulta realmente difícil en entornos reales disponer de configuraciones como la mostrada.

En la figura 7.37 se muestran distintos instantes del proceso de navegación realizado a lo largo de la prueba. Se puede observar claramente que el camino calculado es bastante suave y simple en todos los casos. Gracias al proceso de planificación el agente ha sido capaz de alcanzar el destino final resolviendo adecuadamente las distintas trampas de mínimos locales presentes en el entorno, poniendo de manifiesto que la operación del sistema es correcta.

La trayectoria final recorrida por el agente durante la realización de esta prueba se representa en la figura 7.38, mostrando una trayectoria bastante suave. Se puede concluir por lo tanto

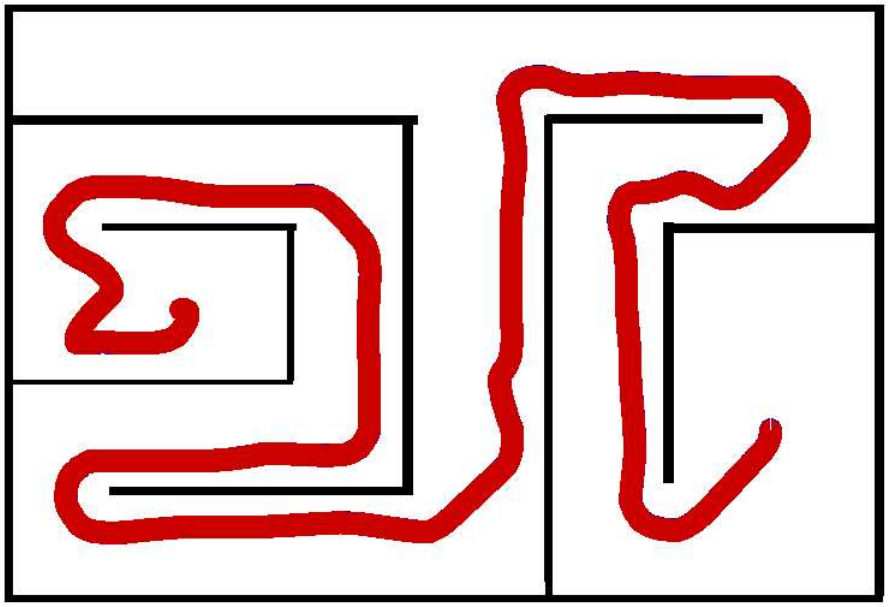

Figura 7.38: Trayectoria final recorrida en la prueba del nivel de Navegación Planificada en entornos simulados.

que el nivel de Navegación Planificada opera correctamente y añade un importante grado de funcionalidad a la navegación realizada por el sistema.

3.2.3 Entornos reales

La prueba realizada sobre el nivel de Navegación Planificada en entornos simulados ha demostrado la capacidad de navegación implementada con el planificador de caminos desarrollado en el sistema. A continuación se va a realizar una prueba similar en un entorno real bastante más sencillo, para verificar la influencia de las lecturas erróneas de los sensores sonar en el proceso de navegación.

Al igual que en las pruebas realizadas en entornos simulados, antes de realizar las correspondientes pruebas es necesario entrenar el sistema reactivo. Este entrenamiento puede realizarse en un entorno real, pero tal y como han demostrado las pruebas realizadas en el apartado 3.1.3 este entrenamiento también se puede realizar en un entorno simulado. Es por ello por lo que se va a utilizar en esta prueba el mismo entrenamiento que el realizado en la prueba del entorno simulado de la figura 7.35, permitiendo demostrar de nuevo la flexibilidad del sistema al operar en entornos reales tras realizar un entrenamiento simulado.

Para verificar la correcta operación del proceso de planificación se va a configurar un entorno donde el agente tenga que sobrepasar una trampa de mínimos locales, y donde coexisten distintos tipos de obstáculos de diferente material, como madera, cartón y metal. La figura 7.39 muestra el entorno utilizado en esta prueba, así como la posición actual del agente y el destino final (marcado con una "X").

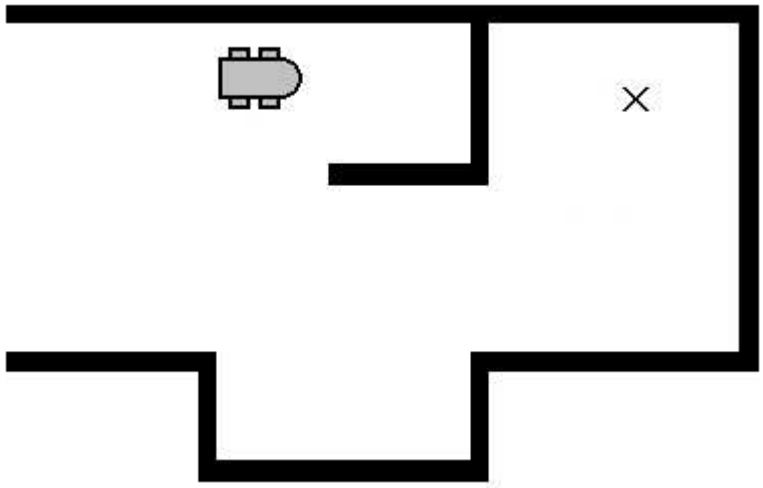

Figura 7.39: Entorno real para la prueba del nivel de Navegación Planificada.

En la figura 7.40 se muestran distintos instantes del proceso de navegación realizado a lo largo de la prueba. Nuevamente los distintos errores en las lecturas de los sensores sonar dan lugar a la generación de mapas métricos más difusos que los obtenidos en entornos simulados, donde estos errores no estaban presentes. Como principal consecuencia, los caminos obtenidos durante el proceso de planificación suelen ser algo más complejos al tener que bordear numerosos obstáculos, aunque muchos de ellos se deben a lecturas erróneas de los sensores sonar que suelen desaparecer del mapa métrico generado tras sucesivas lecturas.

La trayectoria final recorrida por el agente durante la realización de esta prueba se muestra en la figura 7.41. Analizando esta trayectoria se puede afirmar que a pesar de las lecturas erróneas de los sensores sonar el proceso de navegación sigue operando correctamente, proporcionando caminos bastante suaves y resolviendo las distintas situaciones de mínimos locales que se puedan presentar en el entorno.

3.3 Navegación Topológica

El nivel de Navegación Topológica constituye el nivel de navegación más avanzado del sistema, y es responsable de generar una ruta con las regiones a atravesar del entorno para alcanzar el destino final. Este nivel incorpora una mayor capacidad de abstracción al sistema, por lo que permite desarrollar comportamientos más elaborados que posteriormente guiarán al nivel de Navegación Planificada. Según la Jerarquía de Navegación descrita en el apartado 2 del capítulo 1 este nivel de navegación se corresponde con los niveles de Navegación Topológica y Navegación por Reconocimiento del grupo de Búsqueda de Caminos.

El nivel de Navegación Topológica es desarrollado por el esquema de planificación de rutas propuesto en el apartado 3 del capítulo 6. Para comprobar el correcto funcionamiento del sistema se van a desarrollar diferentes pruebas sobre dicho esquema: pruebas para demostrar

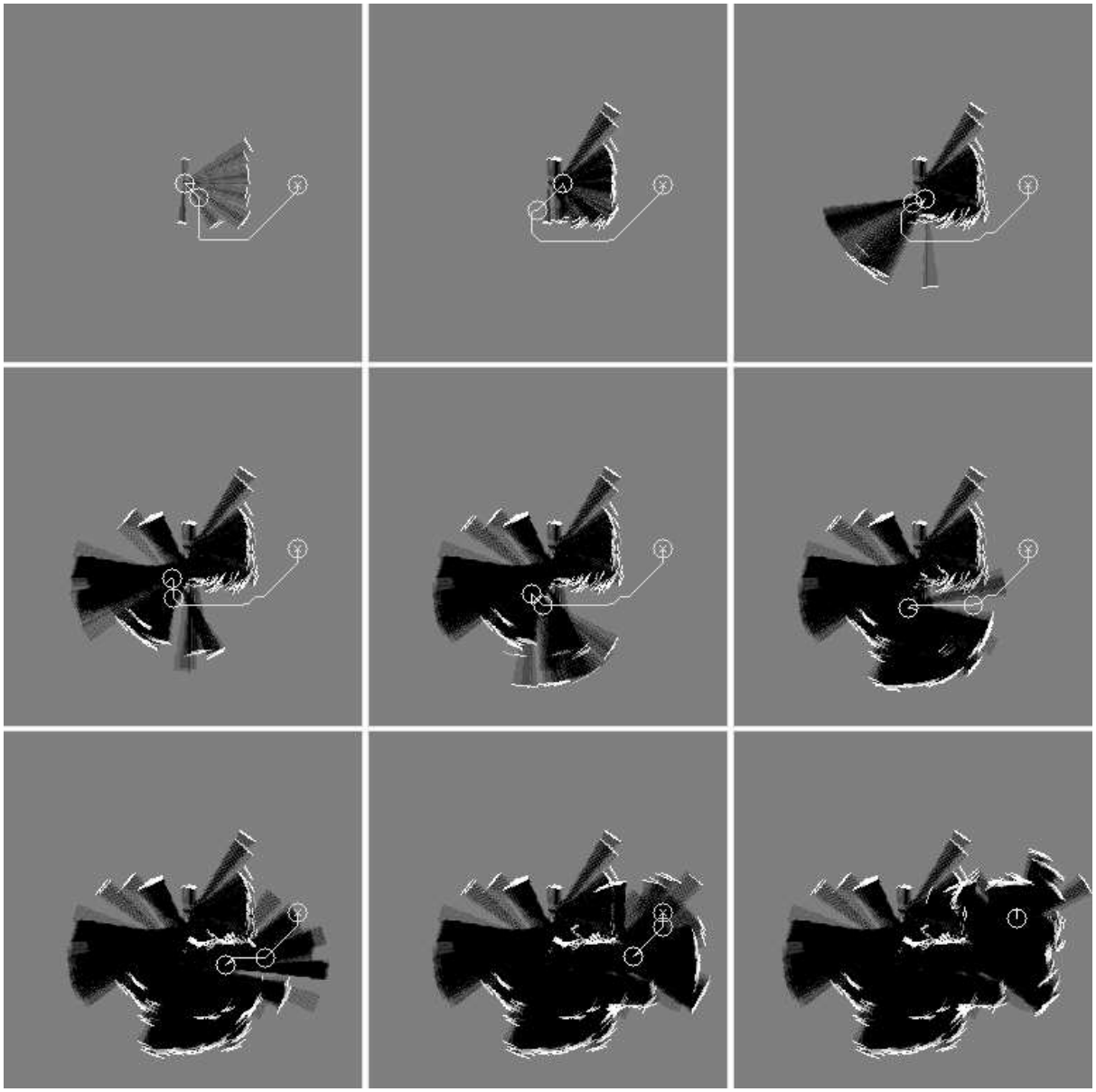

Figura 7.40: Etapas del proceso de navegación en la prueba del nivel de Navegación Planificada en entornos reales.

la mayor capacidad de planificación aportada en el sistema, pruebas en entornos simulados y pruebas en entornos reales. Al igual que en las pruebas anteriores, en los mapas métricos utilizados las zonas libres se muestran en negro, las zonas ocupadas en blanco y las zonas no exploradas en gris.

3.3.1 Planificación de rutas

Antes de proceder a realizar las pruebas de funcionamiento del sistema en el nivel de Navegación Topológica se va a realizar una sencilla prueba para poner de manifiesto la funcionalidad incorporada al sistema con este nivel de planificación.

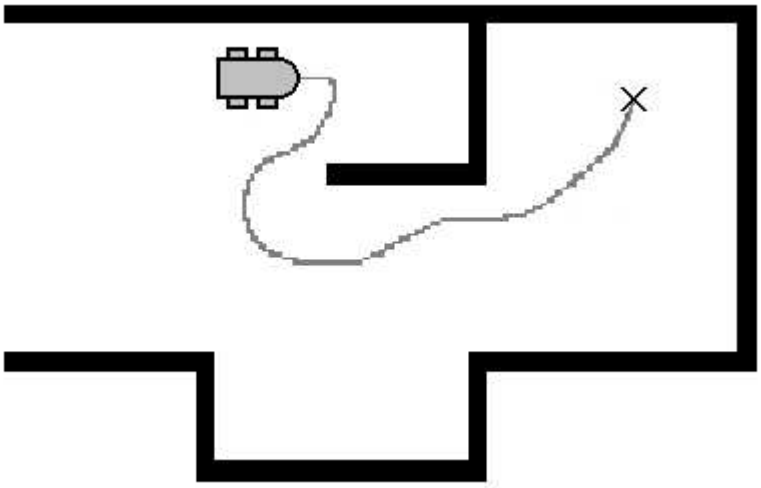

Figura 7.41: Trayectoria final recorrida en la prueba del nivel de Navegación Planificada en entornos reales.

Como se ha descrito en el apartado 3 del capítulo 6, una representación topológica realiza una descripción de las distintas regiones que posee el entorno así como de su conectividad, información que permite planificar el conjunto de regiones que hay que atravesar para alcanzar el destino final. Si por algún motivo la distribución del entorno se modifica la representación topológica absorberá dicha modificación, por lo que el proceso de planificación gestionará adecuadamente dicha circunstancia.

Para verificar este comportamiento se va a configurar un entorno simulado sencillo con una puerta, la cual se va a cerrar para modificar la distribución del entorno. La figura 7.42.a muestra el entorno utilizado en esta prueba, así como la posición actual del agente y el destino final (marcado con una "X"). La figura 7.42.b muestra el mapa métrico generado tras explorar el entorno con la puerta abierta, mientras que la figura 7.42.c muestra el mapa métrico generado tras explorar el entorno con la puerta cerrada.

La figura 7.43.a muestra el mapa topológico generado y la ruta calculada para alcanzar el destino final cuando la puerta se encuentra abierta, así como las distintas regiones de la representación topológica y sus centroides asociados. La posición actual del agente se ha representado mediante un círculo y el destino final con una "X". La ruta calculada atraviesa la puerta al constituir la solución más corta para alcanzar el destino final. Si en algún momento se cierra la puerta el mapa topológico del entorno muestra esta nueva situación, por lo que el proceso de planificación genera una nueva ruta de acuerdo con las nuevas circunstancias. La figura 7.43.b muestra el mapa topológico generado y la ruta calculada para alcanzar el destino final cuando la puerta se encuentra cerrada, así como las distintas regiones de la representación topológica y sus centroides asociados. De nuevo la posición actual del agente se ha representado mediante un círculo y el destino final con una "X". En este caso la ruta calculada no atraviesa la puerta al no estar conectadas entre sí las regiones que representan la posición actual del agente y el destino final, generándose una ruta bastante más larga para alcanzar el destino final de forma satisfactoria.

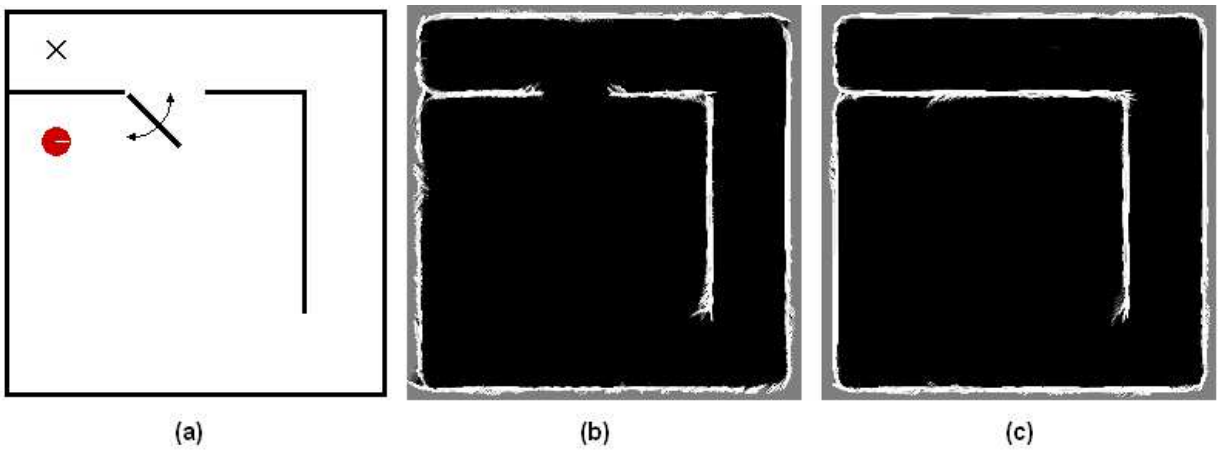

Figura 7.42: Entorno simulado con una puerta para la prueba de variación del entorno del nivel de Navegación Topológica: a) entorno original; b) mapa métrico con la puerta abierta; c) mapa métrico con la puerta cerrada.

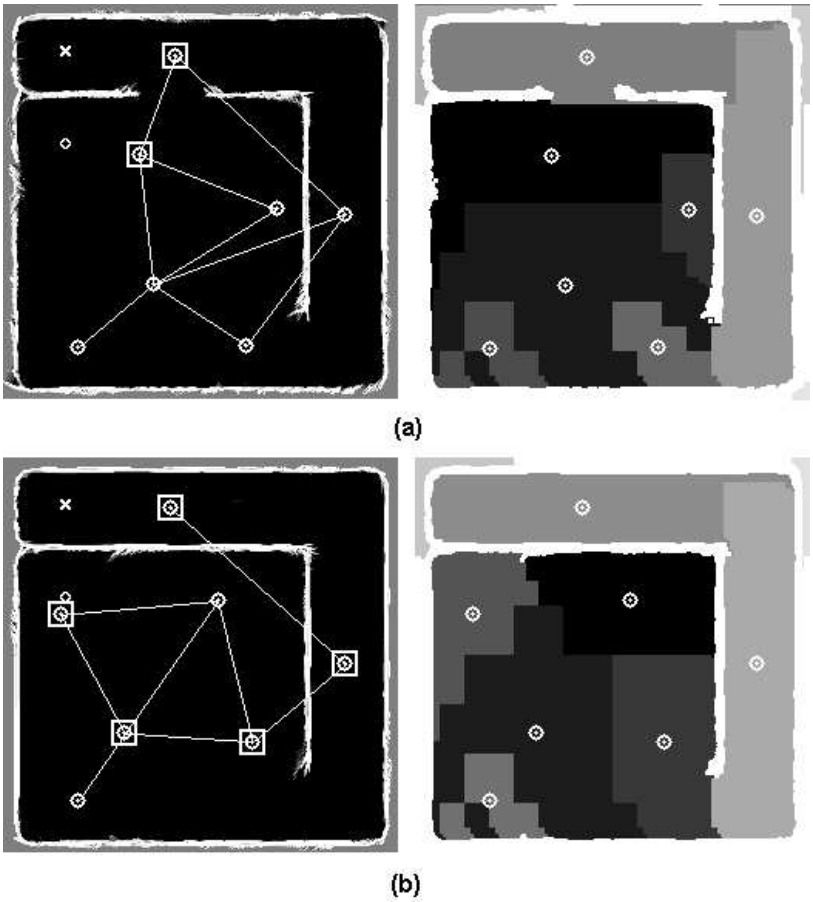

Figura 7.43: Mapa topológico y regiones del entorno simulado con una puerta: a) puerta abierta; b) puerta cerrada.

Con este experimento se ha querido poner de manifiesto la capacidad del sistema para calcular rutas alternativas ante variaciones sustanciales del entorno. Si bien en el sistema actual el mapa

métrico y el mapa topológico representan la misma zona del entorno, en aquellas circunstancias donde el mapa topológico represente un entorno mayor que el mapa métrico es posible realizar una planificación de la ruta a seguir más allá del entorno inmediatamente conocido, lo que puede optimizar la capacidad de navegación del sistema en grandes entornos.

3.3.2 Entornos simulados

Se pretende a continuación comprobar la operación del sistema completo en el nivel de Navegación Topológica. De nuevo, con el fin de permitir una mayor exploración del modo de operación se va a utilizar en primer lugar un entorno simulado, pasando posteriormente a un entorno real para comprobar la influencia de las lecturas erróneas de los sensores sonar en el correspondiente proceso de planificación.

Puesto que el nivel de Navegación Topológica se basa en el nivel de Navegación Planificada, y este a su vez se basa en el nivel de Navegación Reactiva, vuelve a ser necesario realizar un entrenamiento del sistema reactivo antes de realizar ninguna prueba. Para facilitar la comparación de los resultados obtenidos con las pruebas realizadas sobre el nivel de Navegación Planificada se van a mantener las mismas condiciones de entrenamiento, por lo que de nuevo se ha utilizado la base de casos obtenida tras el proceso de optimización de las trayectorias de entrenamiento de la figura 7.35.

La figura 7.44 muestra el entorno utilizado en esta prueba, así como la posición actual del agente y el destino final (marcado con una "X"). En la figura 7.45 se muestran distintos instantes del proceso de navegación realizado a lo largo de la prueba. Para cada instante se muestran el mapa métrico y el mapa topológico del entorno, la ruta calculada para alcanzar el destino final, las regiones actual y siguiente de la ruta, la zona del mapa métrico procesada y el camino con el próximo destino intermedio para alcanzar la siguiente región de la ruta.

Inicialmente el entorno se encuentra inexplorado, y cuando se especifica un destino final el agente comienza a moverse reactivamente a la vez que se comienza a generar el mapa topológico del entorno para calcular una ruta que permita alcanzar dicho destino final. Cuando se obtiene el mapa topológico se selecciona la ruta más adecuada a seguir, determinando la siguiente región a visitar del entorno. Una vez extraída la siguiente región a visitar, se calcula un camino libre de obstáculos entre la región actual y la siguiente región a visitar, el cual seguirá reactivamente el agente mientras evita los posibles obstáculos inesperados que aparezcan en el entorno.

Mientras el agente se va moviendo se va actualizando el mapa métrico, por lo que se genera un nuevo mapa topológico cada vez que el mapa métrico cambia significativamente, como se ha descrito en el apartado 3.4 del capítulo 6. Cuando el entorno está ampliamente inexplorado el proceso de planificación se ejecuta bastante a menudo, pues los cambios significativos en el entorno ocurren con bastante frecuencia.

En las distintas etapas de la figura 7.45 se muestra cómo el proceso de planificación se va ajustando a las características del entorno, el cual cada vez está más explorado. A mayor

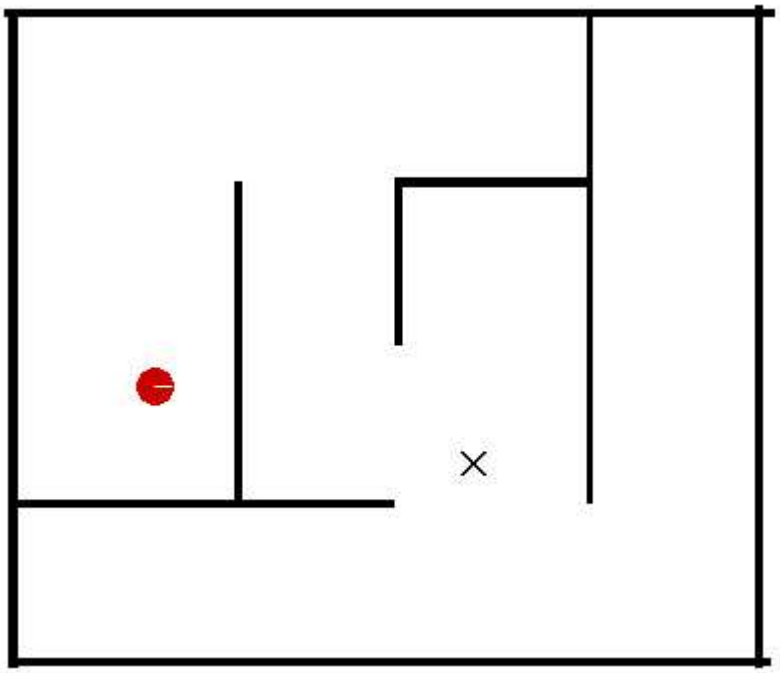

Figura 7.44: Entorno simulado para la prueba del nivel de Navegación Topológica.

exploración del entorno mayor es la información que el proceso de planificación posee del mismo, por lo que se pueden establecer planes más correctos que no necesitan ser recalculados con tanta frecuencia. De hecho, mientras se está procediendo a la exploración del entorno es normal la aparición de numerosos nodos en el mapa topológico que posteriormente se irán fusionando cuando la zona es explorada completamente.

Es importante notar que las zonas inexploradas también han sido representadas en el mapa topológico, por lo que definen regiones que pueden formar parte de la ruta calculada y pueden ser atravesadas. Si durante el proceso de navegación el entorno es explorado en dichas regiones el mapa topológico se actualizará convenientemente para representarlas adecuadamente.

La trayectoria final recorrida por el agente durante la realización de esta prueba se muestra en la figura 7.46, sobre la que se han marcado con un círculo las posiciones donde ha sido necesario emplear el sistema de evitación de obstáculos basado en *CBR*. Dicho sistema de evitación de obstáculos ha sido utilizado en escasas ocasiones, debido a que el entorno es bastante grande y permite navegar a través suya siguiendo el camino propuesto por el planificador de caminos, el cual está lo suficientemente alejado de los obstáculos presentes en el entorno. En las situaciones donde el sistema de evitación de obstáculos tuvo que ser utilizado, todos los casos obtenidos fueron adaptados al presentar situaciones desconocidas. Este hecho se debe al reducido conocimiento que posee el sistema inicialmente, al haber realizado un escaso entrenamiento del mismo. La principal consecuencia es que la trayectoria final recorrida presenta algunos cambios de dirección abruptos en determinadas posiciones, aunque es importante recordar que todos estos casos adaptados son progresivamente incorporados al sistema, aumentando el conocimiento del mismo y obteniendo trayectorias cada vez más suaves.

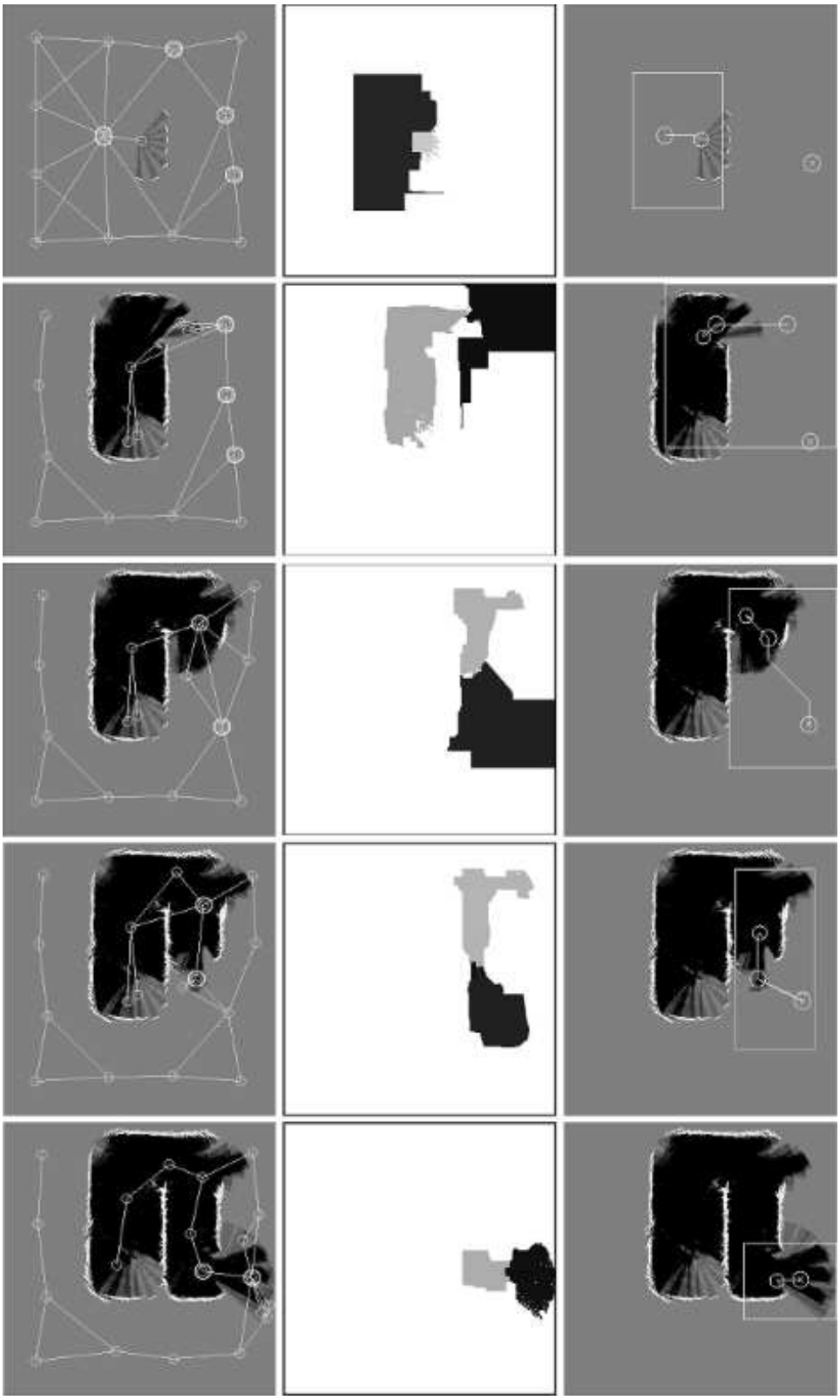

Figura 7.45: Etapas del proceso de navegación en la prueba del nivel de Navegación Topológica en entornos simulados.

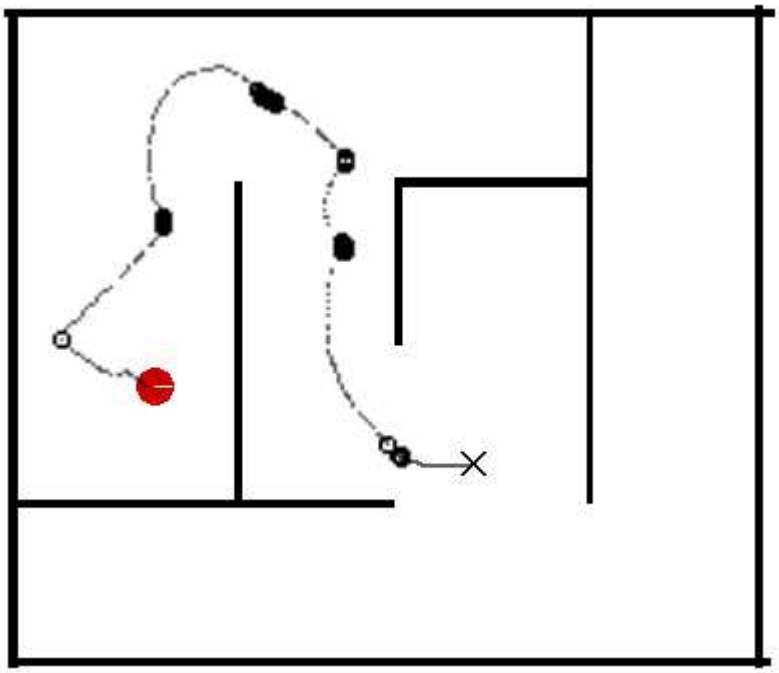

Figura 7.46: Trayectoria final recorrida en la prueba del nivel de Navegación Topológica en entornos simulados.

Resulta interesante comparar los resultados obtenidos por el nivel de Navegación Topológica de la figura 7.46 con los resultados obtenidos por el nivel de Navegación Planificada de la figura 7.38. A simple vista se puede constatar que resulta más suave y eficiente la trayectoria obtenida mediante el nivel de Navegación Planificada. Este hecho corrobora el análisis realizado en el apartado 4 del capítulo 6, y se debe a que el planificador de caminos produce una trayectoria que une directamente la posición actual del agente con el destino final, mientras que el planificador de rutas produce una trayectoria que une la posición actual del agente con el destino final a través de los centroides de las regiones a atravesar en el proceso de navegación. Obviamente resulta menos eficiente alcanzar el destino final atravesando una serie de centroides que alcanzar el destino final mediante un camino directo. No obstante, la trayectoria final recorrida sigue siendo bastante adecuada.

Como ya ha sido analizado en el apartado 4 del capítulo 6 la mejora de la capacidad de navegación implementada por el nivel de Navegación Topológica radica en el proceso de planificación en grandes entornos, pues a mayor tamaño del mapa métrico más costosa desde el punto de vista temporal resulta la planificación de caminos y menos costosa en comparación resulta la planificación de rutas. Además, permite implementar otro tipo de comportamientos más inteligentes como navegación a través de varios emplazamientos simultáneamente, navegación en grandes entornos, o exploración eficiente de entornos. Si bien estos comportamientos no se han incorporado en la presente Tesis resultan una posible expansión del sistema, como se describirá en el apartado 3.2 del capítulo 8.

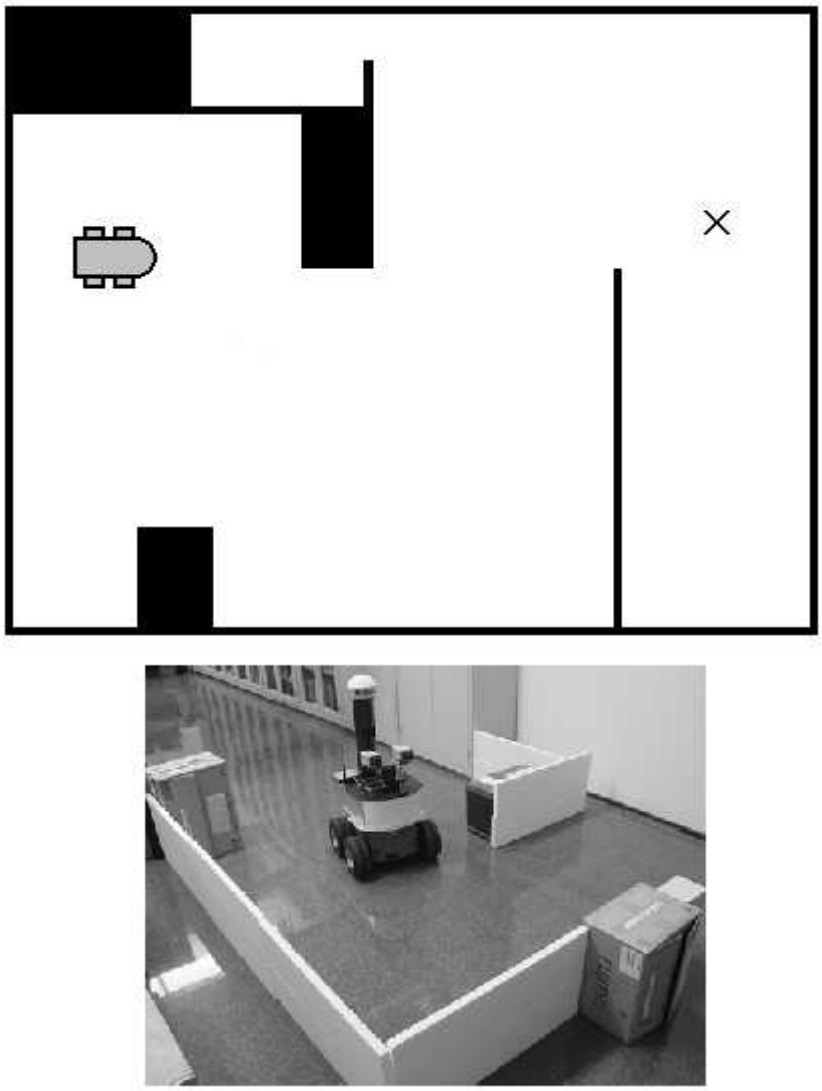

Figura 7.47: Entorno real para la prueba del nivel de Navegación Topológica.

3.3.3 Entornos reales

La prueba realizada sobre el nivel de Navegación Topológica en entornos simulados ha demostrado la capacidad de navegación incorporada en el sistema con el planificador de rutas desarrollado. A continuación se va a realizar una prueba similar en un entorno real, con el objetivo de verificar de nuevo la influencia de las lecturas erróneas de los sensores sonar en el proceso de navegación.

Antes de realizar las correspondientes pruebas es necesario entrenar el sistema reactivo, para lo cual se va a volver a utilizar el mismo entrenamiento que el realizado en la prueba del entorno simulado, descrito por las trayectorias de la figura 7.35.

Con el fin de demostrar de nuevo la eficiencia del proceso de planificación se va a configurar un entorno donde el agente tenga que sobrepasar una trampa de mínimos locales, y donde coexisten

distintos tipos de obstáculos de diferente material, como madera, cartón y metal. La figura 7.47 muestra el entorno utilizado en esta prueba, así como la posición actual del agente y el destino final (marcado con una "X").

En la figura 7.48 se muestran distintos instantes del proceso de navegación realizado a lo largo de la prueba. Para cada instante se muestran el mapa métrico y el mapa topológico del entorno, la ruta calculada para alcanzar el destino final, las regiones actual y siguiente de la ruta, la zona del mapa métrico procesada y el camino con el próximo destino intermedio para alcanzar la siguiente región de la ruta.

El comportamiento del sistema en el entorno real sigue siendo correcto, si bien el mapa métrico presenta menor definición. Este hecho se debe de nuevo a la característica no ideal de los sensores que posee el agente, que al ser de tipo sonar están sujetos a lecturas erróneas por ecos múltiples y reflexiones débiles que no estaban presentes en el entorno simulado. No obstante, aunque la definición del mapa métrico es menor el sistema sigue operando correctamente, si bien aparecen un elevado número de nodos en el mapa topológico que se corresponden con regiones pequeñas del entorno. Estas pequeñas regiones se van fusionando conforme el entorno va siendo explorado, por lo que las incertidumbres propias de los sensores sonar son correctamente gestionadas por el sistema.

La trayectoria final recorrida por el agente durante la realización de esta prueba se muestra en la figura 7.49, sobre la que se han marcado con un círculo las posiciones donde se ha tenido que utilizar el sistema de evitación de obstáculos basado en *CBR*. Este sistema de evitación de obstáculos debe ser empleado bastante a menudo, debido a que el tamaño del entorno es sustancialmente menor que en el caso de la prueba sobre entornos simulados y los obstáculos están bastante más cercanos al agente. La mayoría de los casos obtenidos tuvieron que ser adaptados al representar circunstancias desconocidas, hecho que de nuevo es debido al reducido conocimiento que posee el sistema inicialmente al haber realizado un escaso entrenamiento del mismo.

Es interesante resaltar, por último, que el sistema también es capaz de operar correctamente con obstáculos móviles gracias al esquema reactivo de navegación implementado, como ya ha sido comprobado en el apartado 3.1.4. Un obstáculo móvil es representado en el mapa métrico como una zona dispersa que es borrada cuando desaparece, a no ser que permanezca estático en el entorno, en cuyo caso será representado como otro obstáculo más.

En este punto se concluye la fase de pruebas realizadas sobre el sistema, considerándose que los resultados obtenidos son plenamente satisfactorios y corroboran el correcto funcionamiento del sistema propuesto conforme ha sido diseñado en capítulos anteriores.

4 Conclusiones

En el presente capítulo se han descrito las diversas pruebas efectuadas sobre el sistema para caracterizar y verificar el correcto funcionamiento del mismo. Tras describir los distintos recursos

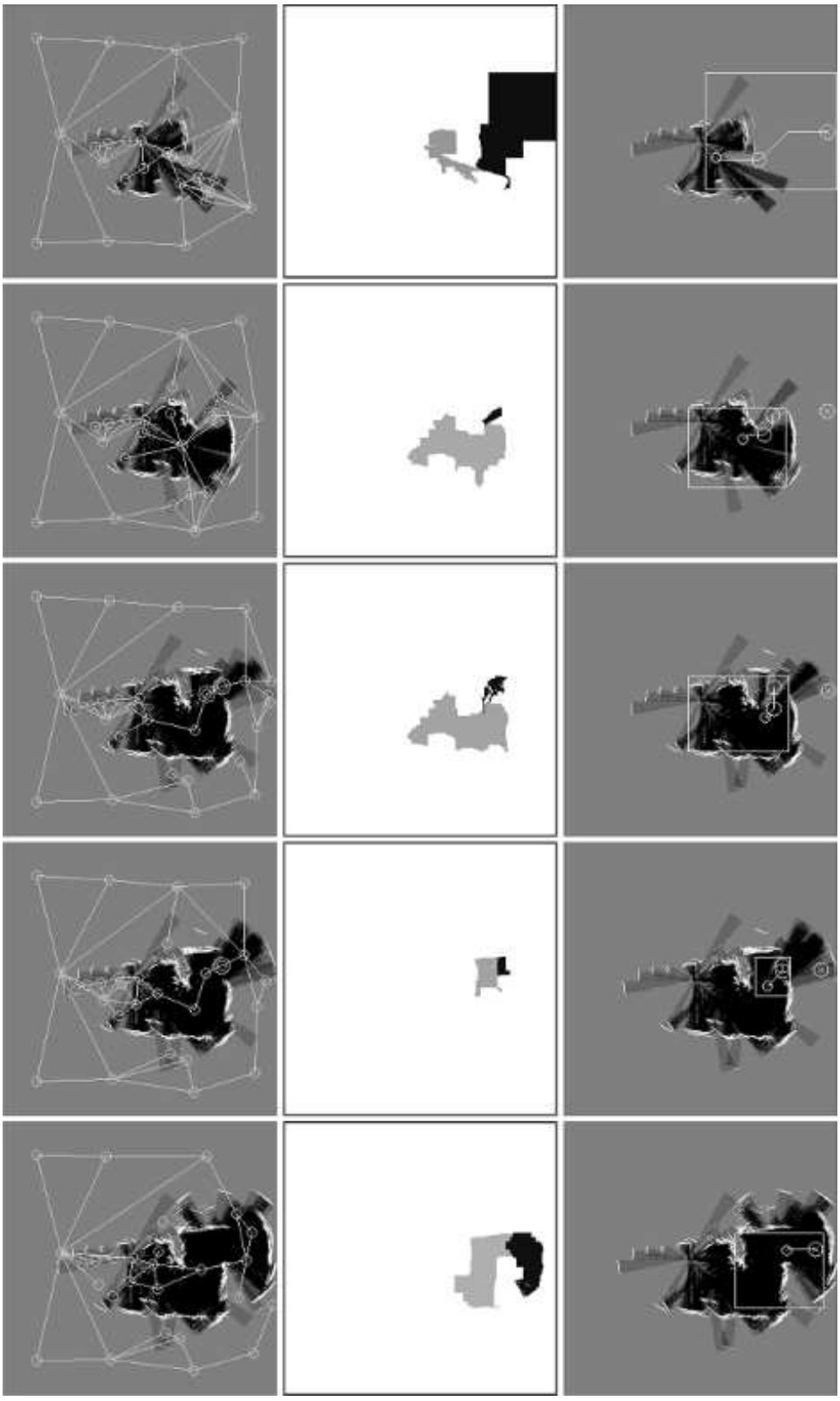

Figura 7.48: Etapas del proceso de navegación en la prueba del nivel de Navegación Topológica en entornos reales.

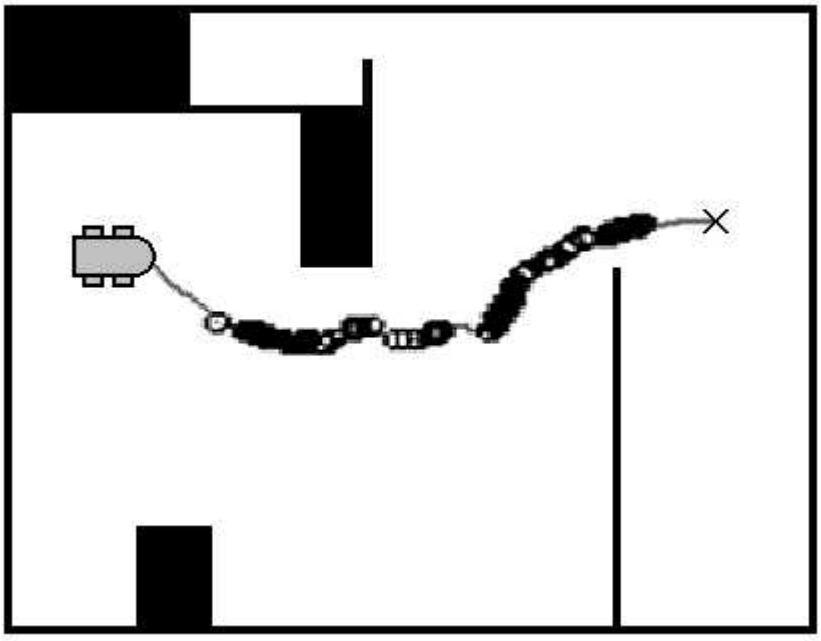

Figura 7.49: Trayectoria final recorrida en la prueba del nivel de Navegación Topológica en entornos reales.

y elementos empleados en el entorno de pruebas se procede a analizar exhaustivamente el estilo de la arquitectura *DLA*. Para ello se ha caracterizado el comportamiento de cada uno de los esquemas implementados: **esquema distribuido básico**, **esquema distribuido síncrono** y **esquema local síncrono**. El esquema distribuido básico presenta unas prestaciones bastante superiores al esquema original propuesto por Dulimarta, en el cual se basa la arquitectura implementada. El esquema distribuido síncrono aumenta la eficiencia del sistema al permitir la activación y desactivación selectiva de los distintos módulos, permitiendo una mejor utilización de los recursos disponibles y nuevas funcionalidades como la distribución automática de módulos. Por último, el esquema local síncrono implementa una optimización muy destacable en aquellos escenarios en los que distintos módulos residen en la misma máquina, circunstancia habitual en agentes diseñados para operar en entornos exteriores.

Una vez concluidas las pruebas del estilo se verifica el correcto funcionamiento de la estructura de la arquitectura *DLA*. Para ello se ha comprobado el funcionamiento de cada uno de los niveles de navegación implementados en el sistema explorando las nuevas funcionalidades incorporadas en el sistema con cada uno de ellos: **Navegación Reactiva**, **Navegación Planificada** y **Navegación Topológica**. Por motivos de seguridad y flexibilidad suele ser conveniente realizar inicialmente las pruebas bajo un entorno simulado, pues posibilita la configuración de entornos con características especiales. No obstante, es extremadamente importante verificar el comportamiento del sistema bajo condiciones reales de operación, para comprobar la sensibilidad del mismo frente a efectos no deseados como errores en las lecturas de los sensores, presencia de obstáculos móviles o cualquier tipo de imprevisto. Es por ello por lo que se han realizado numerosas pruebas, tanto en entornos simulados como en entornos reales. Todas ellas

han puesto de manifiesto una gran versatilidad del esquema reactivo propuesto, así como una mayor eficiencia y capacidad de navegación mediante los procesos de planificación diseñados.

Capítulo 8

Conclusiones y Líneas Futuras

En este capítulo se describen las conclusiones y líneas futuras derivadas con la realización de la presente Tesis, así como las contribuciones más importantes derivadas de la misma.

Este capítulo se ha organizado como se indica a continuación. En el apartado 1 se presentan las conclusiones más importantes obtenidas a lo largo de los diferentes capítulos. En el apartado 2 se enumeran las principales contribuciones y logros obtenidos durante la realización de la presente Tesis. Por último, en el apartado 3 se describen las líneas futuras que pueden contribuir a la ampliación del trabajo realizado.

1 Conclusiones

A lo largo de los distintos capítulos en los que se ha organizado esta Tesis se han ido obteniendo numerosas conclusiones del trabajo realizado, las cuales permiten determinar las características más importantes del sistema desarrollado. Se presentan a continuación todas estas conclusiones.

1.1 Introducción

Aunque la producción industrial ha sido tradicionalmente uno de los principales campos de aplicación de la Robótica, la evolución tecnológica de nuestros días está permitiendo una silenciosa pero imparable introducción de nuevos robots en nuestras vidas. Todos estos robots que nos acompañan día a día están asistiendo al ser humano en la realización de tareas complejas, repetitivas, desagradables y peligrosas, ya sea en entornos hostiles o de difícil acceso.

En este escenario, la Robótica se ha convertido en un sector estratégico clave para el desarrollo de las distintas potencias mundiales. De hecho, las previsiones de crecimiento de los diferentes sectores del mercado de la Robótica para los próximos años son realmente elevadas, marcando una tendencia hacia el desarrollo de robots que asistan al ser humnao y redunden en un claro beneficio económico y social.

Una característica bastante extendida y deseada en la implementación de estos nuevos robots radica en incorporar en los mismos la capacidad de **navegación en entornos dinámicos**, con el fin de realizar tareas cada vez más versátiles y complejas en el propio entorno natural del ser humano. En este sentido, han sido desarrollados numerosos esquemas de navegación para agentes autónomos móviles, aunque la evidente falta de estandarización existente dificulta enormemente su adaptación a otros sistemas distintos de aquellos para los que fueron concebidos. En consecuencia, la implementación de nuevos agentes implica el desarrollo de nuevos esquemas de navegación, siendo bastante complejo reutilizar los ya existentes.

El objetivo de esta Tesis consiste en establecer una infraestructura básica que permita incorporar la capacidad de navegación en entornos dinámicos no estructurados a un agente autónomo móvil genérico, permitiendo además su posterior adaptación a distintos agentes y una fácil ampliación de sus capacidades. Para desarrollar un esquema de navegación lo suficientemente versátil, se ha utilizado una **Jerarquía de Navegación** que clasifica los comportamientos de navegación existentes en el reino animal según su complejidad y eficiencia, procediendo a un desarrollo gradual de los mismos hasta alcanzar los comportamientos más avanzados.

1.2 Arquitecturas de control

El desarrollo de agentes autónomos móviles constituye una tarea lo suficientemente compleja como para emplear algún tipo de estrategia que simplifique su implementación. Normalmente, en su desarrollo se suele llevar a cabo una **descomposición** del sistema en un conjunto de unidades funcionales menores conocidas como **módulos**. Estos módulos poseen una complejidad menor que el sistema completo, por lo que su diseño suele ser bastante más abordable. Finalmente, tras la implementación de estos módulos es necesario realizar un proceso de **integración** para que todos ellos operen conjuntamente en el desarrollo del sistema completo.

Con el fin de abordar eficientemente el desarrollo de agentes autónomos móviles empleando esta estrategia es deseable disponer de alguna herramienta que gestione adecuadamente la complejidad resultante de aplicar los procesos de descomposición e integración. Esta herramienta se conoce como **arquitectura de control**, responsable de suministrar los mecanismos adecuados que permiten simplificar el desarrollo de agentes autónomos móviles. Las arquitecturas de control constituyen una parte casi imprescindible en la implementación de agentes autónomos móviles, determinando en gran medida las prestaciones finales obtenidas en su implementación.

Actualmente se utilizan como mejor alternativa las **arquitecturas híbridas**, capaces de integrar eficientemente comportamientos reactivos rápidos con comportamientos deliberados de planificación. De esta forma es posible responder rápidamente ante los estímulos captados del entorno mientras se posibilita una planificación de la operación del sistema para el desarrollo de tareas complejas.

Desafortunadamente, a pesar de la importancia de las arquitecturas de control en el desarrollo de agentes autónomos móviles, no se han descrito todavía metodologías estructuradas de diseño ni esquemas métricos adecuados para diseñarlas y compararlas. Esta falta de metodología ha

originado la aparición de una gran cantidad de propuestas distintas que solucionan problemas particulares, sin existir una arquitectura de control definitiva lo suficientemente general que se pueda utilizar como referencia.

1.3 La arquitectura de control *DLA*

Una correcta elección de la arquitectura de control utilizada constituye un aspecto fundamental en el desarrollo de un agente autónomo móvil, pues la simplificación obtenida en el proceso de implementación depende directamente de las prestaciones mostradas por la arquitectura de control utilizada. No obstante, a pesar de la gran diversidad existente suele ser relativamente complejo encontrar una arquitectura de control que se adecúe a los requisitos de la aplicación a desarrollar, debido a que proporcionan soluciones particulares no siempre adaptables a otros escenarios.

Para intentar aliviar esta situación se ha optado por desarrollar una nueva arquitectura de control, con el objetivo de facilitar su reutilización y adaptación en la implementación de nuevos agentes autónomos móviles. Por este motivo, se ha considerado como objetivo prioritario la **transparencia** de uso de la arquitectura, concretada en la sencillez y portabilidad de la misma.

La arquitectura propuesta permite implementar sistemas cooperativos mediante la interacción de procesos libremente distribuidos, por lo que se ha bautizado como ***DLA*** (***D****istributed and* ***L****ayered* ***A****rchitecture*). Mediante su utilización se ha desarrollado la infraestructura básica necesaria para incorporar la capacidad de navegación en entornos dinámicos no estructurados sobre un agente autónomo móvil genérico, aunque es posible utilizarla en otro tipo de sistemas cooperativos, incluso fuera del ámbito de la Robótica.

Cualquier arquitectura de control se puede descomponer en dos grandes niveles: una **estructura**, responsable de realizar la correcta descomposición del sistema en módulos, y un **estilo**, responsable de llevar a cabo una adecuada integración de los distintos módulos mediante los mecanismos de comunicación adecuados. Ambos niveles se encuentran estrechamente relacionados, por lo que los requisitos y prestaciones de uno de ellos delimitan los requisitos y prestaciones del otro. En el desarrollo de la arquitectura *DLA*, pues, se ha tenido que abordar convenientemente el diseño de ambos niveles.

1.3.1 Estilo de la arquitectura *DLA*

El estilo de la arquitectura de control *DLA*, referente a los mecanismos suministrados para soportar la correcta comunicación entre módulos, ha sido cuidadosamente diseñado. De entre los distintos esquemas posibles se ha optado por implementar un modelo de **memoria compartida distribuida**, por manifestar unas excelentes prestaciones de sencillez y portabilidad en el uso de la arquitectura. Dicho modelo se ha desarrollado mediante un esquema de **interacción entre procesos** con un modelo de **interconexión indirecta** mediante ***sockets***.

Se han desarrollado distintos esquemas para implementar el estilo de la arquitectura de control *DLA*, permitiendo optimizar el funcionamiento de la misma bajo diversas circunstancias. Cada esquema proporciona sus propias funcionalidades, y los recursos suministrados por todos ellos pueden ser utilizados simultáneamente de forma transparente. Su implementación ha sido gradual, de forma que se han ido mejorando sucesivamente los diferentes esquemas mediante la incorporación de nuevas características.

Los esquemas implementados han sido los siguientes:

- **Esquema distribuido básico:** basado en el esquema propuesto por Dulimarta implementa el modelo de memoria compartida distribuida que forma la base de los posteriores esquemas. Debido no obstante a su diseño altamente concurrente las prestaciones obtenidas por el esquema básico son muy superiores a las manifestadas por el esquema propuesto por Dulimarta. El principal recurso que implementa son las **conexiones**, zonas de memoria destinadas al intercambio de datos entre módulos.
- **Esquema distribuido síncrono:** la inclusión en la arquitectura de mecanismos de sincronización entre módulos permite aumentar la eficiencia en el uso de los recursos del sistema, al posibilitar la activación y desactivación selectiva de módulos. El principal recurso que implementa son los **buzones**, mecanismos de sincronización entre módulos.
- **Esquema local síncrono:** en aquellas circunstancias extremas donde todos los módulos del sistema deben ser ubicados en la misma máquina este esquema es el más eficiente. El principal recursos que implementa son las conexiones y buzones **locales**, destinados a ser utilizados en un escenario local.

El estilo implementado por la arquitectura de control *DLA* constituye una alternativa muy potente y robusta, manifestando unas prestaciones de sencillez y portabilidad muy superiores a las principales arquitecturas de control híbridas existentes.

1.3.2 Estructura de la arquitectura *DLA*

La estructura de la arquitectura *DLA*, referente a la descomposición del sistema en módulos, sigue una filosofía **híbrida** donde coexisten comportamientos reactivos y comportamientos deliberados. El sistema de navegación desarrollado se basa en la implementación gradual de diferentes capacidades cada vez más complejas, según la clasificación realizada por la Jerarquía de Navegación. De esta forma se han obtenido distintos niveles de navegación con diferentes capacidades:

- **Navegación Reactiva:** responsable de la evitación de obstáculos fijos y/o móviles del entorno durante la navegación. Este nivel se corresponde con el grupo de Navegación Local de la Jerarquía de Navegación, el cual desarrolla los niveles más básicos de navegación: Búsqueda, Seguimiento, Apuntado y Guiado.

- **Navegación Planificada:** responsable de incorporar procesos de planificación de caminos para aumentar la eficiencia del proceso de navegación. Este nivel se corresponde con el nivel de Respuesta Inducida del grupo de Búsqueda de Caminos de la Jerarquía de Navegación.
- **Navegación Topológica:** responsable de utilizar descripciones más abstractas del entorno que permitan realizar procesos de planificación de rutas más complejos. Este nivel se corresponde con los niveles de Navegación Topológica y Navegación por Reconocimiento del grupo de Búsqueda de Caminos de la Jerarquía de Navegación.

La utilización de distintos niveles de navegación en el agente manifiesta importantes ventajas, pues además de facilitar una implementación gradual, permite seleccionar el nivel de navegación más adecuado en función de los recursos disponibles en el agente y posibilita un diseño independiente de los distintos módulos utilizados en cada nivel.

La descomposición realizada del sistema se basa en una estructuración en **capas**, pues permite aumentar la modularidad y flexibilidad al incorporar progresivamente nuevas capas con nuevos comportamientos sobre los ya existentes. Cada una de estas capas está implementada mediante diversos módulos, y entre ellas se encuentra una capa que incorpora los comportamientos reactivos del sistema y una capa que agrupa todos los procesos deliberados de planificación, como corresponde a una arquitectura de control híbrida.

1.3.3 Parámetros de la arquitectura *DLA*

Por último, es importante indicar que las características exhibidas por cualquier arquitectura de control delimitarán en gran medida las prestaciones finales obtenidas en el agente autónomo móvil desarrollado. En este sentido, los **parámetros** más importantes que caracterizan el funcionamiento de la arquitectura *DLA* son:

- **Sencillez:** es extremadamente sencilla de utilizar, requiriendo únicamente la utilización de una librería.
- **Escalabilidad:** es totalmente escalable, posibilitando la implementación de sistemas distribuidos.
- **Extensibilidad:** permite la ampliación del sistema tanto a nivel hardware, permitiendo la incorporación de nuevos recursos en la plataforma robótica, como a nivel software, incorporando directamente nuevos comportamientos y funcionalidades en el sistema.
- **Portabilidad:** es altamente portable tanto a nivel hardware, posibilitando la migración directa entre distintas plataformas robóticas, como a nivel software, permitiendo la utilización de distintos lenguajes y plataformas en el desarrollo de los módulos.
- **Flexibilidad:** es muy flexible en varios niveles, permitiendo el intercambio de cualquier tipo de datos entre módulos y la reestructuración dinámica de los módulos en el sistema.

- **Depuración:** incorpora facilidades de depuración muy potentes, permitiendo monitorizar y modificar en tiempo real todos los datos contenidos en el sistema.

- **Eficiencia:** es muy eficiente gracias a los distintos esquemas implementados, permitiendo optimizar el uso de los recursos disponibles y minimizando la carga que el uso de la arquitectura tiene sobre el sistema.

En resumen, se puede concluir con que se ha obtenido una arquitectura de control muy versátil. Mediante su utilización se ha desarrollado una infraestructura básica para incorporar la capacidad de navegación en entornos dinámicos no estructurados a un agente autónomo móvil, permitiendo una muy sencilla adaptación a nuevos agentes y la expansión de sus capacidades con nuevos comportamientos más avanzados.

1.4 Capas Física, de Representación y de Comando

Una vez implementada la arquitectura de control propuesta y realizada la descomposición del sistema en capas y módulos, se ha procedido al desarrollo de los mismos. En primer lugar se ha abordado el diseño de la Capa Física, la Capa de Representación y la Capa de Comando, necesarias para el correcto funcionamiento del sistema.

La Capa Física controla los recursos disponibles de la plataforma robótica. Para aumentar la generalidad del sistema desarrollado y sus posibilidades de aplicación a distintos agentes se han utilizado únicamente recursos básicos en la plataforma robótica: un sistema de posicionamiento basado en odometría y un sistema sensorial compuesto por sensores sonar. Obviamente es posible incorporar otro tipo de sensores si se desea, si bien esto no es imprescindible.

Respecto a la localización del agente, se ha implementado un sencillo método mediante el filtro de Kalman para corregir los errores del sistema de odometría, aunque es preciso matizar que dicho método es bastante simple y puede ser ampliamente mejorado. Queda pendiente de futuros trabajos incorporar esquemas de localización más robustos y precisos, incorporación sencilla de realizar gracias a la elevada modularidad de la arquitectura *DLA*, no requiriendo la modificación del resto de módulos del sistema.

La Capa de Representación que genera un modelo del entorno utiliza un **mapa métrico probabilístico**, debido a la sencillez en su generación y actualización, lo que redunda en unas excelentes características temporales, y a su capacidad de manejar la incertidumbre de los sensores sonar mediante técnicas estadísticas. No obstante, este tipo de representaciones poseen un claro compromiso entre el grado de resolución deseado y la cantidad de datos contenidos en la representación del entorno, por lo que no son apropiadas para la representación de grandes entornos.

La Capa de Comando actúa de interfaz con el usuario, realizando además la activación y desactivación selectiva de los distintos módulos del sistema en función de la tarea a ejecutar.

1.5 Capa de Navegación

La Capa de Navegación integra todos los comportamientos reactivos de navegación del sistema, y es responsable de desarrollar el nivel de Navegación Reactiva. Se ha optado por implementarla mediante un **esquema basado en aprendizaje**, pues permiten particularizar su comportamiento para distintos escenarios e incorporan directamente restricciones cinemáticas en el movimiento del agente, facilitando su adaptación a distintos agentes con diferentes características dinámicas.

Dentro de los esquemas basados en aprendizaje existen distintos paradigmas, siendo los más relevantes el razonamiento basado en reglas, las redes neuronales y el razonamiento basado en casos. El **razonamiento basado en reglas** es complejo de aplicar si no se puede determinar un conjunto de reglas que modelen el comportamiento del sistema, que suele ser precisamente el caso de la navegación reactiva, donde es complejo abarcar todas las situaciones posibles durante el proceso de navegación.

Las **redes neuronales** constituyen una solución más adaptable al problema concreto de la navegación reactiva, pues a partir de un proceso de entrenamiento se permite modelar el comportamiento del sistema sin conocer exactamente su naturaleza. No obstante, las redes neuronales intentan extraer un único modelo de entre los distintos patrones de entrenamiento suministrados al sistema, si bien esta situación no produce resultados adecuados cuando no todo el comportamiento del sistema es generalizable con un único modelo. Además, para obtener un funcionamiento lo más correcto posible suele ser necesario disponer de un elevado número de patrones de entrenamiento. Por último, en las redes neuronales no es posible disponer recuperar los patrones de entrenamiento utilizados a partir de la configuración del sistema ni saber la importancia que cada uno de esos patrones tiene en una determinada solución, complicando la depuración del sistema y la optimización del conocimiento que posee.

El **razonamiento basado en casos** (*CBR*) trata de utilizar experiencias pasadas en la resolución de problemas actuales, sin intentar generalizar mediante ningún tipo de modelo el conjunto de experiencias almacenadas. Esta es la principal ventaja de este paradigma frente a las redes neuronales, ya que permite incorporar distintos comportamientos en el sistema. Así por ejemplo, es posible enseñar al agente a navegar cerca de una pared cuando se encuentre en un pasillo y a alejarse de ella cuando se encuentre en una habitación amplia, mostrando dos comportamientos totalmente distintos aun cuando la distancia a la pared sea la misma en ambas situaciones. Además, este paradigma puede comenzar a operar con un muy escaso entrenamiento inicial, adquiriendo un mayor conocimiento conforme el sistema va resolviendo nuevos problemas. Por último, este paradigma permite recuperar en todo momento las experiencias que forman el conocimiento del sistema, así como identificar la experiencia particular que se está utilizando en una determinada solución, factores que simplifican enormemente el proceso de depuración del sistema y la optimización del conocimiento que posee.

Así pues, tras analizar las distintas alternativas se ha escogido el paradigma del razonamiento basado en casos por su mayor versatilidad. Su implementación se basa en la correcta representación de las distintas experiencias que se quieren almacenar en el sistema. Dicha repre-

sentación se materializa mediante una estructura conocida como **caso**, la cual debe poseer la información necesaria para describir por completo un problema y la solución adoptada en su resolución. Puesto que se pretende implementar un esquema reactivo, toda la información que puede ser incorporada en cada caso se corresponde con la que se percibe del entorno de forma inmediata, formada básicamente por la posición actual del agente y por la situación de los obstáculos en el entorno.

Todos estos casos que forman el conocimiento del sistema son incorporados en lo que se conoce como **base de casos**. La estructura que tenga esta base de casos determinará en gran medida las prestaciones alcanzadas en el sistema, por lo que su correcta elección es bastante importante en función de la aplicación que se desee implementar. Las principales estructuras existentes para implementar una base de casos en un sistema *CBR* son la estructura plana y la estructura jerárquica. La **estructura plana** es más sencilla y siempre proporciona el caso más parecido al actual, aunque puede derivar en tiempos de búsqueda elevados si el número de casos del sistema es elevado. La **estructura jerárquica** es más compleja y no siempre proporciona el caso más parecido al actual, aunque exhibe bajos tiempos de búsqueda aun con un elevado el número de casos. Debido a su sencillez y por devolver siempre el caso más parecido al actual en el sistema se ha escogido una estructura plana, implementando un método de optimización de la base de casos que permite mantener acotado el número de casos en el sistema.

Respecto al esquema de aprendizaje, se han implementado las dos estrategias básicas de este tipo de sistemas:

- **Aprendizaje por observación:** consistente en la incorporación de un conjunto inicial de casos válidos en el sistema, los cuales se pueden obtener mediante un proceso de entrenamiento realizado por un operador humano o por el esquema analítico de los Campos Potenciales. El entrenamiento a partir de un **operador humano** permite particularizar la operación del sistema de forma supervisada en distintas circunstancias, mientras se manejan implícitamente restricciones cinemáticas. Por su parte, el entrenamiento mediante **Campos Potenciales** permite realizar un aprendizaje automático no supervisado del sistema. Además, se permite realizar el entrenamiento en cualquiera de los dos casos sobre entornos simulados, aunque la operación del sistema se lleve a cabo en entornos reales. De esta forma se facilita el proceso de entrenamiento inicial al poder configurar circunstancias particulares complejas de encontrar en el mundo real.
- **Aprendizaje por experiencia:** consistente en la incorporación de las soluciones generadas por el propio sistema ante situaciones desconocidas. Existen distintas alternativas posibles para realizar el aprendizaje por experiencia en el sistema, habiéndose escogido la estrategia de **aprendizaje positivo**, la cual se combina posteriormente con el proceso de optimización de la base de casos.

En el sistema propuesto se ha implementado un **proceso de optimización** de la base de casos mediante un algoritmo de segmentación en clases que permite agrupar los casos más parecidos de la base de casos y representarlos mediante un prototipo. Este proceso de optimización mejora la operación del sistema en varios niveles:

- Mantiene acotado el número de casos de la base de casos, requisito indispensable al haber utilizado una estructura plana en la misma.
- Permite un mayor nivel de abstracción del conocimiento almacenado en el sistema, agrupando casos parecidos que se corresponden con situaciones semejantes y reduciendo así la sensibilidad del sistema ante errores esporádicos de los sensores sonar.
- Favorece los casos más eficientes y penaliza a los menos eficientes, implementando un sistema de olvido que permite descartar casos erróneos que fueron almacenados en el sistema.
- Permite optimizar la operación del sistema según los objetivos perseguidos en el proceso de navegación, considerando criterios como **suavidad**, **distancia** y **seguridad** en la navegación.

En conclusión, se ha obtenido un sistema de navegación reactivo muy robusto y flexible, capaz de aprender distintos comportamientos e incorporar restricciones cinemáticas en la operación del agente. Por último, muestra unas excelentes prestaciones temporales, siendo capaz de operar prácticamente en tiempo real como requiere cualquier esquema reactivo.

1.6 Capa de Planificación

La Capa de Planificación integra todos los comportamientos deliberados de planificación del sistema, y es responsable de desarrollar los niveles de Navegación Planificada y de Navegación Topológica. Para desarrollar correctamente estos niveles es necesario incorporar en el sistema un proceso de planificación de caminos y un proceso de planificación de rutas.

El proceso de **planificación de caminos** diseñado permite calcular un camino libre de obstáculos para alcanzar el destino final. Actúa dirigiendo los comportamientos reactivos de navegación implementados en el sistema, permitiendo una operación más eficiente y correcta de los mismos sin degradar sus características temporales. En este sentido, este nivel de planificación es capaz de evitar las trampas de mínimos locales y de optimizar parámetros globales del proceso de navegación como la distancia total recorrida.

Existen diferentes alternativas para implementar un planificador de caminos sobre un mapa métrico probabilístico, habiendo escogido finalmente por su simplicidad y eficiencia la aproximación de los **Campos Potenciales**. El método propuesto se basa en una combinación de operaciones como **propagación de ondas** y **crecimiento de obstáculos**, exhibiendo unas excelentes prestaciones: longitud cuasi-óptima, sencillo de seguir, suave, rápido y seguro.

Una vez generado el camino sobre el mapa métrico se realiza una descomposición del mismo determinando sus puntos de inflexión, para lo cual se utiliza el análisis del **código cadena** del camino generado. Gracias a la sencillez del camino calculado este análisis se realiza muy rápidamente.

Por último, se ha diseñado cuidadosamente la adecuada integración del planificador de caminos implementado con el sistema reactivo disponible en el sistema, con el objetivo de no alterar las prestaciones temporales del mismo. Para ello, una vez generado el camino a seguir se le envían sucesivamente a la Capa de Navegación como destinos intermedios los puntos de inflexión extraídos del camino, de forma que el sistema seguirá aproximadamente el camino calculado. Este camino se mantiene hasta que se detecten cambios significativos en el entorno, momento en el cual se vuelve a calcular otro camino más acorde a las nuevas circunstancias.

El proceso de **planificación de rutas** diseñado permite descomponer el entorno en una serie de regiones conectadas entre sí, calculando posteriormente una ruta formada por las regiones a atravesar hasta alcanzar el destino final. El planificador de rutas controla directamente al planificador de caminos implementado, y al incorporar un mayor nivel de abstracción de la realidad permite planificar tareas más complejas, como determinar caminos que visitan distintas localizaciones o que exploren completamente un entorno en el menor tiempo posible.

Para que el planificador de rutas pueda operar necesita una representación del entorno más compacta que la proporcionada por el mapa métrico, por lo que dicho planificador genera en primer lugar una representación topológica del entorno que describa las distintas regiones que posee. Existen distintas aproximaciones para generar representaciones topológicas, y al disponer de una representación métrica en el sistema se ha escogido una estrategia indirecta basada en una **estructura jerárquica piramidal**. Esta representación topológica exhibe importantes ventajas:

- Es capaz de operar con entornos total o parcialmente explorados, representando adecuadamente distintos tipos de regiones para ser utilizados por los posteriores procesos de planificación.
- Puede utilizar regiones de forma irregular para realizar la correcta descomposición del entorno.
- Muestra unas prestaciones temporales muy destacables.

Una vez generado el mapa topológico del entorno se ha utilizado el algoritmo de **Dijkstra** para determinar la ruta con las regiones a atravesar hasta el destino final. Dicho algoritmo es un algoritmo óptimo muy eficiente, aunque sus prestaciones temporales se degradan considerablemente si aumenta el número de regiones del entorno a analizar. No obstante, gracias a la elevada compactación de la información llevada a cabo por el mapa topológico el tiempo de cómputo empleado por este algoritmo se mantiene bastante bajo.

Por último, se ha diseñado cuidadosamente la adecuada integración del planificador de rutas implementado en el sistema. Para ello, una vez generada la ruta a seguir se le envían sucesivamente al planificador de caminos las regiones a visitar, de forma que calcule un camino que pueda ser seguido por el sistema reactivo para alcanzar dichas regiones. Para acelerar este proceso se ha limitado la zona del mapa métrico que debe procesar el planificador de caminos a las regiones actual y siguiente de la ruta calculada. La ruta calculada se mantiene hasta que se

detecten cambios significativos en el entorno, momento en el cual se vuelve a calcular otra ruta más acorde a las nuevas circunstancias.

Ambos procesos de planificación son complementarios, por lo que su adecuada integración permite seleccionar en cada momento la estrategia de navegación más adecuada en función de los objetivos perseguidos con el proceso de navegación:

- **Planificador de caminos:** la operación llevada a cabo por el planificador de caminos permite que la trayectoria final recorrida por el agente sea más óptima, aunque posee el importante inconveniente de que sus prestaciones temporales se degradan considerablemente conforme aumenta el tamaño del entorno representado en el mapa métrico y no permite comportamientos más avanzados de planificación.
- **Planificador de rutas:** la operación llevada a cabo por el planificador de rutas permite acelerar el proceso de navegación en grandes entornos y es capaz de exhibir comportamientos avanzados de planificación, como establecer rutas hacia varias ubicaciones o estrategias eficientes de exploración de entornos. No obstante, presenta el inconveniente de que la trayectoria final recorrida por el agente no es muy óptima.

En conclusión, se ha logrado una adecuada integración entre los distintos procesos deliberados de planificación y los procesos reactivos del sistema, permitiendo un considerable aumento en la eficiencia del proceso de navegación sin degradar las prestaciones temporales del mismo.

1.7 Resultados

Tras haber concluido el diseño completo del sistema, se han realizado numerosas pruebas sobre el mismo, no sólo para verificar el correcto funcionamiento de las distintas funcionalidades implementadas sino también para caracterizar mediante parámetros cuantificables su operación.

1.7.1 Estilo de la arquitectura *DLA*

El estilo de la arquitectura *DLA* ha sido profundamente probado. Para ello se han analizado detalladamente las prestaciones temporales obtenidas por el sistema en función de los esquemas utilizados:

- **Esquema distribuido básico:** las pruebas realizadas sobre este esquema han corroborado que la arquitectura *DLA* es bastante superior al esquema propuesto por Dulimarta en condiciones reales de funcionamiento.
- **Esquema distribuido síncrono:** las pruebas realizadas sobre este esquema han demostrado que la operación del sistema puede ser mejorada con la incorporación de mecanismos de sincronización que posibiliten la activación y desactivación selectiva de los distintos módulos.

- **Esquema local síncrono:** las pruebas realizadas sobre este esquema han puesto de manifiesto la importante mejora conseguida en el sistema utilizando mecanismos de memoria compartida para la interconexión directa de los distintos módulos.

Las prestaciones obtenidas con estas pruebas, además de verificar la sucesiva optimización conseguida con los distintos esquemas utilizados, han permitido cuantificar la velocidad de operación del sistema en condiciones reales de funcionamiento. Estos parámetros dependen obviamente de los recursos utilizados y de la distribución de los módulos entre los mismos, pero utilizando una distribución adecuada se pueden conseguir en el caso más restrictivo de Navegación Topológica retardos del orden de 100 ms para el sistema reactivo y del orden de 140 ms para el sistema total, lo que implica que a una velocidad del agente de unos 300 mm/s se recorren aproximadamente 3 cm antes de reaccionar frente a un nuevo estímulo y 4 cm antes de generar los planes adecuados ante un nuevo comando.

Estas prestaciones temporales son realmente destacables, y han sido posibles gracias a la confluencia de dos factores: las sucesivas optimizaciones realizadas sobre el estilo de la arquitectura *DLA*, reduciendo sucesivamente los tiempos empleados en el intercambio de datos entre los distintos módulos del sistema, y la correcta selección y optimización de los algoritmos utilizados para implementar los distintos módulos del sistema.

1.7.2 Estructura de la arquitectura *DLA*

Las pruebas realizadas sobre la estructura de la arquitectura *DLA* han permitido verificar que los distintos niveles de la Jerarquía de Navegación han sido correctamente implementados. Para ello se han comprobado las capacidades mostradas por los distintos niveles de navegación, caracterizando la operación del sistema y derivando las siguientes **conclusiones**:

- **Navegación Reactiva:**
  - **Flexibilidad:** la operación del sistema es muy flexible, permitiendo realizar y combinar entrenamientos muy diversos. Así por ejemplo, es posible realizar una etapa de entrenamiento simulado mediante esquemas analíticos como los Campos Potenciales para posteriormente incluir algunas circunstancias particulares mediante un entrenamiento real a partir de uno o varios operadores humanos.
  - **Capacidad de navegación:** la funcionalidad del sistema muestra una capacidad de navegación superior a la exhibida por esquemas analíticos como los Campos Potenciales, aun utilizando dicho esquema en el proceso de entrenamiento, debido a la continua incorporación de nuevas experiencias cada vez más optimizadas que van mejorando la operación del sistema.
  - **Cantidad de conocimiento:** se ha corroborado que tanto un escaso como un excesivo conocimiento suele ser perjudicial para la operación del sistema. Un escaso conocimiento plantea la resolución de excesivas situaciones desconocidas, las cuales

pueden proporcionar soluciones no muy eficientes en determinadas circunstancias. Un excesivo conocimiento suele derivar en efectos no deseados como oscilaciones, al alternar el sistema entre casos muy semejantes con soluciones muy diferentes. Los efectos negativos del escaso conocimiento se resuelven con un mayor nivel de entrenamiento en el sistema, o incluso con un mayor tiempo de operación para incluir un número suficiente de nuevas experiencias en el mismo. Los efectos negativos del excesivo conocimiento se resuelven gracias a la implementación del mecanismo de optimización de la base de casos, el cual permite compactar la información del sistema eliminando pequeñas discordancias entre casos y aumentando la eficiencia de los casos almacenados.

- **Versatilidad:** el paradigma del razonamiento basado en casos es muy versátil, pues modificando ligeramente la representación del caso se ha posibilitado la implementación de comportamientos de navegación más complejos como la operación en entornos multiagente. Esto demuestra que el razonamiento basado en casos es capaz de incorporar prácticamente cualquier tipo de conocimiento en el sistema, siempre que se realice una adecuada representación de las distintas experiencias en forma de caso. En este sentido, es necesario resaltar que quizás el aspecto más crítico en la implementación de un sistema mediante el paradigma del razonamiento basado en casos radica precisamente en la correcta selección de la información a incorporar en cada caso, pues las prestaciones finales exhibidas por el sistema dependen no sólo de la información almacenada en el mismo sino también de la sensibilidad que muestre la operación del sistema ante la variación de dicha información.
- **Operación real:** el sistema implementado es muy robusto, manejando adecuadamente los errores en las lecturas de los sensores sonar que caracterizan la operación en entornos reales. En primer lugar, si se ha realizado un entrenamiento en entornos reales y se han incorporado casos erróneos en el sistema, estos casos son sucesivamente eliminados del sistema gracias al proceso de optimización de la base de casos desarrollado, mediante la fusión adecuada con casos semejantes carentes de dichos errores. En segundo lugar, la operación continua del sistema garantiza que el efecto que estos errores esporádicos puedan tener en el mismo es prácticamente despreciable, al ser rápidamente corregidos en cuanto desaparecen con la generación de nuevos casos.

- **Navegación Planificada:**

  - **Eficiencia:** la eficiencia del proceso de navegación se ve considerablemente reforzada, pues el proceso de planificación es capaz de producir caminos suaves y seguros que resuelven adecuadamente las distintas trampas de mínimos locales del entorno.
  - **Operación real:** La operación del proceso de planificación sigue siendo correcta aun en presencia de lecturas erróneas de los sensores sonar, las cuales tienden a producir la generación de mapas métricos difusos. En este sentido, se ha corroborado que los caminos calculados suelen presentar inicialmente trayectorias algo más largas y no tan suaves, aunque esta situación se corrige cuando el proceso de exploración ha avanzado lo suficiente como para eliminar en su mayor parte estas lecturas erróneas del mapa métrico, gracias al mecanismo de generación probabilístico del mismo.

- **Navegación Topológica:**
  - **Capacidad de navegación:** las capacidades alcanzadas con el proceso de navegación aumentan, pues el proceso de planificación es capaz de generar rutas completas con las regiones a atravesar del entorno utilizando tanto las regiones exploradas como las regiones no exploradas.
  - **Operación real:** la operación del proceso de planificación sigue siendo correcta aun en presencia de lecturas erróneas de los sensores sonar, las cuales tienden a producir la generación de mapas métricos difusos. En este sentido, se ha comprobado que la utilización de mapas métricos difusos producen mapas topológicos compuestos por un mayor número de regiones, aunque esta situación se corrige cuando el proceso de exploración ha avanzado lo suficiente como para eliminar en su mayor parte estas lecturas erróneas del mapa métrico, gracias al mecanismo de generación probabilístico del mismo.

Todas las pruebas realizadas demuestran que el sistema desarrollado es muy robusto y flexible, exhibiendo además unas prestaciones temporales realmente destacables. Así mismo, es muy sencillo de utilizar y ha sido concebido para garantizar una alta portabilidad entre distintas plataformas. Su elevada modularidad permite una muy sencilla ampliación y modificación de los comportamientos implementados, posibitando la directa sustitución si se desea de algunos de los algoritmos desarrollados por otros más complejos o avanzados. Así por ejemplo, se pueden incorporar otras técnicas de localización más complejas y eficientes o incluso nuevos algoritmos de planificación que sustituyan o complementen a los existentes.

En conclusión, se ha cumplido satisfactoriamente el objetivo inicial de implementar una infraestructura básica para incorporar la capacidad de navegación en entornos dinámicos no estructurados a un agente autónomo móvil genérico, facilitando su posterior adaptación a otros agentes y su expansión con comportamientos más avanzados. Las capacidades de navegación incorporadas se corresponden con los niveles superiores de la Jerarquía de Navegación, los cuales manifiestan las capacidades más versátiles y avanzadas.

2 Contribuciones

Han sido numerosas las contribuciones derivadas con la realización de la presente Tesis. Se presenta a continuación un resumen de las mismas.

2.1 Publicaciones

Las distintas publicaciones relacionadas con el trabajo realizado así como los aspectos que cubren de la presente Tesis son los que se presentan a continuación:

- **Revistas:** artículos en revista relacionados con la presente Tesis:

1. **RAS02:** [PPB+02]

Título
Efficient Integration of Metric and Topological Maps for Directed Exploration of Unknown Environments
Revista
Robotics and Autonomous Systems
Resumen
Esta publicación aborda el desarrollo de esquemas avanzados de exploración de entornos mediante la utilización de mapas topológicos y algoritmos genéticos.

2. **CP02:** [PPU+02c]

Título
Survey Navigation for a Mobile Robot by Using a Hierarchical Cognitive Map
Revista
Cognitive Processing
Resumen
Esta publicación aborda la implementación de esquemas avanzados de navegación en entornos parcialmente explorados mediante la utilización de mapas topológicos, cubriendo el planificador de rutas desarrollado.

3. **IASC05:** [PPUS05]

Título
A Hybrid Path Planning Technique for Partially Unknown Indoor Environments
Revista
Intelligent Automation and Soft Computing
Resumen
Esta publicación aborda el desarrollo de esquemas avanzados de exploración de entornos mediante la utilización de mapas topológicos y algoritmos genéticos.

4. **AR06:** [UPVS+06]

Título
A Purely Reactive Navigation Scheme for Dynamic Environments Using Case-Based Reasoning
Revista
Autonomous Robots
Resumen
Esta publicación aborda la implementación de un esquema reactivo de navegación mediante el paradigma del razonamiento basado en casos, cubriendo el sistema reactivo de navegación desarrollado.

- **Capítulos:** capítulos de libro relacionados con la presente Tesis:

  1. **CISAR02:** [PPU+02a]

Título
Intelligent Navigation in Partially Unknown Indoor Environments
Libro
Computational Intelligent Systems for Applied Research
Resumen
Esta publicación aborda el desarrollo de esquemas avanzados de exploración de entornos mediante la utilización de mapas topológicos y algoritmos genéticos.

  2. **ARS03:** [UBP+03]

Título
Hierarchical Planning in a Mobile Robot for Map Learning and Navigation
Libro
Autonomous Robotic Systems. Soft Computing and Hard Computing Methodologies and Applications
Resumen
Esta publicación aborda el desarrollo completo de un sistema híbrido que muestre capacidades avanzadas de navegación en entornos dinámicos no estructurados, cubriendo la arquitectura de control, la generación del mapa métrico, el planificador de caminos y el planificador de rutas desarrollados.

- **Congresos:** actas de congreso relacionadas con la presente Tesis:

  1. **SIRS99:** [PUB+99]

Título
A Path Tracking Method for Autonomous Mobile Robots Based on Grid Decomposition
Congreso
Symposium on Intelligent Robotics Systems
Resumen
Esta publicación aborda el problema del seguimiento de caminos mediante una técnica de persecución pura, cubriendo la etapa de seguimiento del planificador de caminos desarrollado.

  2. **SIRS00:** [PPB+00]

Título
A Curvature-Based Path Tracking Method on Probabilistic Grids for Autonomous Mobile Robots
Congreso
Symposium on Intelligent Robotics Systems
Resumen
Esta publicación aborda el problema del seguimiento de caminos mediante la extracción de los puntos de inflexión utilizando una función de curvatura, cubriendo la etapa de seguimiento del planificador de caminos desarrollado.

3. **SIRS01:** [LMP+01]

Título
3D Reconstruction of a Static Indoor Enviroment by Fusion of Sonar and Video Data
Congreso
Symposium on Intelligent Robotics Systems
Resumen
Esta publicación aborda el problema de generación de entornos *3D* utilizando la transformada de Hough sobre un mapa métrico probabilístico, incorporando información de textura captada mediante un sistema de visión.

4. **FLINS02:** [PPU+02b]

Título
Intelligent Navigation in Partially Unknown Indoor Environments
Congreso
International FLINS Conference on Computational Intelligent Systems for Applied Research
Resumen
Esta publicación aborda el desarrollo de esquemas avanzados de exploración de entornos mediante la utilización de mapas topológicos y algoritmos genéticos.

5. **IASTED03:** [UPSVS03a]

Título
A CBR Based Pure Reactive Layer for Autonomous Robot Navigation
Congreso
IASTED International Conference on Artificial Intellegence and Soft Computing
Resumen
Esta publicación aborda la implementación de un esquema reactivo de navegación mediante el paradigma del razonamiento basado en casos, cubriendo el sistema reactivo de navegación desarrollado.

6. **IAT03:** [UPSVS03b]

Título
A Hybrid Architecture for Autonomous Navigation in Dynamic Environments
Congreso
IEEE/WIC International Conference on Intelligent Agent Technology
Resumen
Esta publicación aborda el desarrollo de un sistema híbrido que muestre capacidades avanzadas de navegación en entornos dinámicos no estructurados, cubriendo el sistema reactivo de navegación, la generación del mapa métrico, el planificador de caminos y el planificador de rutas desarrollados.

7. **ISORA04:** [PUSVS04]

Título
A CBR Strategy for Autonomous Reactive Navigation Learning
Congreso
International Symposium on Robotics and Applications
Resumen
Esta publicación aborda la implementación de un esquema reactivo de navegación mediante el paradigma del razonamiento basado en casos, cubriendo el sistema reactivo de navegación desarrollado.

8. **IAT04:** [UPS04]

Título
A Time Stamp Control Strategy for CBR Based Reactive Navigation in Dynamic Environments with Priorities
Congreso
IEEE/WIC International Conference on Intelligent Agent Technology
Resumen
Esta publicación aborda la implementación de un esquema reactivo de navegación multiagente basado en prioridades mediante el paradigma del razonamiento basado en casos, cubriendo el sistema reactivo de navegación desarrollado.

9. **RAM04:** [PVN+04]

Título
A Hough-based Method for Concurrent Mapping and Localization in Indoor Environments
Congreso
IEEE Conference on Robotics, Automation and Mechatronics
Resumen
Esta publicación aborda el problema de la localización en entornos interiores mediante la utilización de la transformada de Hough y el filtro de Kalman.

10. **WAF05:** [PUS05]

Título
Arquitectura Híbrida para Navegación Mediante Esquemas de Memoria Compartida Distribuida
Congreso
Workshop en Agentes Físicos
Resumen
Esta publicación aborda la implementación de una arquitectura de control mediante un esquema de memoria compartida distribuida para el desarrollo de un sistema híbrido que muestre capacidades avanzadas de navegación en entornos dinámicos no estructurados, cubriendo la arquitectura de control desarrollada.

11. **VTC06:** [VMPU+06]

Título
Hybrid Navigation Guidance for Intelligent Mobiles
Congreso
IEEE Vehicular Technology Conference
Resumen
Esta publicación aborda el problema de la localización en entornos exteriores mediante la utilización de un dispositivo *GPS* y el filtro de Kalman.

2.2 Sistemas desarrollados

Se resumen a continuación los principales sistemas en los que ha intervenido el trabajo realizado en la presente Tesis:

- Navegación en entornos interiores sobre las plataformas robóticas *Nomad 200*, *Pioneer P2AT* y *Spherik* [1] [PPU+02c, UPSVS03a, UBP+03, UPSVS03b, PUSVS04, UPS04, PUS05, UPVS+06].
- Navegación en entornos exteriores sobre la plataforma robótica *Pioneer P2AT* [VMPU+06].
- Exploración completa de entornos mediante comportamientos avanzados [PPU+02b, PPU+02a, PPB+02, PPUS05].
- Sistemas de localización para interiores [PVN+04] y exteriores [VMPU+06].
- Generación de entornos virtuales *3D* con información de texturas [LMP+01].
- Desarrollo de comportamientos avanzados mediante *CBR* para la plataforma robótica *Aibo* [HUP+05]

3 Líneas futuras

Se describen a continuación las principales líneas futuras de trabajo que pueden ampliar el contenido de la presente Tesis.

3.1 Estilo de la arquitectura *DLA*

El estilo de la arquitectura *DLA* puede ser mejorado y ampliado en algunos aspectos, entre los que se destacan:

- Es deseable expandir el uso de la arquitectura a **nuevos lenguajes y plataformas**, para poder extender su uso y abarcar el mayor número de sistemas posible. Para ello, es necesario trascribir la librería *DLALibrary* convenientemente, aspecto sencillo de realizar en los esquemas distribuidos basados en *sockets* pero un poco más complejo en el esquema local basado en memoria compartida y semáforos.
- Aunque no ha sido necesario durante el trabajo realizado en la presente Tesis, es previsible que en un futuro se deban incorporar **nuevas funcionalidades** que complementen a las conexiones y buzones disponibles, para dotar de mayor flexibilidad la interacción entre los

[1] Plataforma robótica desarrollada en la *Facultat d'Informàtica de Barcelona* de la *Universitat Politècnica de Catalunya*.

distintos módulos del sistema. Entre estos mecanismos se encuentran el pase de mensajes, la especificación de prioridades y la gestión de eventos.

- La continua aparición de nuevos lenguajes de programación facilita cada vez el desarrollo de sistemas cooperativos [BF02, OC03]. Entre ellos se encuentra ***CORBA*** (*Common Object Request Broker Architecture*), que es un lenguaje orientado a objetos concebido para permitir una transparente distribución del código entre diversas máquinas. La utilización de *CORBA* para desarrollar la arquitectura *DLA* puede facilitar la incorporación de numerosas funcionalidades en el sistema, ya proporcionadas mediante dicho lenguaje. No obstante, debido a que *CORBA* utiliza mecanismos de interacción entre módulos bastante más abstractos y complejos que los utilizados en la arquitectura *DLA*, su utilización puede degradar las prestaciones temporales del sistema. Un detallado estudio de esta alternativa debe ser realizada.

- El esquema distribuido síncrono de la arquitectura *DLA* posibilita la distribución dinámica de módulos entre distintas máquinas. Una nueva optimización del sistema puede consistir en implementar un mecanismo de monitorización del uso de los distintos recursos disponibles en el sistema, de forma que se pueda realizar una **distribución automática de módulos** en función de la carga instantánea del mismo.

- Se puede expandir y potenciar el uso de la arquitectura *DLA*, transformándola en una herramienta de libre distribución con licencia ***GPL*** (*General Public License*).

3.2 Estructura de la arquitectura *DLA*

La estructura de la arquitectura *DLA* es lo suficientemente modular como para permitir una sencilla ampliación y modificación de los distintos comportamientos desarrollados. En este sentido el sistema puede ser mejorado en los siguientes aspectos:

- Incorporación de otros sensores y actuadores más sofisticados en la plataforma robótica para ampliar el funcionamiento del sistema. En este sentido, un dispositivo **láser** puede mejorar enormemente la operación en entornos reales al presentar una precisión mucho mayor que los sensores sonar utilizados, eliminando lecturas erroneas y generando mapas métricos mucho más definidos. Un dispositivo ***GPS*** puede habilitar al agente para una correcta operación en exteriores. Así mismo, un sistema de cámaras que permita incorporar nuevos comportamientos basados en **visión** y un **brazo robótico** que permita realizar tareas de manipulación pueden ser también desables. La incorporación de este equipamiento no modificaría ninguno de los módulos implementados en el sistema, únicamente el correspondiente servidor hardware *RobotServer* que controla la plataforma robótica.

- El sistema de **localización** puede ser mejorado con algoritmos más complejos y eficientes. En este sentido, nuevas técnicas de localización local pueden mejorar la corrección de los errores cometidos en el sistema de odometría, mientras que técnicas de localización global pueden habilitar la ubicación del agente en grandes entornos.

- Integración de **nuevos comportamientos reactivos** que permitan por ejemplo el seguimiento de personas, comportamiento interesante en determinadas aplicaciones de guiado como puede ser el transporte de mercancías o el apoyo a personas discapacitadas.
- Automatización del funcionamiento del **sistema *CBR***, realizando automáticamente el mecanismo de optimización de la base de casos que permita integrar directamente nuevos casos en función de su eficiencia.
- Desarrollo de un **controlador** que permita combinar y arbitrar entre distintos comportamientos reactivos.
- Integración de **nuevos procesos de planificación** que permitan desarrollar comportamientos más avanzados, como por ejemplo el establecimiento de rutas completas parar visitar varias ubicaciones mientras se minimiza la distancia total recorrida o la exploración completa de entornos en el menor tiempo posible.
- Ampliación del uso del mapa topológico para abarcar **grandes entornos**, mucho mayores que la representación realizada mediante el mapa métrico. De esta forma es posible planificar grandes rutas que permitan al agente dirigirse de forma eficiente hacia destinos lejanos.
- Desarrollo de un **interfaz de usuario** más potente, habilitando por ejemplo la utilización de comandos de voz y la generación de representaciones *3D* del entorno. También se puede desarrollar este interfaz en un entorno *Web*, de forma que cualquier persona pueda acceder remotamente al mismo y especificar los comandos de navegación que desee, visualizando los resultados obtenidos con la ejecución de dichos comandos.
- Incorporación de **inteligencia de alto nivel** en el sistema que utilice su capacidad de navegación para el desarrollo de tareas complejas.
- Se pueden buscar **aplicaciones reales** en las que utilizar el sistema desarrollado, como por ejemplo realización de tareas logísticas en oficinas, guía en museos o apoyo a personas con discapacidades.

Bibliografía

[Aam91] Agnar Aamodt. *A Knowledge Intensive Approach to Problem Solving and Sustained Learning.* Tesis Doctoral, University of Trondheim, Norwegian Institute of Technology, 1991.

[ACF+98] Rachid Alami, Raja Chatila, Sara Fleury, Malik Ghallab, and Felix Ingrand. An architecture for autonomy. *International Journal of Robotics Research*, 17(4):315–337, 1998.

[ACF+00] Rachid Alami, Raja Chatila, Sara Fleury, Matthieu Herrb, Felix Ingrand, Maher Khatib, Benoit Morisset, Philippe Moutarlier, and Thierry Siméon. Around the lab in 40 days ... In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 88–94, San Francisco, California, April 2000.

[AdRMF91] Angelo Arleo, José del R. Millán, and Dario Floreano. Efficient learning of variable-resolution cognitive maps for autonomous indoor navigation. *IEEE Transactions on Robotics and Automation*, 15(6):990–1000, 1991.

[Alb91] James S. Albus. Outline for a theory of intelligence. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(3):473–509, 1991.

[Alt88] R. Alterman. Adaptive planning. *Cognitive Science*, 12:393–422, 1988.

[Alt89] Klaus D. Althoff. Knowledge acquisition in the domain of cnc machine centers: The moltke approach. In *Proceedings of the 3rd European Workshop on Knowledge-Based Systems*, pages 180–195, Paris, France, July 1989.

[AML87] James S. Albus, Harry G. McCain, and Ronald Lumia. Nasa/nbs standard reference model for telerobot control system architecture (nasrem). Technical Report 1235, National Institute of Standard and Technology (NIST), Gaithersburg, Maryland, 1987.

[And83] John R. Anderson. *The Architecture of Cognition.* Harvard University Press, Cambridge, Massachusetts, 1983.

[And91] Gregory R. Andrews. Paradigms for process interaction in distributed programs. *ACM Computing Surveys*, 23(1):49–90, 1991.

[Ang89] Colin Angle. Tooth docs: the paper. 1989.

[AP94] Agnar Aamodt and Enric Plaza. Case based reasoning: Foundational issues, methodological variations, and system approaches. *Artificial Intelligence Communications*, 7(1):39–52, 1994.

[App00] Nearness Diagram Navigation (ND): A New Real Time Collision Avoidance Approach. Javier minguez and luis montano. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2094–2100, San Francisco, California, April 2000.

[Ark89] Ronald C. Arkin. Motor schema-based mobile robot navigation. *International Journal of Robotics Research*, 8(4):92–112, 1989.

[Ark90] Ronald C. Arkin. Integrating behavioral, perceptual and world knowledge in reactive navigation. *Robotics and Autonomous Systems*, 6:105–122, 1990.

[Ark98] Ronald C. Arkin. *Behavior-Based Robotics*. MIT Press, Cambridge, Massachusetts, 1998.

[Ash91] Kevin D. Ashley. *Modeling Legal Arguments: Reasoning with Cases and Hypotheticals*. MIT Press, Cambridge, Massachusetts, 1991.

[BA95] L. Karl Branting and D. W. Aha. Stratified case-based reasoning: Reusing hierarchical problem solving episodes. In *Proceedings of the 14th IJCAI*, pages 384–390, Montreal, Canada, 1995.

[BAFH84] Anthony J. Barbera, James S. Albus, Mary L. Fitzgerald, and Leonard S. Haynes. Rcs: the nbs real-time control system. In *Proceedings of the Robots 8 Conference and Exposition*, volume 2 - Future Considerations, pages 19.1–19.33, Detroit, Michigan, June 1984.

[Bai86] W.M. Bain. *Case-Based Reasoning: A computer Model of Subjective Assessment*. Tesis Doctoral, Yale University, CT, US, 1986.

[BANS02] Edward Brent, Albert Anderson, Lisa Neidert, and Pawel Slusarz. Clearing the metadata hurdle for inexperienced users: A case-based reasoning approach. In *Annual meeting of the International Association for Social Science Information Service and Technology*, Storrs, Connecticut, 2002.

[Bar89] E. Ray Bareiss. *Exemplar Based Knowledge Acquisition: A Unified Approach to Concept Representation, Classification, and Learning*. Academic Press, San Diego, California, 1989.

[BCME+98] Jean-Jacques Borrelly, Eve Coste-Manière, Bernard Espiau, Konstantinos Kapellos, Roger Pissard-Gibollet, Daniel Simon, and Nicolas Turro. The orccad architecture. *International Journal of Robotics Research*, 17(4):338–359, 1998.

[BEF96] Johann Borenstein, Bart Everett, and Liqiang Feng. *Navigating Mobile Robots: Sensors and Techniques*. A.K. Peters Ltd., Wellesley, Massachusetts, 1996.

[BF02] Davide Brugali and Mohamed E. Fayad. Distributed computing in robotics and automation. *IEEE Transactions on Robotics and Automation*, 18(4):409–420, 2002.

[BFG+97] R. Peter Bonasso, R. James Firby, Erann Gat, David Kortenkamp, David P. Miller, and Mark G. Slack. Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2):237–256, 1997.

[BK91] Johann Borenstein and Yoram Koren. The vector field histogram - fast obstacle-avoidance for mobile robots. *IEEE Journal of Robotics and Automation*, 7(3):278–288, 1991.

[BK99] Oliver Brock and Oussama Khatib. High-speed navigation using the global dynamic window approach. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 341–346, Detroit, Michigan, May 1999.

[BK00] Oliver Brock and Oussama Khatib. Real-time replanning in high-dimensional configuration spaces using sets of homotopic paths. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 550–555, San Francisco, California, April 2000.

[BLL92] Jérôme Barraquand, Bruno Langlois, and Jean-Claude Latombe. Numerical potencial field techniques for robot path planning. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(2):224–241, 1992.

[BN01] Tim Bailey and Eduardo M. Nebot. Localisation in large-scale environments. *Robotics and Autonomous Systems*, 37(4):261–281, 2001.

[BNL+03] Michael Bosse, Paul Newman, John Leonard, Martin Soika, Wendelin Feiten, and Seth Teller. An atlas framework for scalable mapping. In *Proceedings of the IEEE International Conference on Robotics and Automation*, Taiwan, September 2003.

[Bod95] Margaret A. Boden. The ai's half-century. *AI Magazine*, 16(4):96–99, 1995.

[Bon91] R. Peter Bonasso. Integrating reaction plans and layered competences through synchronous control. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1225–1233, Sydney, Australia, August 1991.

[Bra91] L. Karl Branting. Exploiting the complementarity of rules and precedents with reciprocity and fairness. In *Proceedings of the DARPA Case-Based Reasoning Workshop*, pages 39–50, Washington, May 1991.

[Bro86] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, 1986.

[Bro89] Rodney A. Brooks. A robot that walks: Emergent behavior from a carefully evolved network. *Neural Computation*, 1(2):253–262, 1989.

[Bro90] Rodney A. Brooks. Elephants don't play chess. *Robotics and Autonomous Systems*, 6:3–15, 1990.

[Bro91] Rodney A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–159, 1991.

[BS04] Par Buschka and Alessandro Saffiotti. Some notes on the use of hybrid maps for mobile robots. In *Proceedings of the 8th International Conference on Intelligent Autonomous Systems*, pages 547–556, Amsterdam, The Netherlands, March 2004.

[Car83] Jaime G. Carbonell. Learning by analogy: Formulating and generalising plans from past experience. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach (Volume I)*, pages 163–190. Tioga, 1983.

[Car86] Jaime G. Carbonell. Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach (Volume II)*, pages 371–392. Morgan Kaufmann, 1986.

[CK93] A. Curran and Kostas J. Kyriakopoulos. Sensor-based self-localization for wheeled mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 8–13, Atlanta, May 1993.

[CK97] Kok S. Chong and Lindsay Kleeman. Large scale sonarray mapping using multiple connected local maps. In *Proceedings of the International Conference on Field and Service Robotics*, pages 538–545, Canberra, Australia, December 1997.

[Cla85] W.J. Clancey. Heurestic classification. *Artificial Intelligence*, 27:289–350, 1985.

[CMS00] Eve Coste-Manière and Reid Simmons. Architecture: The backbone of robotic systems. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 67–72, San Francisco, California, April 2000.

[CN01] Howie Choset and Keiji Nagatani. Topological simultaneous localization and mapping (slam): Towards exact localization without explicit localization. *IEEE Transactions on Robotics and Automation*, 17(2):125–137, 2001.

[Con89a] Jonathan H. Connell. A behavior-based arm controller. *IEEE Transactions on Robotics and Automation*, 5(6):784–791, 1989.

[Con89b] Jonathan H. Connell. A colony architecture for an artificial creature. Technical Report AITR-1151, MIT Artificial Intelligence Laboratory, 1989.

[Con92] Jonathan H. Connell. Sss: A hybrid architecture applied to robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2719–2724, Nice, France, May 1992.

[Cor03] Daniel D. Corkill. Collaborating software: Blackboard and multi-agent systems and the future. In *Proceedings of the IEEE International Lisp Conference*, New York, October 2003.

[Dij59] Edsger W. Dijkstra. A note on two problems in conexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[DJ96] Hans Dulimarta and Anil K. Jain. Client/server control architecture for robot navigation. *Pattern Recognition*, 29(8):1259–1284, 1996.

[DMS02] Tom Duckett, Stephen Marsland, and Jonathan Shapiro. Fast, on-line learning of globally consistent maps. *Autonomous Robots*, 12(3):287–300, 2002.

[dT04] Carmen de Trazegnies. *Sistema de Aprendizaje y Reconocimiento de Objetos 3D a partir de Imágenes Planas.* Tesis Doctoral, Departamento de Tecnología Electrónica, Universidad de Málaga, Málaga, Spain, 2004.

[DY82] Colm Ó'Dúnlaing and Chee K. Yap. A retraction method for planning the motion of a disc. *Journal of Algorithms*, 6(1):104–111, 1982.

[ES94] Chris Elsaessar and Marc Slack. Integrating deliberative planning in a robot architecture. In *Proceedings of the AIAA/NASA Conference on Intelligent Robots in Field, Factory, Service and Space*, pages 782–787, Houston, Texas, March 1994.

[FB74] Raphael A. Finkel and Jon L. Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Informatica*, 4:1–9, 1974.

[FBT97] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics and Automation Magazine*, 4(1):23–33, 1997.

[Fir89] R. James Firby. *Adaptive Execution in Complex Dynamic Worlds.* Tesis Doctoral, Department of Computer Science, Yale University, New Haven, Connecticut, 1989.

[FL95] S. Fox and D.B. Leake. Combining case-based planning and introspective reasoning. In *Proceedings of the 6th Midwest Artificial Intelligence and Cognitive Science Society Conference*, Carbondale, Illinois, 1995.

[FM00] Matthias O. Franz and Hanspeter A. Mallot. Biomimetic robot navigation. *Robotics and Autonomous Systems*, 30:133–153, 2000.

[FS00] Elisabetta Fabrizi and Alessandro Saffiotti. Extracting topology-based maps from gridmaps. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2972–2978, San Francisco, California, April 2000.

[Gat91] Erann Gat. *Reliable Goal-Directed Reactive Control of Autonomous Mobile Robots.* Tesis Doctoral, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 1991.

[Gat98] Erann Gat. On three-layer architectures. In David Kortenkamp, R. Peter Bonasso, and Robin Murphy, editors, *Artificial Intelligence and Mobile Robots*, pages 195–210. AAAI Press, 1998.

[GBFK98] Jens-Steffen Gutmann, Wolfram Burgard, Dieter Fox, and Kurt Konolige. An experimental comparison of localization methods. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 736–743, Victoria, Canada, October 1998.

[GDI+94] Erann Gat, Rajiv Desai, Robert Ivlev, John Loch, and David P. Miller. Behavior control for robotic exploration of planetary surfaces. *IEEE Transactions on Robotics and Automation*, 10(4):490–503, 1994.

[Gen83] Dedre Gentner. Structure mapping: A theoretical framework for analogy. *Cognitive Science*, 7:155–170, 1983.

[GF02] Jens-Steffen Gutmann and Dieter Fox. An experimental comparison of localization methods continued. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 454–459, Lausanne, Switzerland, October 2002.

[GL87] Michael P. Georgeff and Amy L. Lansky. Reactive reasoning and planning. In *Proceedings of the 6th National Conference on Artificial Intelligence*, pages 677–682, Seattle, Washington, July 1987.

[Gon02] Juan J. González. Herramientas de gestión de una arquitectura de control para un agente autónomo móvil. Proyecto Fin de Carrera, Departamento de Tecnología Electrónica, Universidad de Málaga, Málaga, Spain, 2002.

[GOU96] Fabio Gambino, Giuseppe Oriolo, and Giovanni Ulivi. A comparison of three uncertainty calculus techniques for ultrasonic map building. In *Proceedings of the SPIE International Symposium on Aerospace/Defense Sensing and Control*, pages 249–260, Orlando, Florida, April 1996.

[GS99] Jorge Gasós and Alessandro Saffiotti. Integrating fuzzy geometric maps and topological maps for robot navigation. In *Proceedings of the International Symposium on Soft Computing*, pages 754–760, Genova, Italy, June 1999.

[GSV94] D.N. Green, J.Z. Sasiadek, and G.S. Vukovich. Path tracking, obstacle avoidance, and position estimation by an autonomous, wheeled planetary rover. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1300–1305, San Diego, California, 1994.

[HA92] Yong K. Hwang and Narendra Ahuja. Gross motion planning: A survey. *ACM Computing Surveys*, 24(3):219–291, 1992.

[Ham89] Kristian J. Hammond. *Case Based Planning: Viewing Planning as a Memory Task*. AcademicPress, San Diego, California, 1989.

[HB96] Huosheng Hu and Michael Brady. A parallel processing architecture for sensor based control of an intelligent mobile robot. *Robotics and Autonomous Systems*, 17:235–257, 1996.

[Hin92] Thomas R. Hinrichs. *Problem Solving in Open Worlds: A Case Study in Design.* Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1992.

[HP91] Ralph Hartley and Frank Pipitone. Experiments with the subsumption architecture. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1652–1658, Sacramento, California, April 1991.

[HUP+05] Ignacio Herrero, Cristina Urdiales, Jose M. Peula, Isabel Sanchez-Tato, and Francisco Sandoval. A guided learning strategy for vision based navigation of 4-legged robots. *AI-Communications*, (to appear), 2005.

[HV95] Karen Zita Haigh and Maria Manuela Veloso. Route planning by analogy. In *Case-Based Reasoning Research and Development, Proceedings of ICCBR-95*, pages 169–180, October 1995.

[Igl04] José M. Iglesias. Implementación de una arquitectura de control multiplataforma para un agente autónomo móvil. Proyecto Fin de Carrera, Departamento de Tecnología Electrónica, Universidad de Málaga, Málaga, Spain, 2004.

[IWB92] A. Basden I.D. Watson and P.S. Brandon. The client centred approach: Expert system maintenance. *Expert Systems*, 9(4):189–196, 1992.

[JZ00] David Jung and Alexander Zelinsky. Grounded symbolic communication between heterogeneous cooperating robots. *Autonomous Robots*, 8(3):269–292, 2000.

[Kae86] Leslie P. Kaelbling. An architecture for intelligent reactive systems. In *Proceedings of the Workshop on Planning and Reasoning about Action*, pages 395–410, San Mateo, California, 1986.

[Kae88] Leslie P. Kaelbling. Goals as parallel program specifications. In *Proceedings of the National Conference on Artificial Intelligence*, pages 60–65, Saint Paul, Minnesota, August 1988.

[Kal60] Rudolph E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the American Society of Mechanical Engineers, Journal of Basic Engineering*, 82(D):35–45, 1960.

[KB91a] Yoram Koren and Johann Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1394–1404, Sacramento, California, April 1991.

[KB91b] Benjamin Kuipers and Yung-Tai Byun. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. *Robotics and Autonomous Systems*, 8:47–63, 1991.

[KC95] M. Khatib and R. Chatila. An extended potential field approach for mobile robot sensor-based motions. *Intelligent Autonomous Systems*, pages 490–496, 1995.

[Kha86] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5(1):90–98, 1986.

[KM96] Kurt G. Konolige and Karen L. Myers. The saphira architecture for autonomous mobile robots. In David Kortenkamp, R. Peter Bonasso, and Robin Murphy, editors, *AI-based Mobile Robots: Case Studies of Successful Robot Systems*, pages 211–242. MIT Press, 1996.

[Kol83] Janet L. Kolodner. Maintaining organization in a dynamic long-term memory. *Cognitive Science*, 7(4):243–280, 1983.

[Kol93] Janet L. Kolodner. *Case Based Reasoning*. Morgan Kaufmann, San Mateo, California, 1993.

[Kot89] P. Koton. *Using Experience in Learning and Problem Solving*. Tesis Doctoral, Massachusetts Institute of Technology, Laboratory of Computer Science, Boston, Massachusetts, 1989.

[Kru03] M. Kruusma. Global navigation in dynamic environments using case-based reasoning. *Autonomous Robots*, 14:71–91, 2003.

[KS89] Janet L. Kolodner and Robert L. Simpson. The mediator: Analysis of an early case based problem solver. *Cognitive Science*, 13(4):507–549, 1989.

[Kui00] Benjamin Kuipers. The spatial semantic hierarchy. *Artificial Intelligence*, 119:191–233, 2000.

[LA01] M. Likhachev and R.C. Arkini. Spatio-temporal case-based reasoning for behavioral selection. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1627–1634, Seoul, Korea, May 2001.

[Lat91] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Norwell, Massachusetts, 1991.

[Lip87] Richard P. Lippmann. An introduction to computing with neural networks. *IEEE Acoustics, Speech, and Signal Processing Magazine*, 4(2):4–22, 1987.

[LL90] Tod S. Levitt and Daryl T. Lawton. Qualitative navigation for mobile robots. *Artificial Intelligence*, 44(3):305–360, 1990.

[LMP+01] José M. Leiva, Pedro Martínez, Eduardo J. Pérez, Cristina Urdiales, and Francisco Sandoval. 3d reconstruction of a static indoor enviroment by fusion of sonar and video data. In *Proceedings of the 9th Symposium on Intelligent Robotics Systems*, Toulouse, France, July 2001.

[LOC00] Mattias Lindström, Anders Orebäck, and Henrik I. Christensen. Berra: a research architecture for service robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3278 –3283, San Francisco, California, April 2000.

[Mar93] Dario Maravall. *Reconocimiento de Formas y Visión Artificial.* RA-MA, Madrid, 1993.

[Mat92] Maja J. Matarić. Integration of representation into goal-driven behavior-based robots. *IEEE Transactions on Robotics and Automation*, 8(3):304–312, 1992.

[MCM99] David L. Martin, Adam J. Cheyer, and Douglas B. Moran. The open agent architecture: A framework for building distributed software systems. *Applied Artificial Intelligence*, 13(1-2):91–128, 1999.

[MDG+92] David P. Miller, Rajiv S. Desai, Erann Gat, Robert Ivlev, and John Loch. Reactive navigation through rough terrain: Experimental results. In *Proceedings of the National Conference on Artificial Intelligence*, pages 823–828, San Jose, California, July 1992.

[MdLS00] Dario Maravall, Javier de Lope, and Francisco Serradilla. Combination of model-based and reactive methods in autonomous navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2328–2333, San Francisco, California, April 2000.

[MDS93] David J. Musliner, Edmund H. Durfee, and Kang G. Shin. Circa: a cooperative intelligent real-time control architecture. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(6):1561–1574, 1993.

[MJ98] Fernando Matía and Agustín Jiménez. Multisensor fusion: An autonomous mobile robot. *Journal of Intelligent and Robotic Systems: Theory and Applications*, 22(2):129–141, 1998.

[MMSA01] Javier Minguez, Luis Montano, Thierry Siméon, and Rachid Alami. Global nearness diagram navigation (gnd). In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 33–39, Seoul, Korea, May 2001.

[MNPW97] Nicola Muscettola, P. Pandurang Nayak, Barney Pell, and Brian C. Williams. The new millennium remote agent: To boldly go where no ai system has gone before. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, Aichi, Japan, August 1997.

[Mor84] Hans P. Moravec. Locomotion, vision and intelligence. In Michael Brady and Richard Paul, editors, *Robotics Researh 1*, pages 215–224. MIT Press, 1984.

[Mor88] Hans P. Moravec. Sensor fusion in certainty grids for mobile robots. *AI Magazine*, 9(2):61–74, 1988.

[MP43] Warren McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[MP47] Warren McCulloch and Walter Pitts. How we know universals: the perception of auditory and visual forms. *Bulletin of Mathematical Biophysics*, 9:127–147, 1947.

[Nar03] Miguel A. Narváez. Localización de agentes autónomos móviles mediante filtros de kalman. Proyecto Fin de Carrera, Departamento de Tecnología Electrónica, Universidad de Málaga, Málaga, Spain, 2003.

[Nil69] Nils J. Nilsson. A mobile automaton: An application of artificial intelligence techniques. In *Proceedings of the 1st International Joint Conference on Artificial Intelligence*, pages 509–520, Washington DC, 1969.

[Nil82] Nils J. Nilsson. *Principles of Artificial Intelligence*. Springer Verlag, Berlin, 1982.

[OC03] Anders Orebäck and Henrik I. Christensen. Evaluation of architectures for mobile robotics. *Autonomous Robots*, 14:33–49, 2003.

[Oeh92] R. Oehlmann. Learning causal models by self-questioning and experimentation. In *Proceedings of the AAAI-92 Workshop on Communicating Scientific and Technical Knowledge*, pages 73–80, Menlo Park, California, July 1992.

[OVU95] Giuseppe Oriolo, Marilena Vendittelli, and Giovanni Ulivi. On-line map building and navigation for autonomous mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2900–2906, Nagoya, Japan, May 1995.

[PB86] Bruce W. Porter and E. Ray Bareiss. Protos: Anexperiment in knowledge acquisition for heuristic classification tasks. In *Proceedings of the 1st International Meeting on Advances in Learning*, pages 159–174, Les Arcs, France, July 1986.

[PBH90] Bruce W. Porter, E. Ray Bareiss, and Robert Holte. Concept learning and heuristic classification in weak theory domains. *Artificial Intelligence*, 45(1-2):229–263, 1990.

[PNDW98] Daniel Pagac, Eduardo M. Nebot, and Hugh F. Durrant-Whyte. An evidential approach to map-building for autonomous vehicles. *IEEE Transactions on Robotics and Automation*, 14(4):623–629, 1998.

[Pod05] Dionisio Podadera. Administración web de una arquitectura de control para un agente autónomo móvil. Proyecto Fin de Carrera, Departamento de Tecnología Electrónica, Universidad de Málaga, Málaga, Spain, 2005.

[PPB+00] Eduardo J. Pérez, Alberto Poncela, Antonio Bandera, Cristina Urdiales, and Francisco Sandoval. A curvature-based path tracking method on probabilistic grids for autonomous mobile robots. In *Proceedings of the Symposium on Intelligent Robotics Systems*, Reading, United Kingdom, July 2000.

[PPB+02] Alberto Poncela, Eduardo J. Pérez, Antonio Bandera, Cristina Urdiales, and Francisco Sandoval. Efficient integration of metric and topological maps for directed exploration of unknown environments. *Robotics and Autonomous Systems*, 41(1):21–39, 2002.

[PPU+02a] Alberto Poncela, Eduardo J. Pérez, Cristina Urdiales, Antonio Bandera, and Francisco Sandoval. Intelligent navigation in partially unknown indoor environments. In

Da Ruan, Pierre D'hondt, and Etienne E. Kerre, editors, *Computational Intelligent Systems for Applied Research*, pages 495–502. World Scientific, 2002.

[PPU+02b] Alberto Poncela, Eduardo J. Pérez, Cristina Urdiales, Antonio Bandera, and Francisco Sandoval. Intelligent navigation in partially unknown indoor environments. In *Proceedings of the International FLINS Conference on Computational Intelligent Systems for Applied Research*, Gent, Belgium, September 2002.

[PPU+02c] Eduardo J. Pérez, Alberto Poncela, Cristina Urdiales, Antonio Bandera, and Francisco Sandoval. Survey navigation for a mobile robot by using a hierarchical cognitive map. *Cognitive Processing*, 3(3-4):99–106, 2002.

[PPUS05] Alberto Poncela, Eduardo J. Pérez, Cristina Urdiales, and Francisco Sandoval. A hybrid path planning technique for partially unknown indoor environments. *Intelligent Automation and Soft Computing*, 11(3):155–166, 2005.

[PUB+99] Ana Pozo, Cristina Urdiales, Antonio Bandera, Eduardo J. Pérez, and Francisco Sandoval. A path tracking method for autonomous mobile robots based on grid decomposition. In *Proceedings of the Symposium on Intelligent Robotics Systems*, Coimbra, Portugal, July 1999.

[PUdTS04] Alberto Poncela, Cristina Urdiales, Carmen de Trazegnies, and Francisco Sandoval. A new sonar landmark for place recognition. In *Proceedings of the International FLINS Conference on Computational Intelligent Systems for Applied Research*, Blankenberghe, Belgium, September 2004.

[PUS05] Eduardo J. Pérez, Cristina Urdiales, and Francisco Sandoval. Arquitectura híbrida para navegación mediante esquemas de memoria compartida distribuida. In *Actas del Workshop en Agentes Físicos*, Granada, España, Septiembre 2005.

[PUSVS04] Eduardo J. Pérez, Cristina Urdiales, Francisco Sandoval, and Javier Vázquez-Salceda. A cbr strategy for autonomous reactive navigation learning. In *Proceedings of the International Symposium on Robotics and Applications*, pages 179–186, Seville, Spain, June 2004.

[PVN+04] Jose M. Pérez, Ricardo Vázquez, Pedro Núñez, Eduardo J. Pérez, and Francisco Sandoval. A hough-based method for concurrent mapping and localization in indoor environments. In *Proceedings of the IEEE Conference on Robotics, Automation and Mechatronics*, Singapore, December 2004.

[QK93] Sean Quinlan and Oussama Khatib. Elastic bands: Connecting path planning and control. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 802–807, Atlanta, Georgia, May 1993.

[Qui79] J.R. Quinlan. Induction over large databases. Technical Report STAN-CS-739, Stanford University, San Francisco, US, 1979.

[Rei79] J. Reif. Complexity of the mover's problem and generalizations. In *FOCS*, pages 421–427, 1979.

[RH84] William B. Rouse and Ruston M. Hunt. Human problem solving in fault diagnosis tasks. *Advances in Man-Machines Systems Research*, 1:195–222, 1984.

[RK04] Emilio Remolina and Benjamin Kuipers. Towards a general theory of topological maps. *Artificial Intelligence*, 152(1):47–104, 2004.

[Rod01] Juan A. Rodríguez. *Segmentación Espacio-Temporal de Imágenes Mediante Estructuras Jerárquicas de Enlace Adaptativo.* Tesis Doctoral, Departamento de Tecnología Electrónica, Universidad de Málaga, Málaga, Spain, 2001.

[Ros89] Brian H. Ross. Some psychological results on case based reasoning. In *Proceedings of the DARPA Case-Based Reasoning Workshop*, pages 144–147, San Mateo, California, May 1989.

[Ros95] Julio K. Rosenblatt. Damn: A distributed architecture for mobile navigation. In *Proceedings of the Spring Symposium on Lessons Learned for Implemented Software Architectures for Physical Agents*, pages 167–178, Stanford, California, March 1995.

[RP89] Julio K. Rosenblatt and David W. Payton. A fine-grained alternative to the subsumption architecture for mobile robot control. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, volume 2, pages 317–324, Washington DC, June 1989.

[RS89] Christopher K. Riesbeck and Roger C. Schank. *Inside Case Based Reasoning.* Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1989.

[RS93] A. Ram and J.C. Santamaria. A multistrategy case-based and reinforcement learning approach to self-improving reactive control systems for autonomous robotic navigation. In *Proceedings of the 2nd Int. Workshop on Multistrategy Learning*, Harpers Ferry, West Virginia, 1993.

[RW91] M. M. Richter and S. Wess. Similarity, uncertainty and case-based reasoning in patdex. In R. S. Boyer, editor, *Automated Reasoning: Essays in Honor of Woody Bledsoe*, pages 249–266. Kluwer, 1991.

[SA77] Roger C. Schank and Robert P. Abelson. *Scripts, Plans, Goals and Understanding.* Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1977.

[Sav05] Francesco Savelli. *Topological Mapping of Ambiguous Space: Combining Qualitative Biases and Metrical Information.* Tesis Doctoral, Department of Computer and Systems Science, University of Rome La Sapienza, Rome, Italy, 2005.

[SC96] Mary Shaw and Paul Clements. A field guide to boxology: Preliminary classification of architectural styles for software systems. In *Proceeding of the 2nd International Software Architecture Workshop*, pages 50–54, San Francisco, California, October 1996.

[Sch82] Roger C. Schank. *Dynamic Memory: A Theory of Reminding and Learning in Computers and People.* Cambridge University Press, New York, 1982.

[Sch94] Douglas C. Schmidt. Ace: An object-oriented framework for developing distributed applications. In *Proceedings of the 6th USENIX C++ Technical Conference*, Cambridge, Massachusetts, April 1994.

[SCPCW98] Stanley A. Schneider, Vincent W. Chen, Gerardo Pardo-Castellote, and Howard H. Wang. Controlshell: A software architecture for complex electromechanical systems. *International Journal of Robotics Research*, 17:360–382, 1998.

[SD92] Saul Simhon and Gregory Dudek. A global topological map formed by local metric maps. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robotic Systems*, pages 1708–1714, Victoria, Canada, October 1992.

[Sim85] R.L. Simpson. A computer model of case-based reasoning in problem solving: An investigation in the domain of dispute mediation. Technical Report GIT-ICS-85/18, Georgia Institute of Technology, School of Information and Computer Science, Atlanta, US, 1985.

[Sim90] Reid G. Simmons. An architecture for coordinating planning, sensing and action. In *Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*, pages 292–297, San Diego, California, 1990.

[Sim94] Reid G. Simmons. Structured control for autonomous robots. *IEEE Transactions on Robotics and Automation*, 10(1):34–43, 1994.

[SK91] Reid G. Simmons and Eric Krotkov. An integrated walking system for the ambler planetary rover. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2086–2091, Sacramento, California, April 1991.

[SK97] Hagit Shatkay and Leslie P. Kaelbling. Learning topological maps with weak local odometric information. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 920–927, Aichi, Japan, August 1997.

[SLF90] Reid G. Simmons, Long-Ji Lin, and Christopher Fedor. Autonomous task control for mobile robots. In *Proceedings of the IEEE International Symposium on Intelligent Control*, volume 2, pages 663–668, Philadelphia, Pennsylvania, September 1990.

[SMCR+97] Miquel Sànchez-Marrè, Ulises Cortés, Ignasi R. Roda, Manel Poch, and Javier Lafuente. Learning and adaptation in wwtp through case-based reasoning. *Special issue on Machine Learning of Microcomputers in Civil Engineering*, 12(4):251–266, 1997.

[Sol90] Monnett H. Soldo. Reactive and preplanned control in a mobile robot. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1128–1132, Cincinnati, Ohio, May 1990.

[SR92] David B. Skalak and Edwina L. Rissland. Arguments and cases: An inevitable twining. *Artificial Intelligence and Law, An International Journal*, 1(1):3–48, 1992.

[SS88] S. Sharma and D. Sleeman. Refiner: A case-based differential diagnosis aide for knowledge acquisition and knowledge refinement. In *Proceedings of the European Working Session on Learning*, pages 201–210, London, England, October 1988.

[ST79] Kenneth R. Sloan and Steven L. Tanimoto. Progressive refinement of raster images. *IEEE Transactions on Computers*, C-28:871–874, 1979.

[Ste90] W. Richard Stevens. *UNIX Network Programming.* Prentice Hall, London, 1990.

[Syc87] Katia P. Sycara. Finding creative solutions in adversarial impasses. In *Proceedings of the 9th Annual Conference of the Cognitive Science Society*, pages 997–1003, Seattle, Washington, August 1987.

[Syc88] Katia P. Sycara. Using case based reasoning for plan adaptation and repair. In *Proceedings of the Workshop on Case-Based Reasoning*, pages 425–434, Clearwater Beach, Florida, May 1988.

[TBB+98] Sebastian Thrun, Arno Buecken, Wolfram Burgard, Dieter Fox, Thorsten Froehlinghaus, Daniel Hennig, Thomas Hofmann, Michael Krell, and Timo Schmidt. Map learning and high-speed navigation in rhino. In David Kortenkamp, R. Peter Bonasso, and Robin Murphy, editors, *AI-Based Mobile Robots: Case Studies of Successful Robot Systems*, pages 21–52. MIT Press, 1998.

[Thr98] Sebastian Thrun. Learning maps for indoor mobile robot navigation. *Artificial Intelligence*, 99:21–71, 1998.

[Thr02] Sebastian Thrun. Robotic mapping: A survey. In G. Lakemeyer and B.Ñebel, editors, *Exploring Artificial Intelligence in the New Millenium.* Morgan Kaufmann, 2002.

[Tom01] Nicola Tomatis. *Hybrid, Metric-Topological, Mobile Robot Navigation.* Tesis Doctoral, Département de Microtechnique, École Polytechnique Fédérale de Lausanne, Suisse, Switzerland, 2001.

[Tso97] John K. Tsotsos. Intelligent control for perceptually attentive agents: The s* proposal. *Robotics and Autonomous Systems*, 21(1):5–21, 1997.

[UB98] Iwan Ulrich and Johann Borenstein. Vfh+: Reliable obstacle avoidance for fast mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1572–1577, Leuven, Belgium, May 1998.

[UB00] Iwan Ulrich and Johann Borenstein. Vfh*: Local obstacle avoidance with look-ahead verification. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2505–2511, San Francisco, California, April 2000.

[UBP+03] Cristina Urdiales, Antonio Bandera, Eduardo J. Pérez, Alberto Poncela, and Francisco Sandoval. Hierarchical planning in a mobile robot for map learning and navigation. In Changjiu Zhou, Darío Maravall, and Da Ruan, editors, *Autonomous Robotic Systems: Soft Computing and Hard Computing Methodologies and Applications*, pages 165–188. Physica-Verlag, 2003.

[UPS04] Cristina Urdiales, Eduardo J. Pérez, and Francisco Sandoval. A time stamp control strategy for cbr based reactive navigation in dynamic environments with priorities. In *Proceedings of the IEEE/WIC International Conference on Intelligent Agent Technology*, pages 58–64, Beijing, China, September 2004.

[UPSVS03a] Cristina Urdiales, Eduardo J. Pérez, Francisco Sandoval, and Javier Vázquez-Salceda. A cbr based pure reactive layer for autonomous robot navigation. In *Proceedings of the IASTED International Conference on Artificial Intellegence and Soft Computing*, pages 99–104, Banff, Canada, July 2003.

[UPSVS03b] Cristina Urdiales, Eduardo J. Pérez, Francisco Sandoval, and Javier Vázquez-Salceda. A hybrid architecture for autonomous navigation in dynamic environments. In *Proceedings of the IEEE/WIC International Conference on Intelligent Agent Technology*, pages 225–232, Halifax, Canada, October 2003.

[UPVS+06] Cristina Urdiales, Eduardo J. Pérez, Javier Vázquez-Salceda, Miquel Sànchez-Marrè, and Francisco Sandoval. A purely reactive navigation scheme for dynamic environments using case-based reasoning. *Autonomous Robots*, (to appear), 2006.

[Urd99] Cristina Urdiales. *Arquitectura de Control de Movimiento y Exploración para un Agente Autónomo*. Tesis Doctoral, Departamento de Tecnología Electrónica, Universidad de Málaga, Málaga, Spain, 1999.

[Val03] Gabriel Valencia. *Extracción Jerárquica de Regiones de una Secuencia de Vídeo*. Tesis Doctoral, Departamento de Tecnología Electrónica, Universidad de Málaga, Málaga, Spain, 2003.

[VC93] Manuela M. Veloso and Jaime G. Carbonell. Derivational analogy in prodigy: Automating case acquisition, storage and utilization. *Machine Learning*, 10:249–278, 1993.

[vLL98] M. van Lent and J. Laird. Learning by observation in a complex domain. In *Proceedings of the 11th Workshop on Knowledge Acquisition, Modeling and Management*, Banff, Alberta, Canada, 1998.

[VMPU+06] Ricardo Vázquez-Martín, Eduardo J. Pérez, Cristina Urdiales, José C. del Toro, and Francisco Sandoval. Hybrid navigation guidance for intelligent mobiles. In *Proceedings of the IEEE Vehicular Technology Conference*, Melbourne, Australia, May 2006.

[Wat95] Ian Watson. An introduction to case based reasoning. In I. Watson, editor, *Progress in Case-basedreasoning*, pages 3–16. Springer Verlag, 1995.

[WBD98] Peter Willett, John M. Barnard, and Geoffrey M. Downs. Chemical similarity searching. *Journal of Chemical Information and Computer Sciences*, 38(6):983–996, 1998.

[YB96] Brian Yamauchi and Randall Beer. Spatial learning for navigation in dynamic environments. *IEEE Transactions on Systems, Man, and Cybernetics*, 26(3):496–505, 1996.

Apéndice A

Guía de Instalación y Manual de Usuario

Con el trabajo realizado en la presente Tesis se ha generado un CD que incluye todo el software desarrollado para implementar el sistema propuesto, por lo que en este apéndice se va a abordar la instalación y la utilización de dicho software.

Este apéndice se ha organizado como se indica a continuación. En el apartado 1 se especifica el contenido del CD, enumerando los directorios incorporados y la información que contienen. En el apartado 2 se describe detalladamente el proceso de instalación del sistema completo, compuesto por la arquitectura de control *DLA*, la plataforma robótica a utilizar, el sistema *CBR* y los distintos módulos descritos en capítulos anteriores. Por último, en el apartado 3 se presenta el manual de usuario para poder utilizar el sistema propuesto en la implementación de agentes autónomos móviles con capacidad de navegación en entornos dinámicos no estructurados.

1 Contenido del CD

La figura A.1 muestra el árbol de directorios que contiene el CD, cuyo contenido se describe detalladamente a continuación.

Memoria

Este directorio contiene una copia en formato electrónico (*LaTeX*, *PDF* y *PostScript*) de la presente Tesis.

Presentación

Este directorio contiene una copia de la presentación realizada para la defensa de la presente Tesis en formato *PowerPoint*.

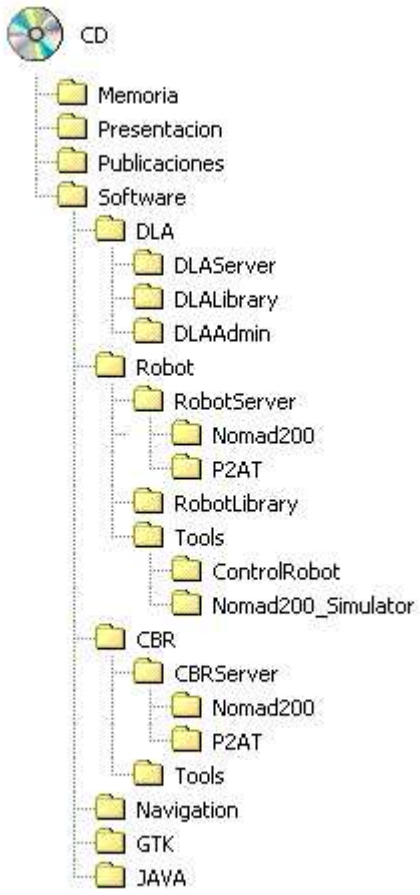

Figura A.1: Contenido del CD generado con la presente Tesis.

Publicaciones

Este directorio contiene una copia en formato electrónico (*PDF*) de todas las publicaciones relacionadas con la presente Tesis. Su contenido se ha organizado en los siguientes directorios:

- **Revistas:** artículos en revista relacionados con la presente Tesis:

1. **RAS02:** [PPB+02]

Título
Efficient Integration of Metric and Topological Maps for Directed Exploration of Unknown Environments
Revista
Robotics and Autonomous Systems
Resumen
Esta publicación aborda el desarrollo de esquemas avanzados de exploración de entornos mediante la utilización de mapas topológicos y algoritmos genéticos.

2. **CP02:** [PPU+02c]

Título
Survey Navigation for a Mobile Robot by Using a Hierarchical Cognitive Map
Revista
Cognitive Processing
Resumen
Esta publicación aborda la implementación de esquemas avanzados de navegación en entornos parcialmente explorados mediante la utilización de mapas topológicos, cubriendo el planificador de rutas desarrollado.

3. **IASC05:** [PPUS05]

Título
A Hybrid Path Planning Technique for Partially Unknown Indoor Environments
Revista
Intelligent Automation and Soft Computing
Resumen
Esta publicación aborda el desarrollo de esquemas avanzados de exploración de entornos mediante la utilización de mapas topológicos y algoritmos genéticos.

4. **AR06:** [UPVS+06]

Título
A Purely Reactive Navigation Scheme for Dynamic Environments Using Case-Based Reasoning
Revista
Autonomous Robots
Resumen
Esta publicación aborda la implementación de un esquema reactivo de navegación mediante el paradigma del razonamiento basado en casos, cubriendo el sistema reactivo de navegación desarrollado.

- **Capítulos:** capítulos de libro relacionados con la presente Tesis:

  1. **CISAR02:** [PPU+02a]

Título
Intelligent Navigation in Partially Unknown Indoor Environments
Libro
Computational Intelligent Systems for Applied Research
Resumen
Esta publicación aborda el desarrollo de esquemas avanzados de exploración de entornos mediante la utilización de mapas topológicos y algoritmos genéticos.

  2. **ARS03:** [UBP+03]

Título
Hierarchical Planning in a Mobile Robot for Map Learning and Navigation
Libro
Autonomous Robotic Systems. Soft Computing and Hard Computing Methodologies and Applications
Resumen
Esta publicación aborda el desarrollo completo de un sistema híbrido que muestre capacidades avanzadas de navegación en entornos dinámicos no estructurados, cubriendo la arquitectura de control, la generación del mapa métrico, el planificador de caminos y el planificador de rutas desarrollados.

- **Congresos:** actas de congreso relacionadas con la presente Tesis:

  1. **SIRS99:** [PUB+99]

Título
A Path Tracking Method for Autonomous Mobile Robots Based on Grid Decomposition
Congreso
Symposium on Intelligent Robotics Systems
Resumen
Esta publicación aborda el problema del seguimiento de caminos mediante una técnica de persecución pura, cubriendo la etapa de seguimiento del planificador de caminos desarrollado.

  2. **SIRS00:** [PPB+00]

Título
A Curvature-Based Path Tracking Method on Probabilistic Grids for Autonomous Mobile Robots
Congreso
Symposium on Intelligent Robotics Systems
Resumen
Esta publicación aborda el problema del seguimiento de caminos mediante la extracción de los puntos de inflexión utilizando una función de curvatura, cubriendo la etapa de seguimiento del planificador de caminos desarrollado.

3. **SIRS01:** [LMP+01]

Título
3D Reconstruction of a Static Indoor Enviroment by Fusion of Sonar and Video Data
Congreso
Symposium on Intelligent Robotics Systems
Resumen
Esta publicación aborda el problema de generación de entornos *3D* utilizando la transformada de Hough sobre un mapa métrico probabilístico, incorporando información de textura captada mediante un sistema de visión.

4. **FLINS02:** [PPU+02b]

Título
Intelligent Navigation in Partially Unknown Indoor Environments
Congreso
International FLINS Conference on Computational Intelligent Systems for Applied Research
Resumen
Esta publicación aborda el desarrollo de esquemas avanzados de exploración de entornos mediante la utilización de mapas topológicos y algoritmos genéticos.

5. **IASTED03:** [UPSVS03a]

Título
A CBR Based Pure Reactive Layer for Autonomous Robot Navigation
Congreso
IASTED International Conference on Artificial Intellegence and Soft Computing
Resumen
Esta publicación aborda la implementación de un esquema reactivo de navegación mediante el paradigma del razonamiento basado en casos, cubriendo el sistema reactivo de navegación desarrollado.

6. **IAT03:** [UPSVS03b]

Título
A Hybrid Architecture for Autonomous Navigation in Dynamic Environments
Congreso
IEEE/WIC International Conference on Intelligent Agent Technology
Resumen
Esta publicación aborda el desarrollo de un sistema híbrido que muestre capacidades avanzadas de navegación en entornos dinámicos no estructurados, cubriendo el sistema reactivo de navegación, la generación del mapa métrico, el planificador de caminos y el planificador de rutas desarrollados.

7. **ISORA04:** [PUSVS04]

Título
A CBR Strategy for Autonomous Reactive Navigation Learning
Congreso
International Symposium on Robotics and Applications
Resumen
Esta publicación aborda la implementación de un esquema reactivo de navegación mediante el paradigma del razonamiento basado en casos, cubriendo el sistema reactivo de navegación desarrollado.

8. **IAT04:** [UPS04]

Título
A Time Stamp Control Strategy for CBR Based Reactive Navigation in Dynamic Environments with Priorities
Congreso
IEEE/WIC International Conference on Intelligent Agent Technology
Resumen
Esta publicación aborda la implementación de un esquema reactivo de navegación multiagente basado en prioridades mediante el paradigma del razonamiento basado en casos, cubriendo el sistema reactivo de navegación desarrollado.

9. **RAM04:** [PVN$^+$04]

Título
A Hough-based Method for Concurrent Mapping and Localization in Indoor Environments
Congreso
IEEE Conference on Robotics, Automation and Mechatronics
Resumen
Esta publicación aborda el problema de la localización en entornos interiores mediante la utilización de la transformada de Hough y el filtro de Kalman.

10. **WAF05:** [PUS05]

Título
Arquitectura Híbrida para Navegación Mediante Esquemas de Memoria Compartida Distribuida
Congreso
Workshop en Agentes Físicos
Resumen
Esta publicación aborda la implementación de una arquitectura de control distribuida mediante un esquema de memoria compartida distribuida para el desarrollo de un sistema híbrido que muestre capacidades avanzadas de navegación en entornos dinámicos no estructurados, cubriendo la arquitectura de control desarrollada.

11. **VTC06:** [VMPU$^+$06]

Título
Hybrid Navigation Guidance for Intelligent Mobiles
Congreso
IEEE Vehicular Technology Conference
Resumen
Esta publicación aborda el problema de la localización en entornos exteriores mediante la utilización de un dispositivo *GPS* y el filtro de Kalman.

Software

Este directorio contiene todo el software desarrollado en la implementación del sistema propuesto en la presente Tesis. Su contenido se ha organizado en los siguientes directorios:

- **DLA:** software correspondiente a la arquitectura de control *DLA*, desarrollado en *C* para *Linux*. Dicho software está compuesto por:
  - **DLAServer:** servidor central de la arquitectura *DLA*.
  - **DLALibrary:** librería de uso de la arquitectura *DLA*.
  - **DLAAdmin:** administrador remoto de la arquitectura *DLA*.
- **Robot:** software necesario para controlar las distintas plataformas robóticas empleadas en el sistema, desarrollado en *C* para *Linux*. Dicho software está compuesto por:
  - **RobotServer:** software que implementa los distintos servidores hardware utilizados en el sistema:
    - **Nomad200:** servidor hardware para la plataforma robótica *Nomad 200*.
    - **P2AT:** servidor hardware para la plataforma robótica *Pioneer P2AT*.
  - **RobotLibrary:** librería de uso del servidor hardware.
  - **Tools:** aplicaciones diversas para su utilización con las plataformas robóticas del sistema:
    - **ControlRobot:** aplicación que permite controlar directamente cualquier plataforma robótica a través del correspondiente servidor hardware.
    - **Nomad200_Simulator:** simulador de la plataforma robótica *Nomad 200* proporcionado por *Nomadics* y sus correspondientes manuales, que se puede descargar gratuitamente de la dirección: `http://nomadic.sourceforge.net`
- **CBR:** software que implementa el sistema *CBR*, desarrollado en *JAVA* y en *C* para *Linux*. Dicho software está compuesto por:
  - **CBRServer:** software que implementa el servidor *CBR* para las distintas plataformas robóticas del sistema:
    - **Nomad200:** servidor *CBR* para la plataforma robótica *Nomad 200*.
    - **P2AT:** servidor *CBR* para la plataforma robótica *Pioneer P2AT*.
  - **Tools:** aplicaciones necesarias para la generación de la base de casos del sistema *CBR*.
- **Navigation:** software de los distintos módulos necesarios para implementar el sistema de navegación en entornos dinámicos no estructurados, desarrollado en *C* para *Linux*.
- **GTK:** software correspondiente al entorno visual *GTK* para *Linux*, que se puede descargar gratuitamente de la dirección: `http://www.gtk.org`
- **JAVA:** software correspondiente al lenguaje de programación *JAVA* para *Linux*, que se puede descargar gratuitamente de la dirección: `http://java.sun.com`

2 Guía de Instalación

En este apartado se va a describir el proceso de instalación sobre *Linux* del sistema desarrollado, a partir del software incluido en el CD. Para realizar dicha instalación se recomienda seguir el orden descrito a continuación y mantener la estructura de directorios original, ya que de esta forma se resuelven adecuadamente las dependencias cruzadas entre el software desarrollado.

Para el proceso de instalación se han desarrollado 10 *scripts*, uno en cada directorio que contiene el software básico a instalar. Todos estos *scripts* de instalación se han denominado `install.sh`, y básicamente se encargan de compilar adecuadamente el software correspondiente a cada directorio.

Para el proceso de desinstalación se han incorporado otros 10 *scripts*, uno en cada directorio que contiene el software básico a desinstalar. Todos estos *scripts* de desinstalación se han denominado `uninstall.sh`, y básicamente se encargan de eliminar los archivos derivados de la compilación del software correspondiente a cada directorio.

Dichos *scripts* se pueden ejecutar directamente desde cualquier consola, aunque para ello es necesario que tengan el correspondiente permiso de ejecución. Suele ser bastante común que los archivos copiados desde un CD en *Linux* carezcan del permiso de ejecución, por lo que el primer paso a realizar debe ser conceder dicho permiso a todos los *scripts* desarrollados.

Puesto que el número de *scripts* implementado es bastante elevado y se encuentran en diversos directorios se ha creado otro *script* que se encarga de conceder el permiso de ejecución a todos los demás. Este nuevo *script* se encuentra en el directorio `Software` del CD y se ha denominado `chmod_all.sh`. Por lo tanto, se puede realizar todo este proceso completo ejecutando simplemente dicho *script*, aunque para ello es necesario concederle en primer lugar el permiso de ejecución. Este proceso de realiza con los siguientes comandos:

```
[%] chmod 777 chmod_all.sh
[%] ./chmod_all.sh
```

Una vez ejecutado este *script* ya se puede iniciar el proceso de instalación utilizando el resto de *scripts* desarrollados, al poseer estos el adecuado permiso de ejecución. La instalación del software desarrollado hay que realizarla según un orden determinado, puesto que algunos programas requieren librerías que hay que generar previamente con otros programas.

Para facilitar el proceso de instalación completa se ha creado el *script* `install_all.sh` ubicado en el directorio `Software` del CD. Este *script* resuelve adecuadamente las dependencias cruzadas entre el software desarrollado, al generar previamente las librerías necesarias y copiarlas en los directorios adecuados. Este proceso de instalación completa del software se realiza con el siguiente comando:

```
[%] ./install_all.sh
```

Durante el proceso de instalación se va mostrando distinta información en la consola, de forma

que se puedan detectar y corregir los posibles errores que surjan durante dicho proceso. Estos errores están relacionados con la falta del software adicional necesario durante la compilación: el entorno visual *GTK* y el lenguaje de programación *JAVA*. Aunque tanto uno como otro se pueden conseguir gratuitamente en Internet se ha incluido en el CD una copia de ambos.

Es necesario indicar que el proceso de instalación completa mediante el *script* `install_all.sh` no instala el servidor hardware *RobotServer* para la plataforma robótica *Pioneer P2AT*, pues dicho proceso necesita ser realizado sobre dicha plataforma. Esta instalación, pues, se debe realizar manualmente, siguiendo las instrucciones que se describen en el apartado 2.2.1.

Para realizar una desinstalación completa del software desarrollado simplemente es necesario borrar los archivos generados y copiados en el proceso de instalación. De nuevo se ha creado un *script* para realizar la desinstalación completa de forma sencilla, denominado `uninstall_all.sh` y ubicado en el directorio `Software` del CD. Este proceso de desinstalación completa se realiza con el siguiente comando:

```
[%] ./uninstall_all.sh
```

De esta forma se puede instalar y desinstalar muy fácilmente todo el software correspondiente al sistema desarrollado. No obstante, si por algún motivo no se desea realizar una instalación o desinstalación completa es necesario llevar a cabo una instalación o desinstalación parcial. Dicho proceso se describe detalladamente en los siguientes apartados, indicando las dependencias cruzadas existentes entre el software y enumerando los comandos necesarios para realizar una correcta instalación o desintalación parcial.

2.1 Arquitectura de control *DLA*

En el directorio `Software/DLA` del CD se encuentra el software correspondiente a la arquitectura de control *DLA*, compuesto por los siguientes elementos:

- **DLAServer:** servidor central de la arquitectura *DLA*.
- **DLALibrary:** librería de uso de la arquitectura *DLA*.
- **DLAAdmin:** administrador remoto de la arquitectura *DLA*.

Se describe a continuación el proceso de instalación y desinstalación de estos elementos.

2.1.1 *DLAServer*

El servidor central *DLAServer* es necesario para poner en funcionamiento la arquitectura de control *DLA*. El proceso de instalación y desinstalación del servidor central se realiza fácilmente mediante los correspondientes *scripts* `install.sh` y `uninstall.sh` ubicados en el directorio

`Software/DLA/DLAServer` del CD. Para ello, no obstante, es necesario en primer lugar conceder el correspondiente permiso de ejecución a ambos *scripts*, operación que se puede realizar con el comando:

```
[%] chmod 777 *.sh
```

Una vez concedido el permiso de ejecución, la instalación del servidor central se puede llevar a cabo fácilmente con el comando:

```
[%] ./install.sh
```

No obstante, si por algún motivo no se desea utilizar este *script* también se puede instalar el servidor central con los comandos:

```
[%] make -f DLAData.make
[%] make -f DLACommand.make
[%] make -f DLAServer.make
```

que son las operaciones que realiza el *script* de instalación. Sin embargo, si por algún motivo tampoco se desea utilizar el comando `make` y se quiere instalar el servidor central directamente se deben emplear los siguientes comandos:

```
[%] gcc -c DLAData.c -O3
[%] gcc -c DLACommand.c -O3
[%] gcc -c DLAServer.c -O3
[%] gcc -o DLAServer DLAData.o DLACommand.o DLAServer.o
```

Tras la instalación se generan los archivos `DLAData.o` y `DLACommand.o`, necesarios para compilar el servidor central, junto con el archivo `DLAServer`, que se corresponde con el propio servidor central.

El proceso de desinstalación se puede realizar fácilmente con el comando:

```
[%] ./uninstall.sh
```

De nuevo, si por algún motivo no se desea utilizar este *script* también se puede desinstalar el servidor central con los comandos:

```
[%] make -f DLAData.make clean
[%] make -f DLACommand.make clean
[%] make -f DLAServer.make clean
```

que son las operaciones que realiza el *script* de desinstalación. Si por algún motivo tampoco se desea utilizar el comando `make` y se quiere desinstalar el servidor central directamente se deben emplear los siguientes comandos:

```
[%] rm -f DLAData.o
[%] rm -f DLACommand.o
[%] rm -f DLAServer.o
[%] rm -f DLAServer
```

Tras la desinstalación se eliminan los archivos generados en el proceso de instalación, restaurando el contenido original del directorio.

2.1.2 *DLALibrary*

La librería de uso de la arquitectura *DLA* es necesaria para que los distintos módulos del sistema puedan utilizar dicha arquitectura. El proceso de instalación y desinstalación de la librería *DLALibrary* se realiza fácilmente mediante los correspondientes *scripts* `install.sh` y `uninstall.sh` ubicados en el directorio `Software/DLA/DLALibrary` del CD. Para ello, no obstante, es necesario en primer lugar conceder el permiso de ejecución a ambos *scripts*, operación que se puede realizar con el comando:

```
[%] chmod 777 *.sh
```

Una vez concedido el permiso de ejecución, la instalación de la librería se puede llevar a cabo fácilmente con el comando:

```
[%] ./install.sh
```

No obstante, si por algún motivo no se desea utilizar este *script* también se puede instalar la librería con el comando:

```
[%] make -f DLALibrary.make
```

que es la operación que realiza el *script* de instalación. Sin embargo, si por algún motivo tampoco se desea utilizar el comando `make` y se quiere instalar la librería directamente se debe emplear el siguiente comando:

```
[%] gcc -c DLALibrary.c -O3
```

Tras la instalación se genera el archivo `DLALibrary.o`, que junto con el archivo `DLALibrary.h` serán necesarios durante el proceso de instalación de cualquier módulo que pretenda utilizar la arquitectura de control *DLA*.

El proceso de desinstalación se puede realizar fácilmente con el comando:

```
[%] ./uninstall.sh
```

De nuevo, si por algún motivo no se desea utilizar este *script* también se puede desinstalar la librería con el comando:

```
[%] make -f DLALibrary.make clean
```

que es la operación que realiza el *script* de desinstalación. Si por algún motivo tampoco se desea utilizar el comando `make` y se quiere desinstalar la librería directamente se debe emplear el siguiente comando:

```
[%] rm -f DLALibrary.o
```

Tras la desinstalación se eliminan los archivos generados en el proceso de instalación, restaurando el contenido original del directorio.

2.1.3 *DLAAdmin*

El administrador remoto *DLAAdmin* permite monitorizar y modificar en tiempo real la operación del servidor central *DLAServer*, facilitando la depuración del sistema desarrollado bajo la arquitectura de control *DLA*. El proceso de instalación y desinstalación del administrador remoto se realiza fácilmente mediante los correspondientes *scripts* `install.sh` y `uninstall.sh` ubicados en el directorio `Software/DLA/DLAAdmin` del CD. Para ello, no obstante, es necesario en primer lugar conceder el permiso de ejecución a ambos *scripts*, operación que se puede realizar con el comando:

```
[%] chmod 777 *.sh
```

Una vez concedido el permiso de ejecución, la instalación del administrador remoto se puede llevar a cabo fácilmente con el comando:

```
[%] ./install.sh
```

No obstante, si por algún motivo no se desea utilizar este *script* también se puede instalar el administrador remoto con los comandos:

```
[%] make -f DLAData.make
[%] make -f DLACommand.make
[%] make -f DLAAdmin.make
```

que son las operaciones que realiza el *script* de instalación. Sin embargo, si por algún motivo tampoco se desea utilizar el comando `make` y se quiere instalar el administrador remoto directamente se deben emplear los siguientes comandos:

```
[%] gcc -c DLAData.c -O3
[%] gcc -c DLACommand.c -O3
[%] gcc -c DLAAdmin.c -O3
[%] gcc -o DLAAdmin DLAData.o DLACommand.o DLAAdmin.o -lrt
```

Tras la instalación se generan los archivos `DLAData.o` y `DLACommand.o`, necesarios para compilar el administrador remoto, junto con el archivo `DLAAdmin`, que se corresponde con el propio administrador remoto.

El proceso de desinstalación se puede realizar fácilmente con el comando:

```
[%] ./uninstall.sh
```

De nuevo, si por algún motivo no se desea utilizar este *script* también se pueden desinstalar el administrador remoto con los comandos:

```
[%] make -f DLAData.make clean
[%] make -f DLACommand.make clean
[%] make -f DLAAdmin.make clean
```

que son las operaciones que realiza el *script* de desinstalación. Si por algún motivo tampoco se desea utilizar el comando `make` y se quiere desinstalar el administrador remoto directamente se deben emplear los siguientes comandos:

```
[%] rm -f DLAData.o
[%] rm -f DLACommand.o
[%] rm -f DLAAdmin.o
[%] rm -f DLAAdmin
```

Tras la desinstalación se eliminan los archivos generados en el proceso de instalación, restaurando el contenido original del directorio.

2.2 Plataforma robótica

En el directorio `Software/Robot` del CD se encuentra el software necesario para controlar las distintas plataformas robóticas empleadas en el sistema, compuesto por los siguientes elementos:

- **RobotServer:** software que implementa los distintos servidores hardware utilizados en el sistema:
  - **Nomad200:** servidor hardware para la plataforma robótica *Nomad 200*.
  - **P2AT:** servidor hardware para la plataforma robótica *Pioneer P2AT*.
- **RobotLibrary:** librería de uso del servidor hardware.
- **Tools:** aplicaciones diversas para su utilización con las plataformas robóticas del sistema:
  - **ControlRobot:** aplicación que permite controlar directamente cualquier plataforma robótica a través del correspondiente servidor hardware.
  - **Nomad200_Simulator:** simulador de la plataforma robótica *Nomad 200* proporcionado por *Nomadics* y sus correspondientes manuales, que se puede descargar gratuitamente de la dirección: `http://nomadic.sourceforge.net`

Se describe a continuación el proceso de instalación y desinstalación de estos elementos.

2.2.1 *RobotServer*

El servidor hardware de la plataforma robótica empleada en el sistema es necesario para poder acceder y controlar dicha plataforma. Se debe disponer de tantos servidores hardware como plataformas robóticas se pretendan utilizar, por lo que se ha implementado un servidor hardware

para la plataforma robótica *Nomad 200* y otro servidor hardware para la plataforma robótica *Pioneer P2AT*.

Nomad200

La instalación del servidor hardware para la plataforma robótica *Nomad 200* necesita la librería proporcionada por el fabricante para acceder a la misma, compuesta por los archivos `NClient.h` y `NClient.o`. El proceso de instalación y desinstalación de dicho servidor hardware se realiza fácilmente mediante los correspondientes *scripts* `install.sh` y `uninstall.sh` ubicados en el directorio `Software/Robot/RobotServer/Nomad200` del CD. Para ello, no obstante, es necesario en primer lugar conceder el permiso de ejecución a ambos *scripts*, operación que se puede realizar con el comando:

```
[%] chmod 777 *.sh
```

Una vez concedido el permiso de ejecución, la instalación del servidor hardware se puede llevar a cabo fácilmente con el comando:

```
[%] ./install.sh
```

No obstante, si por algún motivo no se desea utilizar este *script* también se puede instalar el servidor hardware con los comandos:

```
[%] make -f Nclient.make
[%] make -f RobotServer_Nomad200.make
```

que son las operaciones que realiza el *script* de instalación. Sin embargo, si por algún motivo tampoco se desea utilizar el comando `make` y se quiere instalar el servidor hardware directamente se deben emplear los siguientes comandos:

```
[%] gcc -c Nclient.c -O3
[%] gcc -c RobotServer_Nomad200.c -O3
[%] gcc -o RobotServer_Nomad200 Nclient.o RobotServer_Nomad200.o -lm
```

Tras la instalación se genera el archivo `Nclient.o`, necesario para compilar el servidor hardware para la plataforma robótica *Nomad 200*, junto con el archivo `RobotServer_Nomad200`, que se corresponde con el propio servidor hardware para la plataforma robótica *Nomad 200*.

El proceso de desinstalación se puede realizar fácilmente con el comando:

```
[%] ./uninstall.sh
```

De nuevo, si por algún motivo no se desea utilizar este *script* también se puede desinstalar el correspondiente servidor hardware con los comandos:

```
[%] make -f Nclient.make clean
[%] make -f RobotServer_Nomad200.make clean
```

que son las operaciones que realiza el *script* de desinstalación. Si por algún motivo tampoco se desea utilizar el comando `make` y se quiere desinstalar el servidor hardware directamente se

deben emplear los siguientes comandos:

```
[%] rm -f Nclient.o
[%] rm -f RobotServer_Nomad200.o
[%] rm -f RobotServer_Nomad200
```

Tras la desinstalación se eliminan los archivos generados en el proceso de instalación, restaurando el contenido original del directorio.

P2AT

La instalación del servidor hardware para la plataforma robótica *Pioneer P2AT* debe efectuarse directamente sobre dicha plataforma, pues requiere de una serie de librerías presentes en el sistema operativo *Saphira* [KM96] de la misma. El proceso de instalación y desinstalación de dicho servidor hardware se realiza fácilmente mediante los correspondientes *scripts* `install.sh` y `uninstall.sh` ubicados en el directorio `Software/Robot/RobotServer/P2AT` del CD. Para ello, no obstante, es necesario en primer lugar conceder el permiso de ejecución a ambos *scripts*, operación que se puede realizar con el comando:

```
[%] chmod 777 *.sh
```

Una vez concedido el permiso de ejecución, la instalación del servidor hardware se puede llevar a cabo fácilmente con el comando:

```
[%] ./install.sh
```

No obstante, si por algún motivo no se desea utilizar este *script* también se puede instalar el servidor hardware con el comando:

```
[%] make -f RobotServer_P2AT.make
```

que son las operaciones que realiza el *script* de instalación. No se recomienda realizar la instalación del servidor hardware directamente, pues el comando `make` utilizado resuelve las dependencias cruzadas existentes con el sistema operativo *Saphira* de la misma.

Tras la instalación se genera el archivo `RobotServer_P2AT`, que se corresponde con el propio servidor hardware para la plataforma robótica *Pioneer P2AT*.

El proceso de desinstalación se puede realizar fácilmente con el comando:

```
[%] ./uninstall.sh
```

De nuevo, si por algún motivo no se desea utilizar este *script* también se puede desinstalar el correspondiente servidor hardware con el comando:

```
[%] make -f RobotServer_P2AT.make clean
```

que son las operaciones que realiza el *script* de desinstalación. Si por algún motivo tampoco se desea utilizar el comando `make` y se quiere desinstalar el servidor hardware directamente se

deben emplear los siguientes comandos:

```
[%] rm -f RobotServer_P2AT.o
[%] rm -f RobotServer_P2AT
```

Tras la desinstalación se eliminan los archivos generados en el proceso de instalación, restaurando el contenido original del directorio.

2.2.2 *RobotLibrary*

La librería de uso del servidor hardware *RobotServer* es necesaria para poder acceder a dicho servidor, por lo que debe ser empleada por todos aquellos módulos del sistema que quieran controlar la plataforma robótica del sistema. El proceso de instalación y desinstalación de la librería *RobotLibrary* se realiza fácilmente mediante los correspondientes *scripts* `install.sh` y `uninstall.sh` ubicados en el directorio `Software/Robot/RobotLibrary` del CD. Para ello, no obstante, es necesario en primer lugar conceder el permiso de ejecución a ambos *scripts*, operación que se puede realizar con el comando:

```
[%] chmod 777 *.sh
```

Una vez concedido el permiso de ejecución, la instalación de la librería se puede llevar a cabo fácilmente con el comando:

```
[%] ./install.sh
```

No obstante, si por algún motivo no se desea utilizar este *script* también se puede instalar la librería con el comando:

```
[%] make -f RobotLibrary.make
```

que es la operación que realiza el *script* de instalación. Sin embargo, si por algún motivo tampoco se desea utilizar el comando `make` y se quiere instalar la librería directamente se debe emplear el siguiente comando:

```
[%] gcc -c RobotLibrary.c -O3
```

Tras la instalación se genera el archivo `RobotLibrary.o`, que junto con el archivo `RobotLibrary.h` serán necesarios durante el proceso de instalación de cualquier módulo que pretenda acceder y controlar la plataforma robótica del sistema.

El proceso de desinstalación se puede realizar fácilmente con el comando:

```
[%] ./uninstall.sh
```

De nuevo, si por algún motivo no se desea utilizar este *script* también se puede desinstalar la librería con el comando:

```
[%] make -f RobotLibrary.make clean
```

que es la operación que realiza el *script* de desinstalación. Si por algún motivo tampoco se desea utilizar el comando `make` y se quiere desinstalar la librería directamente se debe emplear el siguiente comando:

```
[%] rm -f RobotLibrary.o
```

Tras la desinstalación se eliminan los archivos generados en el proceso de instalación, restaurando el contenido original del directorio.

2.2.3 *Tools*

La instalación de las aplicaciones que se utilizan con las distintas plataformas robóticas del sistema se describe a continuación.

ControlRobot

Esta sencilla aplicación permite conectarse al servidor hardware de cualquier plataforma robótica para controlarla directamente. Su uso tiene un gran interés, pues permite mover cualquier plataforma robótica de una forma sencilla a través de un menú de comandos. Puesto que dicha aplicación se conecta con el correspondiente servidor hardware necesita para su compilación la librería *RobotLibrary*, por lo que es necesario instalar previamente dicha librería y copiar los archivos `RobotLibrary.h` y `RobotLibrary.o` en el directorio `Software/Robot/Tools/ControlRobot` del CD, donde se encuentra la aplicación de control de la plataforma robótica.

Una vez incorporada la librería de uso del servidor hardware *RobotLibrary* el proceso de instalación y desinstalación de la aplicación de control de la plataforma robótica se realiza fácilmente mediante los correspondientes *scripts* `install.sh` y `uninstall.sh` ubicados en el directorio `Software/Robot/Tools/ControlRobot` del CD. Para ello, no obstante, es necesario en primer lugar conceder el permiso de ejecución a ambos *scripts*, operación que se puede realizar con el comando:

```
[%] chmod 777 *.sh
```

Una vez concedido el permiso de ejecución, la instalación de la aplicación de control de la plataforma robótica se puede llevar a cabo fácilmente con el comando:

```
[%] ./install.sh
```

No obstante, si por algún motivo no se desea utilizar este *script* también se puede instalar la aplicación de control de la plataforma robótica con el comando:

```
[%] make -f ControlRobot.make
```

que es la operación que realiza el *script* de instalación. Sin embargo, si por algún motivo tampoco se desea utilizar el comando `make` y se quiere instalar la aplicación de control de la plataforma robótica directamente se deben emplear los siguientes comandos:

```
[%] gcc -c ControlRobot.c -O3
[%] gcc -o ControlRobot RobotLibrary.o ControlRobot.o
```

Tras la instalación se genera el archivo `ControlRobot`, que se corresponde con la propia aplicación de control de la plataforma robótica.

El proceso de desinstalación se puede realizar fácilmente con el comando:

```
[%] ./uninstall.sh
```

De nuevo, si por algún motivo no se desea utilizar este *script* también se pueden desinstalar la aplicación de control de la plataforma robótica con el comando:

```
[%] make -f ControlRobot.make clean
```

que es la operación que realiza el *script* de desinstalación. Si por algún motivo tampoco se desea utilizar el comando `make` y se quiere desinstalar la aplicación de control de la plataforma robótica directamente se deben emplear los siguientes comandos:

```
[%] rm -f RobotLibrary.h
[%] rm -f RobotLibrary.o
[%] rm -f ControlRobot.o
[%] rm -f ControlRobot
```

Tras la desinstalación se eliminan los archivos generados en el proceso de instalación, restaurando el contenido original del directorio.

Nomad200_Simulator

El simulador de la plataforma robótica *Nomad 200* proporcionado por *Nomadics* no necesita ser instalado, pues se proporciona ya compilado para ser ejecutado bajo cualquier plataforma *Linux*. La única acción a realizar es concederle el permiso de ejecución al archivo `Nserver` que implementa el simulador, operación que se puede realizar con el comando:

```
[%] chmod 777 Nserver
```

2.3 Sistema *CBR*

En el directorio `Software/CBR` del CD se encuentra el software que implementa el sistema *CBR*, compuesto por los siguientes elementos:

- **CBRServer:** software que implementa el servidor *CBR* para las distintas plataformas robóticas del sistema:

  - **Nomad200:** servidor *CBR* para la plataforma robótica *Nomad 200.*
  - **P2AT:** servidor *CBR* para la plataforma robótica *Pioneer P2AT.*

- **Tools:** aplicaciones necesarias para la generación de la base de casos del sistema *CBR.*

Se describe a continuación el proceso de instalación y desinstalación de estos elementos.

2.3.1 *CBRServer*

El servidor *CBRServer* implementa el sistema *CBR* diseñado [1]. Recuérdese que el número de entradas del caso utilizado por el sistema *CBR* depende del número de sensores de la plataforma robótica empleada, tal y como se ha descrito en el apartado 2.2 del capítulo 5. Así pues, se deberá disponer de un sistema *CBR* distinto para cada plataforma robótica en función de la configuración sensorial de la misma. Por este motivo se ha implementado un servidor *CBR* para la plataforma robótica *Nomad 200* que posee 16 sensores sonar y otro servidor *CBR* para la plataforma robótica *Pioneer P2AT* que posee 8 sensores sonar. Estos servidores *CBR* han sido desarrollados en el lenguaje de programación *JAVA*, por lo que es necesario tenerlo instalado previamente a su compilación.

Si por algún motivo se desea utilizar otra plataforma robótica con distinta configuración sensorial es necesario adaptar el correspondiente servidor *CBRServer* a dicha configuración sensorial. Para ello hay que modificar los archivos `CBRControlPanelET.java` y `CBReasoner2.java` y añadir las nuevas entradas y salidas que definirán el caso del sistema *CBR* en función de la configuración sensorial de la nueva plataforma robótica. Para facilitar este proceso se han marcado en estos archivos las zonas de código que es necesario modificar mediante las etiquetas "`--- MODIFY BEGIN ---`" y "`--- MODIFY END ---`".

Nomad200

El proceso de instalación y desinstalación del servidor *CBR* se realiza fácilmente mediante los correspondientes *scripts* `install.sh` y `uninstall.sh` ubicados en el directorio `Software/CBR/CBRServer/Nomad200` del CD. Para ello, no obstante, es necesario en primer lugar conceder el permiso de ejecución a ambos *scripts*, operación que se puede realizar con el comando:

```
[%] chmod 777 *.sh
```

Una vez concedido el permiso de ejecución, la instalación del servidor *CBRServer* se puede llevar a cabo fácilmente con el comando:

```
[%] ./install.sh
```

[1] El servidor *CBRServer* utilizado ha sido desarrollado por el grupo *KEMLg* (*Knowledge Engineering & Machine Learning Group*) de la *Universitat Politècnica de Catalunya* [SMCR+97].

No obstante, si por algún motivo no se desea utilizar este *script* también se puede instalar el servidor *CBRServer* directamente con los comandos:

```
javac CBRControlPanelET.java
javac CBReasoner2.java
```

Tras la instalación se generan todos los archivos correspondientes a las distintas clases empleadas, así como los archivos `CBRControlPanelET.class` y `CBReasoner2.class`, que se corresponden con la aplicación de generación de la base de casos y con el servidor *CBRServer* respectivamente.

El proceso de desinstalación se puede realizar fácilmente con el comando:

```
[%] ./uninstall.sh
```

De nuevo, si por algún motivo no se desea utilizar este *script* también se puede desinstalar el servidor *CBRServer* directamente con los comandos:

```
[%] rm -f *.class
[%] rm -f upc/lsi/kemlg/general/*.class
[%] rm -f upc/lsi/kemlg/cbr/*.class
[%] rm -f upc/lsi/kemlg/cbr/distances/*.class
[%] rm -f upc/lsi/kemlg/cbr/etools/*.class
[%] rm -f uma/dte/etools/cbr/distances/*.class
[%] rm -f nohup.out
```

Tras la desinstalación se eliminan los archivos generados en el proceso de instalación, restaurando el contenido original del directorio.

P2AT

El proceso de instalación y desinstalación del servidor *CBR* se realiza fácilmente mediante los correspondientes *scripts* `install.sh` y `uninstall.sh` ubicados en el directorio `Software/CBR/CBRServer/P2AT` del CD. Para ello, no obstante, es necesario en primer lugar conceder el permiso de ejecución a ambos *scripts*, operación que se puede realizar con el comando:

```
[%] chmod 777 *.sh
```

Una vez concedido el permiso de ejecución, la instalación del servidor *CBRServer* se puede llevar a cabo fácilmente con el comando:

```
[%] ./install.sh
```

No obstante, si por algún motivo no se desea utilizar este *script* también se puede instalar el servidor *CBRServer* directamente con los comandos:

```
javac CBRControlPanelET.java
javac CBReasoner2.java
```

Tras la instalación se generan todos los archivos correspondientes a las distintas clases empleadas, así como los archivos `CBRControlPanelET.class` y `CBReasoner2.class`, que se corresponden

con la aplicación de generación de la base de casos y con el servidor *CBRServer* respectivamente.

El proceso de desinstalación se puede realizar fácilmente con el comando:

```
[%] ./uninstall.sh
```

De nuevo, si por algún motivo no se desea utilizar este *script* también se puede desinstalar el servidor *CBRServer* directamente con los comandos:

```
[%] rm -f *.class
[%] rm -f upc/lsi/kemlg/general/*.class
[%] rm -f upc/lsi/kemlg/cbr/*.class
[%] rm -f upc/lsi/kemlg/cbr/distances/*.class
[%] rm -f upc/lsi/kemlg/cbr/etools/*.class
[%] rm -f uma/dte/etools/cbr/distances/*.class
[%] rm -f nohup.out
```

Tras la desinstalación se eliminan los archivos generados en el proceso de instalación, restaurando el contenido original del directorio.

2.3.2 *Tools*

Para generar la base de casos del sistema *CBR* se han desarrollado diversas aplicaciones que simplifican dicho proceso. Puesto que estas aplicaciones utilizan la arquitectura de control *DLA* necesitan para su compilación la librería *DLALibrary*, por lo que es necesario instalar previamente dicha librería y copiar los archivos `DLALibrary.h` y `DLALibrary.o` en el directorio `Software/CBR/Tools` del CD, donde se encuentran las distintas aplicaciones implementadas. Así mismo, puesto que algunas aplicaciones se conectan con el correspondiente servidor hardware necesitan para su compilación la librería *RobotLibrary*, por lo que también es necesario instalar previamente dicha librería y copiar los archivos `RobotLibrary.h` y `RobotLibrary.o` en el directorio `Software/CBR/Tools` del CD. Por último, algunas aplicaciones se han desarrollado utilizando el entorno visual *GTK*, por lo que es necesario tenerlo instalado previamente a su compilación.

Una vez incorporadas la librería de uso de la arquitectura *DLALibrary* y la librería de uso del servidor hardware *RobotLibrary* el proceso de instalación y desinstalación de las distintas aplicaciones se realiza fácilmente mediante los correspondientes *scripts* `install.sh` y `uninstall.sh` ubicados en el directorio `Software/CBR/Tools` del CD. Para ello, no obstante, es necesario en primer lugar conceder el permiso de ejecución a ambos *scripts*, operación que se puede realizar con el comando:

```
[%] chmod 777 *.sh
```

Una vez concedido el permiso de ejecución, la instalación de las distintas aplicaciones se puede llevar a cabo fácilmente con el comando:

```
[%] ./install.sh
```

No obstante, si por algún motivo no se desea utilizar este *script* también se pueden instalar las distintas aplicaciones con los comandos:

```
[%] make -f TrainHuman.make
[%] make -f TrainPotential.make
[%] make -f Trace2RAW.make
[%] make -f TraceSelector.make
[%] make -f Trace2CBR.make
[%] make -f CBRSequencer.make
[%] make -f CBR2Case.make
[%] make -f Case2Eval.make
[%] make -f Case2Cluster.make
[%] make -f Cluster2CBR.make
[%] make -f TraceLabel.make
```

que son las operaciones que realiza el *script* de instalación. Sin embargo, si por algún motivo tampoco se desea utilizar el comando `make` y se quieren instalar las distintas aplicaciones directamente se deben emplear los siguientes comandos:

```
[%] gcc -c TrainHuman.c `gtk-config --cflags`
[%] gcc -o TrainHuman DLALibrary.o RobotLibrary.o TrainHuman.o -lrt
`gtk-config --libs`
[%] gcc -c TrainPotential.c `gtk-config --cflags`
[%] gcc -o TrainPotential DLALibrary.o RobotLibrary.o TrainPotential.o -lrt
`gtk-config --libs`
[%] gcc -c Trace2RAW.c -O3
[%] gcc -o Trace2RAW DLALibrary.o Trace2RAW.o -lm -lrt
[%] gcc -c TraceSelector.c -O3
[%] gcc -o TraceSelector DLALibrary.o TraceSelector.o -lrt
[%] gcc -c Trace2CBR.c -O3
[%] gcc -o Trace2CBR DLALibrary.o Trace2CBR.o -lm -lrt
[%] gcc -c CBRSequencer.c -O3
[%] gcc -o CBRSequencer DLALibrary.o CBRSequencer.o -lrt
[%] gcc -c CBR2Case.c -O3
[%] gcc -o CBR2Case DLALibrary.o CBR2Case.o -lrt
[%] gcc -c Case2Eval.c -O3
[%] gcc -o Case2Eval DLALibrary.o Case2Eval.o -lm -lrt
[%] gcc -c Case2Cluster.c -O3
[%] gcc -o Case2Cluster DLALibrary.o Case2Cluster.o -lm -lrt
[%] gcc -c Cluster2CBR.c -O3
[%] gcc -o Cluster2CBR DLALibrary.o Cluster2CBR.o -lrt
[%] gcc -c TraceLabel.c -O3
[%] gcc -o TraceLabel DLALibrary.o TraceLabel.o -lrt
```

Tras la instalación se generan los archivos `TrainHuman`, `TrainPotential`, `Trace2RAW`, `TraceSelector`, `Trace2CBR`, `CBRSequencer`, `CBR2Case`, `Case2Eval`, `Case2Cluster`, `Cluster2CBR` y `TraceLabel`, que se corresponden con las distintas aplicaciones implementadas.

El proceso de desinstalación se puede realizar fácilmente con el comando:

```
[%] ./uninstall.sh
```

De nuevo, si por algún motivo no se desea utilizar este *script* también se pueden desinstalar las distintas aplicaciones con los comandos:

```
[%] make -f TrainHuman.make clean
[%] make -f TrainPotential.make clean
[%] make -f Trace2RAW.make clean
[%] make -f TraceSelector.make clean
[%] make -f Trace2CBR.make clean
[%] make -f CBRSequencer.make clean
[%] make -f CBR2Case.make clean
[%] make -f Case2Eval.make clean
[%] make -f Case2Cluster.make clean
[%] make -f Cluster2CBR.make clean
[%] make -f TraceLabel.make clean
```

que son las operaciones que realiza el *script* de desinstalación. Si por algún motivo tampoco se desea utilizar el comando `make` y se quieren desinstalar las distintas aplicaciones directamente se deben emplear los siguientes comandos:

```
[%] rm -f DLALibrary.h
[%] rm -f DLALibrary.o
[%] rm -f RobotLibrary.h
[%] rm -f RobotLibrary.o
[%] rm -f TrainHuman.o
[%] rm -f TrainHuman
[%] rm -f TrainPotential.o
[%] rm -f TrainPotential
[%] rm -f Trace2RAW.o
[%] rm -f Trace2RAW
[%] rm -f TraceSelector.o
[%] rm -f TraceSelector
[%] rm -f Trace2CBR.o
[%] rm -f Trace2CBR
[%] rm -f CBRSequencer.o
[%] rm -f CBRSequencer
[%] rm -f CBR2Case.o
[%] rm -f CBR2Case
[%] rm -f Case2Eval.o
[%] rm -f Case2Eval
[%] rm -f Case2Cluster.o
[%] rm -f Case2Cluster
[%] rm -f Cluster2CBR.o
[%] rm -f Cluster2CBR
[%] rm -f TraceLabel.o
[%] rm -f TraceLabel
```

Tras la desinstalación se eliminan los archivos generados en el proceso de instalación, restaurando el contenido original del directorio.

2.4 Sistema de navegación

En el directorio `Software/Navigation` del CD se encuentra el software de los distintos módulos necesarios para implementar el sistema de navegación en entornos dinámicos no estructurados.

Puesto que los distintos módulos utilizan la arquitectura de control *DLA* necesitan para su compilación la librería *DLALibrary*, por lo que es necesario instalar previamente dicha librería y copiar los archivos `DLALibrary.h` y `DLALibrary.o` en el directorio `Software/Navigation` del CD, donde se encuentran los distintos módulos del sistema. Así mismo, puesto que algunos módulos se conectan con el correspondiente servidor hardware necesitan para su compilación la librería *RobotLibrary*, por lo que también es necesario instalar previamente dicha librería y copiar los archivos `RobotLibrary.h` y `RobotLibrary.o` en el directorio `Software/Navigation` del CD. Por último, algunos módulos se han desarrollado utilizando el entorno visual *GTK*, por lo que es necesario tenerlo instalado previamente a su compilación.

Una vez incorporadas la librería de uso de la arquitectura *DLALibrary* y la librería de uso del servidor hardware *RobotLibrary* el proceso de instalación y desinstalación de los distintos módulos se realiza fácilmente mediante los correspondientes *scripts* `install.sh` y `uninstall.sh` ubicados en el directorio `Software/Navigation` del CD. Para ello, no obstante, es necesario en primer lugar conceder el permiso de ejecución a ambos *scripts*, operación que se puede realizar con el comando:

```
[%] chmod 777 *.sh
```

Una vez concedido el permiso de ejecución, la instalación de los distintos módulos se puede llevar a cabo fácilmente con el comando:

```
[%] ./install.sh
```

No obstante, si por algún motivo no se desea utilizar este *script* también se pueden instalar los distintos módulos con los comandos:

```
[%] make -f Start.make
[%] make -f IFRobot.make
[%] make -f Navigation.make
[%] make -f MapMetric.make
[%] make -f PathPlanning.make
[%] make -f Pyramid.make
[%] make -f MapTopo.make
[%] make -f IFUser.make
```

que son las operaciones que realiza el *script* de instalación. Sin embargo, si por algún motivo tampoco se desea utilizar el comando `make` y se quieren instalar los distintos módulos directamente se deben emplear los siguientes comandos:

```
[%] gcc -c Start.c -O3
[%] gcc -o Start DLALibrary.o Start.o -lrt
[%] gcc -c IFRobot.c -O3
[%] gcc -o IFRobot DLALibrary.o RobotLibrary.o IFRobot.o -lm -lrt
[%] gcc -c Navigation.c -O3
[%] gcc -o Navigation DLALibrary.o Navigation.o -lm -lrt
[%] gcc -c MapMetric.c -O3
[%] gcc -o MapMetric DLALibrary.o MapMetric.o -lm -lrt
[%] gcc -c PathPlanning.c -O3
[%] gcc -o PathPlanning DLALibrary.o PathPlanning.o -lrt
[%] gcc -c Pyramid.c -O3
[%] gcc -c MapTopo.c -O3
[%] gcc -o MapTopo DLALibrary.o Pyramid.o MapTopo.o -lm -lrt
[%] gcc -c IFUser.c `gtk-config --cflags`
[%] gcc -o IFUser DLALibrary.o IFUser.o -lrt `gtk-config --libs`
```

Tras la instalación se genera el archivo `Pyramid.o`, necesario para compilar el módulo *MapTopo*, junto con los archivos `Start`, `IFRobot`, `Navigation`, `MapMetric`, `PathPlanning`, `MapTopo` y `IFUser`, que se corresponden con los distintos módulos del sistema.

El proceso de desinstalación se puede realizar fácilmente con el comando:

```
[%] ./uninstall.sh
```

De nuevo, si por algún motivo no se desea utilizar este *script* también se pueden desinstalar los distintos módulos con los comandos:

```
[%] make -f Start.make clean
[%] make -f IFRobot.make clean
[%] make -f Navigation.make clean
[%] make -f MapMetric.make clean
[%] make -f PathPlanning.make clean
[%] make -f Pyramid.make clean
[%] make -f MapTopo.make clean
[%] make -f IFUser.make clean
```

que son las operaciones que realiza el *script* de desinstalación. Si por algún motivo tampoco se desea utilizar el comando `make` y se quieren desinstalar los distintos módulos directamente se deben emplear los siguientes comandos:

```
[%] rm -f DLALibrary.h
[%] rm -f DLALibrary.o
[%] rm -f RobotLibrary.h
[%] rm -f RobotLibrary.o
[%] rm -f Start.o
[%] rm -f Start
[%] rm -f IFRobot.o
[%] rm -f IFRobot
[%] rm -f Navigation.o
[%] rm -f Navigation
[%] rm -f MapMetric.o
[%] rm -f MapMetric
[%] rm -f PathPlanning.o
[%] rm -f PathPlanning
[%] rm -f Pyramid.o
[%] rm -f MapTopo.o
[%] rm -f MapTopo
[%] rm -f IFUser.o
[%] rm -f IFUser
```

Tras la desinstalación se eliminan los archivos generados en el proceso de instalación, restaurando el contenido original del directorio.

3 Manual de usuario

En este apartado se va a describir la utilización sobre *Linux* del sistema desarrollado, una vez instalado el software incluido en el CD tal y como se ha descrito en el apartado 2.

3.1 Arquitectura de control *DLA*

En el directorio `Software/DLA` del CD se encuentra el software correspondiente a la arquitectura de control *DLA*, compuesto por los siguientes elementos:

- **DLAServer:** servidor central de la arquitectura *DLA*.
- **DLALibrary:** librería de uso de la arquitectura *DLA*.
- **DLAAdmin:** administrador remoto de la arquitectura *DLA*.

Se describe a continuación la utilización de estos elementos.

3.1.1 *DLAServer*

El servidor central *DLAServer* es el elemento responsable de gestionar el intercambio de información entre los módulos del sistema en la arquitectura *DLA*. Su activación es imprescindible para poner en funcionamiento los esquemas distribuido básico y distribuido síncrono de la arquitectura *DLA*, pero no es necesaria si se pretende utilizar el esquema local síncrono de la arquitectura *DLA*, pues en dicho caso los módulos del sistema intercambian información directamente entre sí a través de un esquema de memoria compartida [2].

La activación del servidor central *DLAServer* se realiza mediante el siguiente comando:

```
[%] ./DLAServer
```

Tras la activación del servidor central *DLAServer* se muestra por la consola la siguiente información:

```
**************************************************
DLAServer INITIALITED
**************************************************

DLAServer Host Name :  dla.dte.uma.es
DLAServer Host Address:  192.168.187.67
DLAServer Host Port :  7000
DLAServer Remote Admin:  ENABLED (port:  32899)
--------------------------------------------------
To shutdown this DLAServer:  kill 3559
BUT NEVER : kill -9 3559
```

donde se muestra el nombre (`dla.dte.uma.es`), la dirección IP (`192.168.187.67`) y el puerto asociado con la operación del servidor central (`7000`), si se encuentra habilitada (`ENABLED`) o inhabilitada (`DISABLED`) la operación de administración remota y el puerto asociado con dicha operación (`32899`), así como el número de proceso asociado con el servidor central para poder desactivarlo (`3559`).

La operación de administración remota se realiza mediante el administrador remoto *DLAAdmin*. La posibilidad de inhabilitar la operación de administración remota se ha añadido por posibles motivos de seguridad.

La modificación de estos parámetros se puede realizar fácilmente gracias a la existencia de un archivo de configuración en formato texto denominado `DLAServer.cfg`. El servidor central accede tras su activación a dicho archivo, configurando adecuadamente los distintos parámetros de interés para su operación. Los parámetros que contiene el archivo de configuración `DLAServer.cfg` son los siguientes:

- `SERVER_PORT`: utilizado para especificar el puerto de acceso que se quiere asociar con la

[2] Para una descripción de los distintos esquemas de operación de la arquitectura *DLA* se puede consultar el apartado 3.2 del capítulo 3.

operación del servidor central.

- `REMOTE_ADMIN`: flag utilizado para habilitar (valor `1`) o inhabilitar (valor `0`) la operación de administración remota.

La desactivación del servidor central *DLAServer* se realiza mediante el comando `kill` sobre el número de proceso asociado al mismo, tal y como se muestra en la información presentada por consola al activarlo. Es importante no especificar la opción "`-9`" en el comando `kill`, pues en este caso el servidor central no podría capturar la señal generada por dicho comando y sería eliminado sin poder liberar los recursos empleados en la operación del mismo. Adicionalmente también se puede desactivar la operación del servidor central *DLAServer* mediante la combinación de teclas `<CTRL>+<C>` sobre la consola en la que se encuentra activo, que tiene el mismo efecto que el comando `kill` anteriormente mencionado.

Tras desactivar el servidor central *DLAServer* se liberan todos los recursos reservados y se muestra el siguiente mensaje por consola:

```
**************************************************
DLAServer SHUTDOWNED
**************************************************
```

3.1.2 *DLALibrary*

La librería *DLALibrary* es el elemento responsable de utilizar la arquitectura *DLA* para posibilitar el intercambio de información entre los módulos del sistema. En los esquemas distribuido básico y distribuido síncrono de la arquitectura *DLA* se encarga de conectar los módulos del sistema entre sí a través del servidor central *DLAServer*, mientras que en el esquema local síncrono de la arquitectura *DLA* se encarga de conectar los módulos del sistema entre sí a través de un esquema de memoria compartida [3].

Todos aquellos módulos que quieran utilizar la arquitectura *DLA* mediante la librería *DLALibrary* deben seguir los siguientes pasos durante el proceso de compilación:

1. Incluir el archivo `DLALibrary.h` en el código mediante el siguiente comando:

   ```
   #include "DLALibrary.h"
   ```

2. Añadir los archivos `DLALibrary.h` y `DLALibrary.o` en una ubicación accesible durante el proceso de compilación. Esta ubicación puede ser cualquier directorio base utilizado por el compilador o simplemente el mismo directorio donde se encuentra el código del módulo a compilar.

3. Compilar el módulo (denominado en este ejemplo `Module`) con el siguiente comando básico:

   ```
   [%] gcc -c Module.c
   ```

[3] Para una descripción de los distintos esquemas de operación de la arquitectura *DLA* se puede consultar el apartado 3.2 del capítulo 3.

4. Generar el ejecutable del módulo (denominado en este ejemplo `Module`) con el siguiente comando básico:

   ```
   [%] gcc -o Module DLALibrary.o Module.o -lrt
   ```

   donde es imprescindible utilizar la opción "`-lrt`" del compilador para incorporar la librería en tiempo real de *Linux* durante el proceso de compilación, ya que es necesaria para compilar correctamente la librería *DLALibrary* al utilizar semáforos y memoria compartida.

La librería *DLALibrary* consta de una serie de funciones utilizadas para manejar las conexiones y los buzones necesarios en el intercambio de información entre los módulos del sistema. Se describe a continuación la utilización de estas funciones:

Conexiones

Las conexiones permiten el intercambio de datos entre los módulos del sistema. El tipo de datos a intercambiar no ha sido predefinido, siendo tratados por la arquitectura *DLA* como un flujo de bytes que puede contener cualquier tipo de datos simple o abstracto que requiera un módulo. Las funciones de la librería *DLALibrary* que permiten el uso de las conexiones son:

- **NewConnection:**

Función
`Connection *NewConnection(char *host, int port, char *name, int size)`
Descripción
Crea una conexión con su zona de memoria asociada para intercambiar datos en el sistema
Parámetros
`host`: dirección IP del servidor central (o la palabra `local` para el esquema local) `port`: puerto de acceso del servidor central (ignorado para el esquema local) `name`: nombre de la conexión a ser creada `size`: tamaño en bytes de la conexión a ser creada
Valor devuelto
Correcto: puntero a una variable de tipo `Connection` para posteriores referencias Error: `NULL`

  Las distintas conexiones del sistema están identificadas por su nombre, por lo que módulos que tienen que intercambiar datos entre sí sólo tienen que crear conexiones con el mismo nombre. Estas conexiones serán transparentemente enlazadas entre sí por la arquitectura *DLA*, independientemente de si utilizan un esquema distribuido básico, un esquema distribuido síncrono o un esquema local síncrono. Las conexiones creadas inicializan automáticamente todos sus datos a cero.

  El tamaño de una conexión indica el tamaño de los datos a intercambiar. Es importante notar que las conexiones no pueden ser redimensionadas una vez han sido creadas, por lo que el primer módulo que cree una conexión será el que establezca su tamaño. Todas las llamadas a la función `NewConnection` sobre una conexión ya existente simplemente ignoran el parámetro `size`.

- **InputConnection:**

Función
`int InputConnection(Connection *conn, void *data, int pos, int size)`
Descripción
Realiza la lectura de los datos almacenados en una conexión
Parámetros
`conn`: puntero a la conexión que se quiere leer `data`: puntero a la zona de memoria donde se quieren almacenar los datos leídos `pos`: posición inicial de los datos que se quieren leer `size`: tamaño en bytes de los datos que se quieren leer
Valor devuelto
Correcto: número de bytes leídos Error: `-1`

Mediante los parámetros `pos` y `size` se permite la lectura selectiva de una zona de datos determinada de la conexión sin necesidad de realizar la lectura completa de todos los datos almacenados en la misma. El parámetro `pos` indica la primera posición que desea ser leída, correspondiendo el valor cero a la primera posición disponible. El parámetro `size` indica la cantidad de datos en bytes que se desea leer, a partir de la posición `pos` especificada. Si el parámetro `size` contiene el valor cero, se interpreta que se quieren leer todos los datos disponibles desde la posición `pos` hasta el final de la conexión. Así pues, indicando cero en ambos parámetros ser procede a realizar la lectura completa de todos los datos almacenados en la conexión.

- **OutputConnection:**

Función
`int OutputConnection(Connection *conn, void *data, int pos, int size)`
Descripción
Realiza la escritura de los datos almacenados en una conexión
Parámetros
`conn`: puntero a la conexión que se quiere escribir `data`: puntero a la zona de memoria que contiene los datos que se quieren escribir `pos`: posición inicial de los datos que se quieren leer `size`: tamaño en bytes de los datos que se quieren leer
Valor devuelto
Correcto: número de bytes escritos Error: `-1`

Mediante los parámetros `pos` y `size` se permite la escritura selectiva de una zona de datos determinada de la conexión sin necesidad de realizar la escritura completa de todos los datos almacenados en la misma. El parámetro `pos` indica la primera posición que desea ser escrita, correspondiendo el valor cero a la primera posición disponible. El parámetro `size` indica la cantidad de datos en bytes que se desea escribir, a partir de la posición `pos` especificada. Si el parámetro `size` contiene el valor cero, se interpreta que se quieren escribir todos los datos disponibles desde la posición `pos` hasta el final de la conexión. Así pues, indicando cero en ambos parámetros ser procede a realizar la escritura completa de todos los datos almacenados en la conexión.

- **SizeConnection:**

Función
`int SizeConnection(Connection *conn)`
Descripción
Determina el tamaño en bytes de una conexión
Parámetros
`conn`: puntero a la conexión cuyo tamaño se quiere determinar
Valor devuelto
Correcto: tamaño en bytes de la conexión Error: `-1`

Esta función es útil para determinar el tamaño real que tiene una conexión recién creada en el sistema, pues como se ha comentado en la función `NewConnection` las conexiones no pueden ser redimensionadas una vez han sido creadas, por lo que el primer módulo que cree una conexión será el que establezca su tamaño.

- **CloseConnection:**

Función
`int CloseConnection(Connection *conn)`
Descripción
Destruye una conexión y libera los recursos reservados por la misma
Parámetros
`conn`: puntero a la conexión que se quiere destruir
Valor devuelto
Correcto: `0` Error: `-1`

Esta función permite liberar los recursos utilizados por una conexión que no se va a utilizar más. Si se utilizan los esquemas distribuido básico o distribuido síncrono, la desconexión de los módulos del servidor central es automáticamente detectada y se liberan los recursos utilizados aunque no se utilice esta función. Si se utiliza el esquema local síncrono no se puede detectar automáticamente la desconexión de los módulos, por lo que es necesario utilizar esta función para liberar los recursos utilizados.

Buzones

Los buzones permiten ampliar la funcionalidad de la arquitectura de control *DLA* incorporando mecanismos de sincronización entre los distintos módulos del sistema. El tipo de datos a intercambiar ha sido predefinido al tipo entero, pues su utilidad principal no está asociada al dato intercambiado sino al instante de tiempo en el que se intercambia. Las funciones de la librería *DLALibrary* que permiten el uso de los buzones son:

- **NewMailbox:**

Función
`Mailbox *NewMailbox(char *host, int port, char *name)`
Descripción
Crea un buzón para realizar operaciones de sincronización en el sistema
Parámetros
`host`: dirección IP del servidor central (o la palabra `local` para el esquema local) `port`: puerto de acceso del servidor central (ignorado para el esquema local) `name`: nombre del buzón a ser creado
Valor devuelto
Correcto: puntero a una variable de tipo `Mailbox` para posteriores referencias Error: `NULL`

Los distintos buzones del sistema están identificados por su nombre, por lo que módulos que tienen que utilizar un mismo buzón para sincronizarse entre sí sólo tienen que crear buzones con el mismo nombre. Estos buzones serán transparentemente enlazados entre sí por la arquitectura *DLA*, independientemente de si utilizan un esquema distribuido básico, un esquema distribuido síncrono o un esquema local síncrono. Los buzones pueden almacenar un valor de tipo entero, y al crearse inicializan automáticamente su valor a cero.

- **ReadMailbox:**

Función
`int ReadMailbox(Mailbox *mail, int *data)`
Descripción
Realiza la lectura del valor almacenado en un buzón
Parámetros
`mail`: puntero al buzón que se quiere leer `data`: puntero a la variable de tipo entero donde se quiere almacenar el valor leído
Valor devuelto
Correcto: `0` Error: `-1`

Esta función permite leer el valor almacenado en un buzón. La función es no bloqueante, por lo que tras su ejecución el valor almacenado en el buzón es instantáneamente leído y almacenado en la variable indicada.

- **WaitMailbox:**

Función
`int WaitMailbox(Mailbox *mail, int *data)`
Descripción
Espera la llegada de nuevos valores a un buzón desde la última vez que se accedió
Parámetros
`mail`: puntero al buzón que se quiere esperar `data`: puntero a la variable de tipo entero donde se quiere almacenar el valor leído
Valor devuelto
Correcto: `0` Error: `-1`

Esta función espera a que se escriban nuevos valores en un buzón desde la última vez que se accedió al mismo. Si no se ha escrito ningún nuevo valor en el buzón desde la última vez que

se accedió la función es bloqueante, por lo que el módulo que realiza la llamada a la función pasa a un estado inactivo hasta que se escriba un nuevo valor en el buzón, momento en el cual será reactivado y se producirá el retorno de la función. Si algún nuevo valor ha sido escrito en el buzón desde la última vez que se accedió la función es no bloqueante, por lo que tras su ejecución el valor almacenado en el buzón es instantáneamente leído y almacenado en la variable indicada, siendo totalmente equivalente a la función `ReadMailbox`.

- **TestMailbox:**

Función
`int TestMailbox(Mailbox *mail)`
Descripción
Comprueba la llegada de nuevos valores a un buzón desde la última vez que se accedió
Parámetros
`mail`: puntero al buzón que se quiere comprobar
Valor devuelto
Correcto: `0` si no se han escrito nuevos valores, ó `1` si se han escrito nuevos valores Error: `-1`

Esta función es útil para determinar si una posterior llamada a la función `WaitMailbox` será o no bloqueante.

- **WriteMailbox:**

Función
`int WriteMailbox(Mailbox *mail, int *data)`
Descripción
Realiza la escritura de un nuevo valor en un buzón
Parámetros
`mail`: puntero al buzón que se quiere escribir `data`: puntero a la variable de tipo entero que contiene el valor que se quiere escribir
Valor devuelto
Correcto: `0` Error: `-1`

Esta función permite escribir un nuevo valor en un buzón. Tras escribir el valor correspondiente en el buzón, todos aquellos módulos bloqueados tras la llamada a la función `WaitMailbox` son automáticamente reactivados, produciéndose el retorno de dicha función.

- **CloseMailbox:**

Función
`int CloseMailbox(Mailbox *mail)`
Descripción
Destruye un buzón y libera los recursos reservados por el mismo
Parámetros
`mail`: puntero al buzón que se quiere destruir
Valor devuelto
Correcto: `0` Error: `-1`

Esta función permite liberar los recursos utilizados por un buzón que no se va a utilizar más. Si se utilizan los esquemas distribuido básico o distribuido síncrono, la desconexión

de los módulos del servidor central es automáticamente detectada y se liberan los recursos utilizados aunque no se utilice esta función. Si se utiliza el esquema local síncrono no se puede detectar automáticamente la desconexión de los módulos, por lo que es necesario utilizar esta función para liberar los recursos utilizados.

3.1.3 *DLAAdmin*

El administrador remoto *DLAAdmin* es el elemento responsable de monitorizar y modificar en tiempo real la operción del servidor central *DLAServer* de la arquitectura *DLA*. Constituye una herramienta adicional que facilita enormemente el proceso de desarrollo y depuración de las aplicaciones que utilicen la arquitectura *DLA*, pues permite acceder y modificar todos los datos intercambiados entre los módulos del sistema. Por motivos de seguridad sólo es posible el uso de esta herramienta cuando haya sido habilitada la operación de administración remota en el servidor central *DLAServer*.

La ejecución del administrador remoto *DLAAdmin* se realiza mediante el siguiente comando:

```
[%] ./DLAAdmin [DLAServer_Host] [DLAServer_Port]
```

donde los parámetros `DLAServer_Host` y `DLAServer_Port` de la línea de comandos indican la dirección IP y el puerto de acceso respectivamente donde se encuentra el servidor central *DLAServer* que se quiere administrar. Si estos parámetros no se especifican en la línea de comandos se solicitará su entrada por teclado en la propia consola. Para emplear el esquema local de la arquitectura *DLA* tan sólo hay que indicar la palabra `local` en el parámetro `DLAServer_Host`, ignorándose en dicho caso el parámetro `DLAServer_Port`.

Es posible que tras indicar los parámetros `DLAServer_Host` y `DLAServer_Port` el administrador remoto *DLAAdmin* no pueda conectarse con el correspondiente servidor central *DLAServer* por encontrarse inhabilitada la operación de administración remota en dicho servidor, en cuyo caso aparece un mensaje por la consola indicando dicha circunstancia ("`DLAServer Remote Administration not activated in DLAServer`").

Si por el contrario la operación de administración remota está habilitada en el correspondiente servidor central *DLAServer* aparece un menú que permite realizar las siguientes operaciones [4]:

- **STATUS server:** presenta información del servidor central. Dicha información está formada por los siguientes campos:
  - **Estado del servidor:** indica si el servidor está activo aceptando nuevas conexiones y/o buzones (`RUNNING`) o si está detenido sin aceptar nuevas conexiones y/o buzones (`STOPPED`).

[4] El administrador remoto *DLAAdmin* ha sido mejorado a lo largo de distintos Proyectos Fin de Carrera realizados bajo la dirección del autor de la presente Tesis [Gon02, Pod05].

- **Conexiones:** indica la información relativa a todas las conexiones existentes en el sistema. Para los esquemas distribuido básico y distribuido síncrono esta información consta de:
  * nombre de la conexión
  * tamaño en bytes de la conexión (`size`)
  * puerto de acceso del servidor específico asociado a la conexión (`port`)
  * número de proceso del servidor específico asociado a la conexión (`pid`)
  * número de módulos que están utilizando la conexión (`clients`)
  * identificativo `[MANUAL]` si la conexión es de tipo manual

  Para el esquema local síncrono esta información consta de:
  * nombre de la conexión
  * tamaño en bytes de la conexión (`size`)
  * identificador del semáforo que controla el acceso a la zona de memoria compartida de la conexión (`sem_id`)
  * número de módulos que están utilizando la conexión (`clients`)
  * identificativo `[MANUAL]` si la conexión es de tipo manual
- **Buzones:** indica la información relativa a todos los buzones existentes en el sistema. Para los esquemas distribuido básico y distribuido síncrono esta información consta de:
  * nombre del buzón
  * puerto de acceso del servidor específico de gestión de los buzones (`port`)
  * número de proceso del servidor específico de gestión de los buzones (`pid`)
  * número de módulos que están utilizando el buzón (`clients`)
  * identificativo `[MANUAL]` si el buzón es de tipo manual

  Para el esquema local síncrono esta información consta de:
  * nombre del buzón
  * identificador del semáforo que controla el acceso al buzón (`sem_id`)
  * número de módulos que están utilizando el buzón (`clients`)
  * identificativo `[MANUAL]` si el buzón es de tipo manual

La opción de detener la actividad del servidor central se ha añadido por motivos de depuración, y permite detener la creación de nuevas conexiones y/o buzones, aunque las conexiones y/o buzones ya existentes en el sistema se seguirán atendiendo normalmente. Esta opción de detener la actividad del servidor central no está disponible para el esquema local síncrono de la arquitectura *DLA*, pues dicho esquema carece de servidor central y los distintos módulos se conectan directamente entre sí mediante memoria compartida.

Las conexiones y buzones de tipo manual son aquellos que no han sido creados mediante las peticiones recibidas de los distintos módulos, sino mediante el administrador remoto *DLAAdmin*. Este tipo de conexiones y buzones de tipo manual permanecen en el sistema aunque no exista ningún módulo que los utilicen, y sólo pueden ser eliminados mediante

el propio administrador remoto *DLAAdmin*. Este modo de operación ha sido añadida para mantener la persistencia de algunos datos que se puedan considerar importantes en el sistema aun cuando no haya módulos que los utilicen.

- **STOP server:** detiene la actividad del servidor central, impidiendo que se puedan crear nuevas conexiones y/o buzones, aunque las conexiones y/o buzones ya existentes en el sistema se seguirán atendiendo normalmente. Esta opción no está disponible para el esquema local síncrono de la arquitectura *DLA*, pues dicho esquema carece de servidor central y los distintos módulos se conectan directamente entre sí mediante memoria compartida.

- **CONTINUE server:** reanuda la actividad del servidor central, posibilitando que se puedan crear nuevas conexiones y/o buzones. Esta opción no está disponible para el esquema local síncrono de la arquitectura *DLA*, pues dicho esquema carece de servidor central y los distintos módulos se conectan directamente entre sí mediante memoria compartida.

- **RESTART server:** reinicia la operación del servidor central, eliminando todas las conexiones y buzones presentes en el sistema y liberando todos los recursos utilizados por los mismos.

- **SHUTDOWN server:** desactiva el servidor central, eliminando previamente todas las conexiones y buzones presentes en el sistema y liberando todos los recursos utilizados por los mismos. Posteriormente finaliza la ejecución del administrador remoto *DLAAdmin*.

- **DISCONNECT server:** se desconecta del servidor central y finaliza la ejecución del administrador remoto *DLAAdmin*.

- **NEW connection:** procede a la creación de una conexión de tipo manual en el sistema, que sólo podrá ser eliminada mediante el propio administrador remoto *DLAAdmin*. Para crear la conexión es necesario especificar el nombre de la misma y su tamaño en bytes. Si la conexión ya existe se indica mediante un mensaje de error ("`ERROR: Connection already exists in DLAServer`") y no se realiza ninguna operación. Si la conexión no existe sistema se crea y se muestra la información básica de la misma. Para los esquemas distribuido básico y distribuido síncrono esta información consta de:

  - nombre de la conexión
  - tamaño en bytes de la conexión (`size`)
  - puerto de acceso del servidor específico asociado a la conexión (`port`)

  Para el esquema local síncrono esta información consta de:

  - nombre de la conexión
  - tamaño en bytes de la conexión (`size`)
  - identificador del semáforo que controla el acceso a la zona de memoria compartida de la conexión (`sem_id`)

- **DELETE connection:** elimina una conexión del sistema. Para eliminar la conexión es necesario especificar el nombre de la misma. Si la conexión no existe se indica mediante un mensaje de error ("`ERROR: Connection not found in DLAServer`") y no se realiza ninguna operación. Si la conexión existe se elimina y se liberan todos los recursos utilizados por la misma.

- **INPUT connection:** muestra los datos almacenados en una conexión. Para leer la conexión es necesario especificar su nombre. Si la conexión no existe se indica mediante un mensaje de error ("`ERROR: Connection not found in DLAServer`") y no se realiza ninguna operación. Si la conexión existe muestra su tamaño en bytes, tras lo cual es necesario indicar la posición de origen y el tamaño en bytes de los datos que se quieren leer. La posición inicial es la cero, y si se indica un tamaño en bytes igual a cero se procederá a leer todos los datos disponibles desde la posición de origen hasta el final de la conexión. Así pues, indicando cero en ambos parámetros ser procede a realizar la lectura completa de todos los datos almacenados en la conexión.

- **OUTPUT connection:** modifica los datos almacenados en una conexión. Para escribir la conexión es necesario especificar su nombre. Si la conexión no existe se indica mediante un mensaje de error ("`ERROR: Connection not found in DLAServer`") y no se realiza ninguna operación. Si la conexión existe muestra su tamaño en bytes, tras lo cual es necesario indicar la posición de origen y el tamaño en bytes de los datos que se quieren escribir. La posición inicial es la cero, y si se indica un tamaño en bytes igual a cero se procederá a escribir todos los datos disponibles desde la posición de origen hasta el final de la conexión. Así pues, indicando cero en ambos parámetros ser procede a realizar la escritura completa de todos los datos almacenados en la conexión. Tras especificar estos parámetros se deben introducir los bytes que se quieren almacenar en la conexión.

- **NEW mailbox:** procede a la creación de un buzón de tipo manual en el sistema, que sólo podrá ser eliminado mediante el propio administrador remoto *DLAAdmin*. Para crear el buzón es necesario especificar el nombre del mismo. Si el buzón ya existe se indica mediante un mensaje de error ("`ERROR: Mailbox already exists in DLAServer`") y no se realiza ninguna operación. Si el buzón no existe se crea y se muestra la información básica del mismo. Para los esquemas distribuido básico y distribuido síncrono esta información consta de:

  - nombre del buzón
  - puerto de acceso del servidor específico de gestión de los buzones (`port`)

  Para el esquema local síncrono esta información consta de:

  - nombre del buzón
  - identificador del semáforo que controla el acceso al buzón (`sem_id`)

- **DELETE mailbox:** elimina un buzón del sistema. Para eliminar el buzón es necesario especificar el nombre del mismo. Si el buzón no existe se indica mediante un mensaje de

error ("`ERROR: Mailbox not found in DLAServer`") y no se realiza ninguna operación. Si el buzón existe se elimina y se liberan todos los recursos utilizados por el mismo.

- **READ mailbox:** muestra el valor almacenado en un buzón. Para leer un buzón es necesario especificar su nombre. Si el buzón no existe se indica mediante un mensaje de error ("`ERROR: Mailbox not found in DLAServer`") y no se realiza ninguna operación. Si el buzón existe muestra la información relacionada con el mismo. Esta información consta del valor almacenado en el buzón, del número de secuencia del buzón [5] y del número de módulos que están bloqueados a la espera de que se escriban nuevos valores en el buzón.

- **WRITE mailbox:** modifica el valor almacenado en un buzón. Para escribir un buzón es necesario especificar su nombre. Si el buzón no existe se indica mediante un mensaje de error ("`ERROR: Mailbox not found in DLAServer`") y no se realiza ninguna operación. Si el buzón existe es necesario especificar el nuevo valor que se desea escribir en el buzón.

3.2 Plataforma robótica

En el directorio `Software/Robot` del CD se encuentra el software necesario para controlar las distintas plataformas robóticas empleadas en el sistema, compuesto por los siguientes elementos:

- **RobotServer:** software que implementa los distintos servidores hardware utilizados en el sistema:
  - **Nomad200:** servidor hardware para la plataforma robótica *Nomad 200*.
  - **P2AT:** servidor hardware para la plataforma robótica *Pioneer P2AT*.
- **RobotLibrary:** librería de uso del servidor hardware.
- **Tools:** aplicaciones diversas para su utilización con las plataformas robóticas del sistema:
  - **ControlRobot:** aplicación que permite controlar directamente cualquier plataforma robótica a través del correspondiente servidor hardware.
  - **Nomad200_Simulator:** simulador de la plataforma robótica *Nomad 200* proporcionado por *Nomadics* y sus correspondientes manuales, que se puede descargar gratuitamente de la dirección: `http://nomadic.sourceforge.net`

Se describe a continuación la utilización de estos elementos.

[5] El número de secuencia es un parámetro interno que se incrementa cada vez que se escribe un valor en el buzón, y se utiliza para diferenciar sucesivas escrituras del mismo valor en el mismo.

3.2.1 *RobotServer*

El servidor hardware *RobotServer* es el elemento responsable de gestionar el acceso a la correspondiente plataforma robótica. Su activación es imprescindible para poder controlar dicha plataforma, y debe ser ejecutado sobre la propia plataforma robótica que se quiere controlar. Puesto que se dispone de las plataformas robóticas *Nomad 200* y *Pioneer P2AT* en el sistema se ha desarrollado un servidor hardware para cada una de ellas.

Nomad200

El servidor hardware para la plataforma robótica *Nomad 200* se denomina `RobotServer_Nomad200`. La activación del servidor hardware se realiza mediante el siguiente comando:

```
[%] ./RobotServer_Nomad200
```

Tras la activación del servidor hardware se muestra por la consola la siguiente información:

```
**************************************************
RobotServer INITIALITED
**************************************************

RobotServer Host Name :  kenobi.dte.uma.es
RobotServer Host Address:  192.168.187.65
RobotServer Host Port :  7001
--------------------------------------------------
To shutdown this RobotServer:  kill 5367
BUT NEVER : kill -9 5367
```

donde se muestra el nombre (`kenobi.dte.uma.es`), la dirección IP (`192.168.187.65`) y el puerto asociado con la operación del servidor hardware (`7001`), así como el número de proceso asociado con el servidor hardware para poder desactivarlo (`5367`).

La modificación de estos parámetros se puede realizar fácilmente gracias a la existencia de un archivo de configuración en formato texto denominado `RobotServer_Nomad200.cfg`. El servidor hardware accede tras su activación a dicho archivo, configurando adecuadamente los distintos parámetros de interés para su operación. Los parámetros que contiene el archivo de configuración `RobotServer_Nomad200.cfg` son los siguientes:

- `SERVER_PORT`: utilizado para especificar el puerto de acceso que se quiere asociar con la operación del correspondiente servidor hardware.
- `REAL_ROBOT`: flag utilizado para indicar si se va a utilizar una plataforma robótica real (valor `1`) o un simulador de la plataforma robótica (valor `0`).

La desactivación del servidor hardware se realiza mediante el comando `kill` sobre el número de proceso asociado al mismo, tal y como se muestra en la información presentada por consola

al activarlo. Es importante no especificar la opción "-9" en el comando kill, pues en este caso el servidor hardware no podría capturar la señal generada por dicho comando y sería eliminado sin poder liberar los recursos empleados en la operación del mismo. Adicionalmente también se puede desactivar la operación del servidor hardware mediante la combinación de teclas <CTRL>+<C> sobre la consola en la que se encuentra activo, que tiene el mismo efecto que el comando kill anteriormente mencionado.

Tras desactivar el servidor hardware se liberan todos los recursos reservados y se muestra el siguiente mensaje por consola:

```
**************************************************
RobotServer SHUTDOWNED
**************************************************
```

Pioneer P2AT

El servidor hardware para la plataforma robótica *Pioneer P2AT* se denomina RobotServer_P2AT. La activación del servidor hardware se realiza mediante el siguiente comando:

```
[%] ./RobotServer_P2AT
```

Tras la activación del servidor hardware se muestra por la consola la siguiente información:

```
**************************************************
RobotServer INITIALITED
**************************************************

RobotServer Host Name :  p2at.dte.uma.es
RobotServer Host Address:  192.168.187.110
RobotServer Host Port :  7001
--------------------------------------------------
To shutdown this RobotServer:  kill 7532
BUT NEVER : kill -9 7532
```

donde se muestra el nombre (p2at.dte.uma.es), la dirección IP (192.168.187.110) y el puerto asociado con la operación del servidor hardware (7001), así como el número de proceso asociado con el servidor hardware para poder desactivarlo (7532).

La modificación de estos parámetros se puede realizar fácilmente gracias a la existencia de un archivo de configuración en formato texto denominado RobotServer_P2AT.cfg. El servidor hardware accede tras su activación a dicho archivo, configurando adecuadamente los distintos parámetros de interés para su operación. Los parámetros que contiene el archivo de configuración RobotServer_P2AT.cfg son los siguientes:

- SERVER_PORT: utilizado para especificar el puerto de acceso que se quiere asociar con la operación del correspondiente servidor hardware.

- `REAL_ROBOT`: flag utilizado para indicar si se va a utilizar una plataforma robótica real (valor `1`) o un simulador de la plataforma robótica (valor `0`).

La desactivación del servidor hardware se realiza mediante el comando `kill` sobre el número de proceso asociado al mismo, tal y como se muestra en la información presentada por consola al activarlo. Es importante no especificar la opción "`-9`" en el comando `kill`, pues en este caso el servidor hardware no podría capturar la señal generada por dicho comando y sería eliminado sin poder liberar los recursos empleados en la operación del mismo. Adicionalmente también se puede desactivar la operación del servidor hardware mediante la combinación de teclas `<CTRL>+<C>` sobre la consola en la que se encuentra activo, que tiene el mismo efecto que el comando `kill` anteriormente mencionado.

Tras desactivar el servidor hardware se liberan todos los recursos reservados y se muestra el siguiente mensaje por consola:

```
**************************************************
RobotServer SHUTDOWNED
**************************************************
```

3.2.2 *RobotLibrary*

La librería *RobotLibrary* es el elemento responsable de utilizar cualquier plataforma robótica que se emplee en el sistema para la cual se haya desarrollado el correspondiente servidor hardware *RobotServer*.

Todos aquellos módulos que quieran utilizar una plataforma robótica mediante la librería *RobotLibrary* deben seguir los siguientes pasos durante el proceso de compilación:

1. Incluir el archivo `RobotLibrary.h` en el código mediante el siguiente comando:

   ```
   #include "RobotLibrary.h"
   ```

2. Añadir los archivos `RobotLibrary.h` y `RobotLibrary.o` en una ubicación accesible durante el proceso de compilación. Esta ubicación puede ser cualquier directorio base utilizado por el compilador o simplemente el mismo directorio donde se encuentra el código del módulo a compilar.

3. Compilar el módulo con el siguiente comando básico:

   ```
   [%] gcc -c Module_name.c
   ```

4. Generar el ejecutable del módulo con el siguiente comando básico:

   ```
   [%] gcc -o Module_name RobotLibrary.o Module_name.o
   ```

La librería *RobotLibrary* consta de una serie de funciones utilizadas para controlar la plataforma robótica por parte de los módulos del sistema. Es importante notar que no todas las funciones

presentes en dicha librería pueden estar desarrolladas para todas las plataformas robóticas, dependiendo de los comandos que admita la propia plataforma. En este tipo de circunstancias aquellas funciones que no hayan podido ser implementadas simplemente no realizarán ninguna operación. Se describe a continuación la utilización de estas funciones:

- **ConnectRobot:**

Función
`Robot ConnectRobot(char *host, int port)`
Descripción
Se conecta con el correspondiente servidor hardware para acceder al control de la plataforma robótica
Parámetros
`host`: dirección IP del servidor hardware `port`: puerto de acceso del servidor hardware
Valor devuelto
Correcto: puntero a una variable de tipo `Robot` para posteriores referencias Error: `-1`

Esta función permite conectarse a un determinado servidor hardware para controlar la correspondiente plataforma robótica.

Es importante notar que únicamente se permite la conexión de un módulo con un servidor hardware determinado, de forma que únicamente dicho módulo pueda controlar la plataforma robótica. Si un módulo pretende conectarse a un servidor hardware que ya está siendo utilizado quedará bloqueado en un estado inactivo hasta que dicho servidor hardware quede libre para ser utilizado.

Un módulo puede controlar distintas plataformas robóticas simultáneamente, por lo que puede utilizar esta función tantas veces como necesite.

- **GetStateRobot:**

Función
`int *GetStateRobot(Robot robot, int num_average, int dist_max)`
Descripción
Solicita el estado en el que se encuentra la plataforma robótica, devolviendo la información captada por los sensores y la posición del sistema de odometría
Parámetros
`robot`: plataforma robótica a la que se quiere enviar el comando `num_average`: número de lecturas consecutivas en el promediado de los sensores `dist_max`: lectura máxima en el filtrado de los sensores
Valor devuelto
Correcto: puntero a un array de enteros con la información de la plataforma robótica Error: `NULL`

Esta función devuelve toda la información captada del entorno por la plataforma robótica. La operación de promediado de los sensores se ha incluido para reducir posibles errores esporádicos producidos en las lecturas de los mismos, y consiste en devolver el valor medio de las últimas `num_average` lecturas. Si este parámetro tiene un valor negativo, `0` ó `1` no se realiza la operación de promediado de los sensores, por lo que únicamente se devuelve la última lectura proporcionada por los mismos.

La operación de filtrado de los sensores se ha incluido para añadir un factor de confianza sobre las lecturas proporcionadas por los mismos, y consiste en no utilizar en la operación de promediado de los sensores aquellas lecturas que superen el valor `dist_max` por no considerarse fiables. Si este parámetro tiene un valor negativo ó `0` no se realiza la operación de filtrado de los sensores. Esta operación sólo se realiza si también se lleva a cabo la operación de promediado de los sensores, y en el caso de que las `num_average` lecturas de un determinado sensor superan el valor `dist_max` se devuelve la última lectura proporcionada por dicho sensor.

El array de enteros devuelto por esta función tiene la siguiente estructura:

`state[0]`: tamaño del array ($2n + 6$ enteros)
`state[1]`: número de sensores de la plataforma robótica (n sensores)
`state[2+0]`: ángulo del sensor `1` (décimas de grado; positivo = contrario agujas reloj)
`state[2+1]`: ángulo del sensor `2` (décimas de grado; positivo = contrario agujas reloj)
...
`state[2+n-1]`: ángulo del sensor `n` (décimas de grado; positivo = contrario agujas reloj)
`state[2+n+0]`: lectura del sensor `1` (milímetros)
`state[2+n+1]`: lectura del sensor `2` (milímetros)
...
`state[2+n+n-1]`: lectura del sensor `n` (milímetros)
`state[2+2n+0]`: posición X de la plataforma robótica (milímetros)
`state[2+2n+1]`: posición Y de la plataforma robótica (milímetros)
`state[2+2n+2]`: orientación de la plataforma robótica (décimas de grado; positivo = contrario agujas reloj)
`state[2+2n+3]`: brújula de la plataforma robótica (décimas de grado; positiva = contrario agujas reloj)

Si se incorporan nuevos sensores en la plataforma robótica simplemente hay que ampliar el array de enteros que devuelve esta función para incorporar los nuevos datos.

- **TransVelRobot:**

Función
`int TransVelRobot(Robot robot, int velocity)`
Descripción
Especifica la velocidad de traslación de la plataforma robótica
Parámetros
`robot`: plataforma robótica a la que se quiere enviar el comando `velocity`: velocidad de traslación (milímetros por segundo; positiva = avance)
Valor devuelto
Correcto: `0` Error: `-1`

Esta función es no bloqueante, por lo que tras su ejecución se devuelve el control al módulo que ha realizado la llamada a la misma.

- **RotVelRobot:**

Función
`int RotVelRobot(Robot robot, int velocity)`
Descripción
Especifica la velocidad de rotación de la plataforma robótica
Parámetros
`robot`: plataforma robótica a la que se quiere enviar el comando `velocity`: velocidad de rotación (décimas de grado por segundo; positiva = contrario agujas reloj)
Valor devuelto
Correcto: `0` Error: `-1`

Esta función es no bloqueante, por lo que tras su ejecución se devuelve el control al módulo que ha realizado la llamada a la misma.

- **TransDistRobot:**

Función
`int TransDistRobot(Robot robot, int distance)`
Descripción
Especifica la distancia de traslación que debe avanzar la plataforma robótica
Parámetros
`robot`: plataforma robótica a la que se quiere enviar el comando `distance`: distancia (milímetros; positiva = avance)
Valor devuelto
Correcto: `0` Error: `-1`

Esta función es bloqueante, por lo que el módulo que ha realizado la llamada a la misma pasa a un estado inactivo hasta que se ha completado la ejecución del comando.

- **RotAngRobot:**

Función
`int RotAngRobot(Robot robot, int angle)`
Descripción
Especifica el ángulo de rotación que debe girar la plataforma robótica
Parámetros
`robot`: plataforma robótica a la que se quiere enviar el comando `angle`: ángulo (décimas de grado; positivo = contrario agujas reloj)
Valor devuelto
Correcto: `0` Error: `-1`

Esta función es bloqueante, por lo que el módulo que ha realizado la llamada a la misma pasa a un estado inactivo hasta que se ha completado la ejecución del comando.

- **StopRobot:**

Función
`int StopRobot(Robot robot)`
Descripción
Detiene la plataforma robótica
Parámetros
`robot`: plataforma robótica a la que se quiere enviar el comando
Valor devuelto
Correcto: `0`
Error: `-1`

Esta función es no bloqueante, por lo que tras su ejecución se devuelve el control al módulo que ha realizado la llamada a la misma.

- **MaxTransVelRobot:**

Función
`int MaxTransVelRobot(Robot robot, int velocity)`
Descripción
Establece la máxima velocidad de traslación para la plataforma robótica
Parámetros
`robot`: plataforma robótica a la que se quiere enviar el comando
`velocity`: velocidad (milímetros por segundo)
Valor devuelto
Correcto: `0`
Error: `-1`

Esta función es no bloqueante, por lo que tras su ejecución se devuelve el control al módulo que ha realizado la llamada a la misma.

- **MaxRotVelRobot:**

Función
`int MaxRotVelRobot(Robot robot, int velocity)`
Descripción
Establece la máxima velocidad de rotación para la plataforma robótica
Parámetros
`robot`: plataforma robótica a la que se quiere enviar el comando
`velocity`: velocidad (décimas de grado por segundo)
Valor devuelto
Correcto: `0`
Error: `-1`

Esta función es no bloqueante, por lo que tras su ejecución se devuelve el control al módulo que ha realizado la llamada a la misma.

- **TransAccelRobot:**

Función
`int TransAccelRobot(Robot robot, int acceleration)`
Descripción
Especifica la aceleración de traslación de la plataforma robótica
Parámetros
`robot`: plataforma robótica a la que se quiere enviar el comando `acceleration`: aceleración (milímetros por segundo al cuadrado)
Valor devuelto
Correcto: `0` Error: `-1`

Esta función es no bloqueante, por lo que tras su ejecución se devuelve el control al módulo que ha realizado la llamada a la misma.

- **RotAccelRobot:**

Función
`int RotAccelRobot(Robot robot, int acceleration)`
Descripción
Especifica la aceleración de rotación de la plataforma robótica
Parámetros
`robot`: plataforma robótica a la que se quiere enviar el comando `acceleration`: aceleración (décimas de grado por segundo al cuadrado)
Valor devuelto
Correcto: `0` Error: `-1`

Esta función es no bloqueante, por lo que tras su ejecución se devuelve el control al módulo que ha realizado la llamada a la misma.

- **DisconnectRobot:**

Función
`int DisconnectRobot(Robot robot)`
Descripción
Se desconecta del correspondiente servidor hardware
Parámetros
`robot`: plataforma robótica de la que se quiere desconectar
Valor devuelto
Correcto: `0` Error: `-1`

Esta función es no bloqueante, por lo que tras su ejecución se devuelve el control al módulo que ha realizado la llamada a la misma.

3.2.3 *Tools*

El uso de las aplicaciones que se utilizan con las distintas plataformas robóticas del sistema se describe a continuación.

ControlRobot

La aplicación *ControlRobot* permite controlar directamente cualquier plataforma robótica para la cual se haya desarrollado un servidor hardware *RobotServer*. Constituye una herramienta adicional muy útil en determinadas circunstancias, pues permite mover cualquier plataforma robótica directamente a través de un menú de comandos.

La ejecución de la aplicación *ControlRobot* se realiza mediante el siguiente comando:

```
[%] ./ControlRobot [RobotServer_Host] [RobotServer_Port]
```

donde los parámetros `RobotServer_Host` y `RobotServer_Port` de la línea de comandos indican la dirección IP y el puerto de acceso respectivamente donde se encuentra el correspondiente servidor hardware *RobotServer* de la plataforma robótica que se quiere controlar. Si estos parámetros no se especifican en la línea de comandos se solicitará su entrada por teclado en la propia consola.

Tras la ejecución de la aplicación *ControlRobot* aparece en la consola un menú que permite ejecutar distintos comandos sobre la plataforma robótica. Estos comandos se corresponden con las funciones desarrolladas en la librería *RobotLibrary*, por lo que se referencia al lector al apartado 3.2.2 para obtener una descripción detallada de los mismos. A modo de resumen, la operación de estos comandos es la siguiente:

- **GetState**: solicita el estado en el que se encuentra la plataforma robótica, devolviendo la información captada por los sensores y la posición del sistema de odometría.
- **TransVelRobot**: especifica la velocidad de traslación de la plataforma robótica.
- **RotVelRobot**: especifica la velocidad de rotación de la plataforma robótica.
- **TransDistRobot**: especifica la distancia de traslación que debe avanzar la plataforma robótica.
- **RotAngRobot**: especifica el ángulo de rotación que debe girar la plataforma robótica.
- **StopRobot**: detiene la plataforma robótica.
- **MaxTransVelRobot**: establece la máxima velocidad de traslación para la plataforma robótica.
- **MaxRotVelRobot**: establece la máxima velocidad de rotación para la plataforma robótica.
- **TransAccelRobot**: especifica la aceleración de traslación de la plataforma robótica.
- **RotAccelRobot**: especifica la aceleración de rotación de la plataforma robótica.
- **DisconnectRobot**: se desconecta del correspondiente servidor hardware.

Nomad200_Simulator

El simulador de la plataforma robótica *Nomad 200* proporcionado por *Nomadics* se puede ejecutar mediante el siguiente comando:

```
[%] ./Nserver
```

Su utilización se puede consultar en los manuales incluidos en el directorio `Software/Robot/Tools/Nomad200_Simulator` del CD.

3.3 Sistema *CBR*

En el directorio `Software/CBR` del CD se encuentra el software que implementa el sistema *CBR*, compuesto por los siguientes elementos:

- **CBRServer:** software que implementa el servidor *CBR* para las distintas plataformas robóticas del sistema:
  - **Nomad200:** servidor *CBR* para la plataforma robótica *Nomad 200*.
  - **P2AT:** servidor *CBR* para la plataforma robótica *Pioneer P2AT*.
- **Tools:** aplicaciones necesarias para la generación de la base de casos del sistema *CBR*.

Se describe a continuación la utilización de estos elementos.

3.3.1 *CBRServer*

El servidor *CBRServer* es el elemento responsable de implementar el sistema *CBR* diseñado [6], por lo que su activación es imprescindible para poner en funcionamiento dicho sistema *CBR*.

Tal y como ha sido desarrollado, el servidor *CBRServer* está compuesto por dos aplicaciones distintas desarrolladas en *JAVA*:

- ***CBRControlPanelET*:** aplicación necesaria para adquirir la base de casos del sistema *CBR*.
- ***CBReasoner2*:** aplicación que implementa el sistema *CBR* con la base de casos indicada y posibilita su utilización remota.

[6] El servidor *CBRServer* utilizado ha sido desarrollado por el grupo *KEMLg* (*Knowledge Engineering & Machine Learning Group*) de la *Universitat Politècnica de Catalunya* [SMCR+97].

Recuérdese que se dispone de un servidor *CBRServer* distinto para cada plataforma robótica disponible, que en el caso del sistema desarrollado se corresponden con las plataformas robóticas *Nomad 200* y *Pioneer P2AT*.

Nomad200

Para activar el servidor *CBRServer* es necesario seguir el siguiente procedimiento:

1. Ejecutar la aplicación *CBRControlPanelET* mediante el siguiente comando:

   ```
   [%] java CBRControlPanelET
   ```

   Esta aplicación permite adquirir la base de casos del sistema *CBR* a partir de un archivo de texto con la descripción de los casos [7].

   Tras su ejecución se despliega una aplicación visual donde hay que seleccionar la opción "`Importar Datos eTools`" del menú "`Fichero`". Una vez seleccionada dicha opción se despliega una ventana con el título "`Abrir fichero defs vars`", donde se debe seleccionar el archivo de texto que describe cada una de las entradas y salidas del caso. En el caso del sistema desarrollado este archivo se denomina `Nomad200_descriptors.txt`. Si por algún motivo se desea utilizar otra plataforma robótica con distinta configuración sensorial es necesario adaptar el sistema *CBR* a dicha configuración sensorial. Para ello hay que crear un nuevo archivo de texto que describa cada una de las entradas y salidas del caso de acuerdo con la configuración sensorial de la nueva plataforma robótica.

   Seguidamente se despliega otra ventana con el título "`Abrir fichero de datos`", donde se debe seleccionar el archivo de texto con la descripción de los casos que se quieren incorporar en la base de casos. Una vez seleccionado el archivo que define los casos se genera automáticamente la base de casos correspondiente.

   El último paso consiste en guardar la base de casos generada para poder ser utilizado posteriormente por el sistema *CBR*. Para ello hay que seleccionar la opción "`Salvar CBR`" del menú "`Fichero`", apareciendo una nueva ventana con el título "`Guardar CBR en...`" que permite seleccionar el nombre y la ubicación del archivo donde se desea guardar la base de casos. Se recomienda que dicho archivo se guarde en el directorio `data`.

2. Activar el correspondiente sistema *CBR* mediante el siguiente comando:

   ```
   [%] nohup java CBReasoner2 7002 data/CBR.cbr
   ```

   donde `7002` se corresponde con el puerto que se quiere asociar al servidor *CBRServer* y `data/CBR.cbr` se corresponde con el archivo que contiene la base de casos que se desea usar y que ha sido generada con la aplicación *CBRControlPanelET*. El comando `nohup` inhabilita la información mostrada por la consola del servidor *CBRServer*, redirigiendo dicha salida al archivo `nohup.out`.

[7] Este archivo de texto con la descripción de los casos es el generado por las aplicaciones `Trace2CBR` y `Cluster2CBR` descritas en el apartado 3.3.2

Pioneer P2AT

Para activar el servidor *CBRServer* es necesario seguir el siguiente procedimiento:

1. Ejecutar la aplicación *CBRControlPanelET* mediante el siguiente comando:

   ```
   [%] java CBRControlPanelET
   ```

   Esta aplicación permite adquirir la base de casos del sistema *CBR* a partir de un archivo de texto con la descripción de los casos [8].

   Tras su ejecución se despliega una aplicación visual donde hay que seleccionar la opción "`Importar Datos eTools`" del menú "`Fichero`". Una vez seleccionada dicha opción se despliega una ventana con el título "`Abrir fichero defs vars`", donde se debe seleccionar el archivo de texto que describe cada una de las entradas y salidas del caso. En el caso del sistema desarrollado este archivo se denomina `P2AT_descriptors.txt`. Si por algún motivo se desea utilizar otra plataforma robótica con distinta configuración sensorial es necesario adaptar el sistema *CBR* a dicha configuración sensorial. Para ello hay que crear un nuevo archivo de texto que describa cada una de las entradas y salidas del caso de acuerdo con la configuración sensorial de la nueva plataforma robótica.

   Seguidamente se despliega otra ventana con el título "`Abrir fichero de datos`", donde se debe seleccionar el archivo de texto con la definición de los casos que se quieren incorporar en la base de casos. Una vez seleccionado el archivo que define los casos se genera automáticamente la base de casos correspondiente.

   El último paso consiste en guardar la base de casos generada para poder ser utilizado posteriormente por el sistema *CBR*. Para ello hay que seleccionar la opción "`Salvar CBR`" del menú "`Fichero`", apareciendo una nueva ventana con el título "`Guardar CBR en...`" que permite seleccionar el nombre y la ubicación del archivo donde se desea guardar la base de casos. Se recomienda que dicho archivo se guarde en el directorio `data`.

2. Activar el correspondiente sistema *CBR* mediante el siguiente comando:

   ```
   [%] nohup java CBReasoner2 7002 data/CBR.cbr
   ```

   donde `7002` se corresponde con el puerto que se quiere asociar al servidor *CBRServer* y `data/CBR.cbr` se corresponde con el archivo que contiene la base de casos que se desea usar y que ha sido generada con la aplicación *CBRControlPanelET*. El comando `nohup` inhabilita la información mostrada por la consola del servidor *CBRServer*, redirigiendo dicha salida al archivo `nohup.out`.

3.3.2 *Tools*

Para generar la base de casos del sistema *CBR* se han desarrollado diversas aplicaciones que simplifican dicho proceso. Estas aplicaciones son las siguientes:

[8] Este archivo de texto con la descripción de los casos es el generado por las aplicaciones `Trace2CBR` y `Cluster2CBR` descritas en el apartado 3.3.2

- **TrainHuman:** aplicación visual que permite dirigir al agente mediante el teclado mientras se salva la traza de la trayectoria recorrida. Es utilizada para realizar el entrenamiento del sistema *CBR* mediante un operador humano.
- **TrainPotential:** aplicación visual que permite dirigir al agente mediante el esquema de los Campos Potenciales mientras se salva la traza de la trayectoria recorrida. Es utilizada para realizar el entrenamiento del sistema *CBR* mediante el esquema de los Campos Potenciales.
- **Trace2RAW:** aplicación que genera un mapa métrico a partir de la traza de la trayectoria recorrida. Es utilizada como herramienta de análisis de los resultados.
- **TraceSelector:** aplicación que selecciona los casos de interés de la traza de la trayectoria recorrida. Es utilizada para seleccionar los casos iniciales que formarán la base de casos.
- **Trace2CBR:** aplicación que genera un archivo de texto con la definición de los casos a incorporar en la base de casos a partir de la traza de la trayectoria recorrida. Es utilizada para generar el archivo de texto que necesita la aplicación *CBRControlPanelET*.
- **CBRSequencer:** aplicación que enumera consecutivamente todos los casos presentes en el archivo de texto con la definición de los casos. Es utilizada como herramienta para facilitar la unión de varios archivos de texto con la definición de los casos en uno solo.
- **CBR2Case:** aplicación que convierte un archivo de texto con la definición de los casos en un archivo de texto auxiliar necesario para la optimización de la base de casos. Es utilizada para generar el archivo de texto que necesitan las aplicaciones *Case2Eval* y *Case2Cluster*.
- **Case2Eval:** aplicación que optimiza la base de casos descartando aquellos casos cuya evaluación se encuentre por debajo de un umbral especificado. Es utilizada como proceso de optimización basado en umbrales de la base de casos.
- **Case2Cluster:** aplicación que optimiza la base de casos tras una segmentación de la misma. Es utilizada como proceso de optimización basado en segmentación de la base de casos.
- **Cluster2CBR:** aplicación que genera un archivo de texto con la definición de los casos a incorporar en la base de casos a partir del proceso de optimización. Es utilizada para generar el archivo de texto que necesita la aplicación *CBRControlPanelET*.
- **TraceLabel:** aplicación que etiqueta todos los casos utilizados en la traza de la trayectoria recorrida con su correspondiente nombre. Es utilizada como herramienta de análisis de los resultados.

Todas estas aplicaciones están relacionadas entre sí, de forma que operan conjuntamente en la implementación del sistema *CBR* desarrollado. La figura A.2 muestra la relación existente entre todas estas aplicaciones.

Además de las aplicaciones implementadas, en la figura A.2 aparecen distintos tipos de archivos, que son utilizados como mecanismo de intercambio de información entre estas aplicaciones. El contenido de cada uno de estos tipos de archivo es el siguiente:

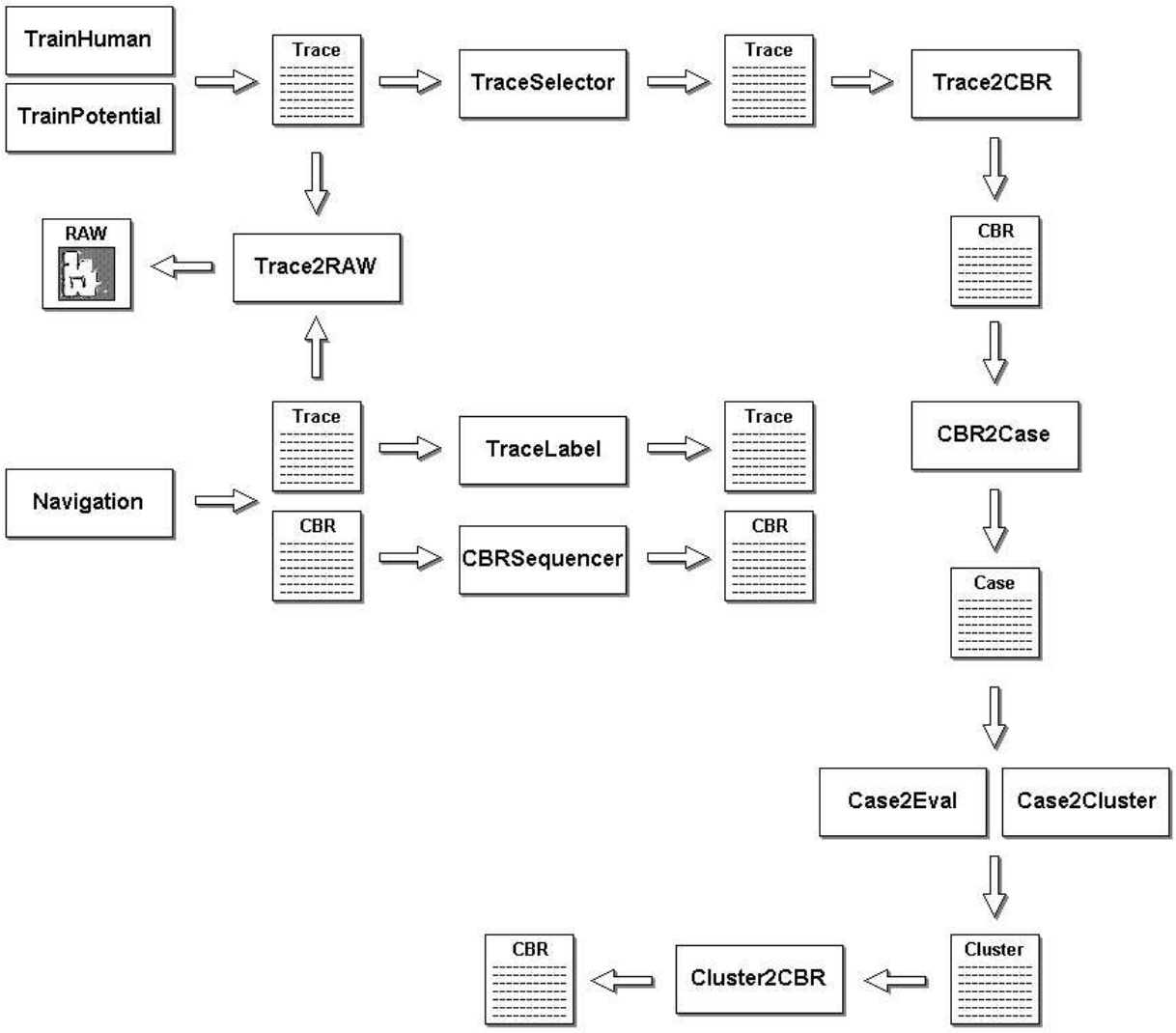

Figura A.2: Relación entre las aplicaciones utilizadas en la implementación del sistema *CBR*.

- **Trace:** archivo de texto con información sobre la trayectoria recorrida por el agente, compuesta por la posición, la orientación y la lectura de todos los sensores a lo largo de dicha trayectoria.
- **CBR:** archivo de texto con la base de casos del sistema, describiendo las entradas y las salidas de todos los casos.
- **Case:** archivo de texto con el conocimiento almacenado en el sistema, compuesto por todos los casos contenidos en la base de casos.
- **Cluster:** archivo de texto con el conocimiento optimizado a almacenar en el sistema, compuesto por el conjunto de casos derivado tras el proceso de optimización.
- **RAW:** archivo gráfico en formato *RAW* con el mapa métrico del entorno y la trayectoria recorrida por el agente.

Es importante hacer notar, por último, que la correcta ejecución de estas aplicaciones depende del valor de algunos parámetros del sistema. Para introducir el valor de estos parámetros se utiliza un módulo especial denominado *Start*, que se encarga de ubicar los distintos parámetros en el servidor central *DLAServer* para que sean accesibles por cualquier aplicación que los necesite. Es por ello por lo que estas aplicaciones utilizan la arquitectura *DLA* para obtener el valor correcto de los parámetros involucrados en su funcionamiento. Una descripción detallada del funcionamiento del módulo *Start* se puede encontrar en el apartado 3.4.

Se describe a continuación la utilización de estas aplicaciones.

TrainHuman

Esta aplicación dirige al agente mediante el teclado mientras se salva la traza de la trayectoria recorrida, y es utilizada durante el proceso de entrenamiento del agente.

La ejecución de la aplicación *TrainHuman* se realiza mediante el siguiente comando:

```
[%] ./TrainHuman [Train_File] [DLAServer_Host] [DLAServer_Port]
```

donde el parámetro `Train_File` indica el nombre del archivo *Trace* donde se desea salvar la traza, y los parámetros `DLAServer_Host` y `DLAServer_Port` de la línea de comandos indican la dirección IP y el puerto de acceso respectivamente donde se encuentra el servidor central *DLAServer*. Si estos parámetros no se especifican en la línea de comandos se solicitará su entrada por teclado en la propia consola. Para emplear el esquema local síncrono de la arquitectura *DLA* tan sólo hay que indicar la palabra `local` en el parámetro `DLAServer_Host`, ignorándose en dicho caso el parámetro `DLAServer_Port`.

Una vez ejecutada la aplicación se presenta un interfaz visual que permite utilizar las siguientes teclas para controlar al agente:

`RETURN` : se conecta/desconecta del agente
`LEFT/RIGHT`: gira a la izquierda/derecha al agente
`UP/DOWN` : mueve adelante/atrás al agente
`A/Z` : aumenta/disminuye la velocidad del agente
`SPACE` : vuelve a la velocidad inicial al agente
`F1/F2` : comienza/finaliza de salvar la traza
`Q/W` : comienza/finaliza de salvar la traza
`ESC` : sale de la aplicación

Además de controlar al agente y salvar la trayectoria recorrida en el archivo de texto especificado, esta aplicación crea el archivo de texto `Train.dat` que almacena el número de entrenamientos realizados sobre el agente. Este número es utilizado para etiquetar cada traza con un nombre distinto, añadiendo además la fecha y la hora del entrenamiento realizado y la configuración sensorial del agente que se ha utilizado.

Si el archivo `Train_File` especificado ya existe simplemente se añade la traza al final del mismo, posibilitando la utilización de un único archivo para salvar distintos entrenamientos.

Se puede detener la ejecución de la aplicación *TrainHuman* cerrando el interfaz visual o utilizando la tecla `<ESC>` sobre dicho interfaz visual.

TrainPotential

Esta aplicación dirige al agente mediante el esquema de los Campos Potenciales mientras se salva la traza de la trayectoria recorrida, y es utilizada durante el proceso de entrenamiento del agente.

La ejecución de la aplicación *TrainPotential* se realiza mediante el siguiente comando:

```
[%] ./TrainPotential [Train_File] [DLAServer_Host] [DLAServer_Port]
```

donde el parámetro `Train_File` indica el nombre del archivo *Trace* donde se desea salvar la traza, y los parámetros `DLAServer_Host` y `DLAServer_Port` de la línea de comandos indican la dirección IP y el puerto de acceso respectivamente donde se encuentra el servidor central *DLAServer*. Si estos parámetros no se especifican en la línea de comandos se solicitará su entrada por teclado en la propia consola. Para emplear el esquema local síncrono de la arquitectura *DLA* tan sólo hay que indicar la palabra `local` en el parámetro `DLAServer_Host`, ignorándose en dicho caso el parámetro `DLAServer_Port`.

Una vez ejecutada la aplicación se presenta un interfaz visual donde hay que indicar el desplazamiento en las coordenadas X e Y respecto a la posición actual del agente donde se encuentra el destino que se quiere alcanzar mediante el esquema de los Campos Potenciales, utilizando además las siguientes teclas para controlar al agente:

`RETURN` : se conecta/desconecta del agente
`A/Z` : aumenta/disminuye la velocidad del agente
`SPACE` : vuelve a la velocidad inicial al agente
`F1/F2` : comienza/finaliza de salvar la traza
`Q/W` : comienza/finaliza de salvar la traza
`ESC` : sale de la aplicación

Además de controlar al agente y salvar la trayectoria recorrida en el archivo de texto especificado, esta aplicación crea el archivo de texto `Train.dat` que almacena el número de entrenamientos realizados sobre el agente. Este número es utilizado para etiquetar cada traza con un nombre distinto, añadiendo además la fecha y la hora del entrenamiento realizado y la configuración sensorial del agente que se ha utilizado.

Si el archivo `Train_File` especificado ya existe simplemente se añade la traza al final del mismo, posibilitando la utilización de un único archivo para salvar distintos entrenamientos.

Se puede detener la ejecución de la aplicación *TrainPotential* cerrando el interfaz visual o utilizando la tecla `<ESC>` sobre dicho interfaz visual.

Trace2RAW

Esta aplicación genera un mapa métrico con la trayectoria recorrida por el agente, y es utilizada como herramienta de análisis de los resultados.

La ejecución de la aplicación *Trace2RAW* se realiza mediante el siguiente comando:

```
[%] ./Trace2RAW [Trace_File] [DLAServer_Host] [DLAServer_Port]
```

donde el parámetro `Trace_File` indica el nombre del archivo *Trace* con las trazas a procesar, y los parámetros `DLAServer_Host` y `DLAServer_Port` de la línea de comandos indican la dirección IP y el puerto de acceso respectivamente donde se encuentra el servidor central *DLAServer*. Si estos parámetros no se especifican en la línea de comandos se solicitará su entrada por teclado en la propia consola. Para emplear el esquema local síncrono de la arquitectura *DLA* tan sólo hay que indicar la palabra `local` en el parámetro `DLAServer_Host`, ignorándose en dicho caso el parámetro `DLAServer_Port`.

Una vez ejecutada la aplicación se genera un archivo *RAW* para cada una de las trazas a representar, donde el nombre de cada archivo coincide con el nombre de la traza. Las trazas utilizadas son generadas por las aplicaciones *TrainHuman* y *TrainPotential* durante el proceso de entrenamiento, o por el módulo *Navigation* durante la operación del sistema.

TraceSelector

Esta aplicación selecciona los casos de interés de las trayectorias recorridas por el agente, y es utilizada para escoger los casos que formarán la base de casos inicial del sistema.

La ejecución de la aplicación *TraceSelector* se realiza mediante el siguiente comando:

```
[%] ./TraceSelector [Trace_File] [Dist_Select] [DLAServer_Host]
[DLAServer_Port]
```

donde el parámetro `Trace_File` indica el nombre del archivo *Trace* con las trazas a procesar, el parámetro `Dist_Select` indica la distancia para la captura de nuevos casos, y los parámetros `DLAServer_Host` y `DLAServer_Port` de la línea de comandos indican la dirección IP y el puerto de acceso respectivamente donde se encuentra el servidor central *DLAServer*. Si estos parámetros no se especifican en la línea de comandos se solicitará su entrada por teclado en la propia consola. Para emplear el esquema local síncrono de la arquitectura *DLA* tan sólo hay que indicar la palabra `local` en el parámetro `DLAServer_Host`, ignorándose en dicho caso el parámetro `DLAServer_Port`.

Una vez ejecutada la aplicación se seleccionan los casos de interés para cada una de las trazas especificadas, descartando simplemente aquellos casos que se encuentren a una distancia menor que `Dist_Select` [9] del último caso almacenado. El archivo `Trace_File` especificado no es modificado, generando dos nuevos archivos de texto con los resultados obtenidos del proceso de selección: el archivo `Trace_File_out` con los casos seleccionados, y el archivo `Trace_File_sel`

[9] Este parámetro se corresponde con el umbral de disparidad U_{disp} descrito en el apartado 2.4.4 del capítulo 5.

que se corresponde con el archivo original `Trace_File` donde se han marcado los casos seleccionados con el símbolo "`(*)`". Las trazas utilizadas son generadas por las aplicaciones *TrainHuman* y *TrainPotential* durante el proceso de entrenamiento.

Trace2CBR

Esta aplicación extrae la descripción de todos los casos a partir de las trayectorias recorridas por el agente, y es utilizada para formar la base de casos del sistema.

La ejecución de la aplicación *Trace2CBR* se realiza mediante el siguiente comando:

```
[%] ./Trace2CBR [Trace_File] [CBR_File] [DLAServer_Host] [DLAServer_Port]
```

donde el parámetro `Trace_File` indica el nombre del archivo *Trace* con las trazas a procesar, el parámetro `CBR_File` indica el nombre del archivo *CBR* donde se desea salvar la base de casos, y los parámetros `DLAServer_Host` y `DLAServer_Port` de la línea de comandos indican la dirección IP y el puerto de acceso respectivamente donde se encuentra el servidor central *DLAServer*. Si estos parámetros no se especifican en la línea de comandos se solicitará su entrada por teclado en la propia consola. Para emplear el esquema local síncrono de la arquitectura *DLA* tan sólo hay que indicar la palabra `local` en el parámetro `DLAServer_Host`, ignorándose en dicho caso el parámetro `DLAServer_Port`.

Una vez ejecutada la aplicación se analizan todas las trazas y se genera un único archivo con la base de casos del sistema, etiquetando cada caso con el nombre de la traza de la cual ha sido derivado. De esta forma se pueden procesar e incorporar distintos entrenamientos simultáneamente para formar la base de casos. Las trazas utilizadas son generadas por las aplicaciones *TrainHuman* y *TrainPotential* durante el proceso de entrenamiento, o por la aplicación *TraceSelector* durante el proceso de selección.

CBRSequencer

Esta aplicación enumera consecutivamente todos los casos contenidos en la base de casos del sistema, y es utilizado para fusionar distintas bases de casos en una mayor.

La ejecución de la aplicación *CBRSequencer* se realiza mediante el siguiente comando:

```
[%] ./CBRSequencer [CBR_File] [Case_Number] [DLAServer_Host] [DLAServer_Port]
```

donde el parámetro `CBR_File` indica el nombre del archivo *CBR* con la base de casos a procesar, el parámetro `Case_Number` indica el número inicial con el que se quiere comenzar a enumerar los casos, y los parámetros `DLAServer_Host` y `DLAServer_Port` de la línea de comandos indican la dirección IP y el puerto de acceso respectivamente donde se encuentra el servidor central *DLAServer*. Si estos parámetros no se especifican en la línea de comandos se solicitará su entrada por teclado en la propia consola. Para emplear el esquema local síncrono de la arquitectura *DLA* tan sólo hay que indicar la palabra `local` en el parámetro `DLAServer_Host`, ignorándose en dicho caso el parámetro `DLAServer_Port`.

Una vez ejecutada la aplicación se enumeran consecutivamente todos los casos de la base de casos comenzando por el número inicial especificado. Su uso tiene un gran interés para facilitar la inclusión en la base de casos de los nuevos casos adaptados obtenidos por el módulo *Navigation* durante el proceso de navegación. La base de casos utilizada es generada por las aplicaciones *Trace2CBR* y *Cluster2CBR*.

CBR2Case

Esta aplicación extrae todos los casos contenidos en la base de casos del sistema, y es utilizada para convertir la base de casos en un formato más adecuado para su posterior procesamiento.

La ejecución de la aplicación *CBR2Case* se realiza mediante el siguiente comando:

```
[%] ./CBR2Case [CBR_File] [Case_File] [DLAServer_Host] [DLAServer_Port]
```

donde el parámetro `CBR_File` indica el nombre del archivo *CBR* con la base de casos a procesar, el parámetro `Case_File` indica el nombre del archivo *Case* donde se desean salvar los casos, y los parámetros `DLAServer_Host` y `DLAServer_Port` de la línea de comandos indican la dirección IP y el puerto de acceso respectivamente donde se encuentra el servidor central *DLAServer*. Si estos parámetros no se especifican en la línea de comandos se solicitará su entrada por teclado en la propia consola. Para emplear el esquema local síncrono de la arquitectura *DLA* tan sólo hay que indicar la palabra `local` en el parámetro `DLAServer_Host`, ignorándose en dicho caso el parámetro `DLAServer_Port`.

Una vez ejecutada la aplicación se extraen todos los casos contenidos en la base de casos, convirtiéndolos en otro formato más adecuado para realizar el posterior proceso de optimización. La base de casos utilizada es generada por las aplicaciones *Trace2CBR* y *Cluster2CBR*.

Case2Eval

Esta aplicación descarta todos aquellos casos de la base de casos cuya evaluación se encuentre por debajo de un umbral determinado, y es utilizada para realizar el proceso de optimización de la base de casos.

La ejecución de la aplicación *Case2Eval* se realiza mediante el siguiente comando:

```
[%] ./Case2Eval [Case_File] [Cluster_File] [Eval_Min] [DLAServer_Host]
[DLAServer_Port]
```

donde el parámetro `Case_File` indica el nombre del archivo *Case* con los casos a procesar, el parámetro `Cluster_File` indica el nombre del archivo *Cluster* donde se desean salvar los casos optimizados, el parámetro `Eval_Min` indica el umbral de evaluación para descartar los casos, y los parámetros `DLAServer_Host` y `DLAServer_Port` de la línea de comandos indican la dirección IP y el puerto de acceso respectivamente donde se encuentra el servidor central *DLAServer*. Si estos parámetros no se especifican en la línea de comandos se solicitará su entrada por teclado en la propia consola. Para emplear el esquema local síncrono de la arquitectura *DLA* tan sólo hay que indicar la palabra `local` en el parámetro `DLAServer_Host`, ignorándose en dicho caso

el parámetro `DLAServer_Port`.

Los distintos parámetros utilizados durante el proceso de optimización y descritos en el apartado 2.4.3 del capítulo 5 (C_{soft}, C_{dist}, C_{sec}, K_{soft}, K_{dist} y K_{sec}) se han incluido al principio del archivo `Case2Eval.c`:

```
/* ----- EVALUATION ----- */
#define C_SOFT 2.3e-2
#define C_DIST 2.3e-2
#define C_SEG 5.8e-5

#define K_SOFT 1.0
#define K_DIST 1.0
#define K_SEG 1.0
/* ----- EVALUATION ----- */
```

Si es necesario modificar estos valores hay que volver a compilar esta aplicación, según se ha descrito en el proceso de instalación del apartado 2.3.2.

Una vez ejecutada la aplicación se descartan todos aquellos casos cuya evaluación se encuentre por debajo del umbral especificado. El archivo con los casos optimizados generado puede ser consultado para analizar los casos seleccionados y descartados en el proceso de optimización. El conjunto de casos inicial para ser optimizado es generado por la aplicación *CBR2Case*.

Case2Cluster

Esta aplicación realiza una segmentación en clases de la base de casos del sistema, y es utilizada para realizar el proceso de optimización de la base de casos.

La ejecución de la aplicación *Case2Cluster* se realiza mediante el siguiente comando:

```
[%] ./Case2Cluster [Case_File] [Cluster_File] [Dist_Cluster] [DLAServer_Host]
[DLAServer_Port]
```

donde el parámetro `Case_File` indica el nombre del archivo *Case* con los casos a procesar, el parámetro `Cluster_File` indica el nombre del archivo *Cluster* donde se desean salvar los casos optimizados, el parámetro `Dist_Cluster` indica la distancia máxima de separación entre casos, y los parámetros `DLAServer_Host` y `DLAServer_Port` de la línea de comandos indican la dirección IP y el puerto de acceso respectivamente donde se encuentra el servidor central *DLAServer*. Si estos parámetros no se especifican en la línea de comandos se solicitará su entrada por teclado en la propia consola. Para emplear el esquema local síncrono de la arquitectura *DLA* tan sólo hay que indicar la palabra `local` en el parámetro `DLAServer_Host`, ignorándose en dicho caso el parámetro `DLAServer_Port`.

Los distintos parámetros utilizados durante el proceso de optimización y descritos en el apartado 2.4.3 del capítulo 5 (C_{soft}, C_{dist}, C_{sec}, K_{soft}, K_{dist} y K_{sec}) se han incluido al principio del archivo `Case2Cluster.c`:

```
/* ----- EVALUATION ----- */
#define C_SOFT 2.3e-2
#define C_DIST 2.3e-2
#define C_SEG 5.8e-5

#define K_SOFT 1.0
#define K_DIST 1.0
#define K_SEG 1.0
/* ----- EVALUATION ----- */
```

Si es necesario modificar estos valores hay que volver a compilar esta aplicación, según se ha descrito en el proceso de instalación del apartado 2.3.2.

Una vez ejecutada la aplicación se agrupan todos los casos en clases según el algoritmo de segmentación *MaxMin* [Mar93] con la distancia mínima entre clases que indica el parámetro `Dist_Cluster` [10]. El archivo con los casos optimizados generado puede ser consultado para analizar las clases creadas, sus prototipos, los casos contenidos en cada clase y los casos descartados en el proceso de segmentación. El conjunto de casos inicial para ser optimizado es generado por la aplicación *CBR2Case*.

Cluster2CBR

Esta aplicación extrae la descripción de todos los casos a partir de un conjunto de casos optimizados, y es utilizada para formar la base de casos del sistema.

La ejecución de la aplicación *Cluster2CBR* se realiza mediante el siguiente comando:

```
[%] ./Cluster2CBR [Cluster_File] [CBR_File] [DLAServer_Host] [DLAServer_Port]
```

donde el parámetro `Cluster_File` indica el nombre del archivo *Cluster* con los casos optimizados a procesar, el parámetro `CBR_File` indica el nombre del archivo *CBR* donde se desea salvar la base de casos, y los parámetros `DLAServer_Host` y `DLAServer_Port` de la línea de comandos indican la dirección IP y el puerto de acceso respectivamente donde se encuentra el servidor central *DLAServer*. Si estos parámetros no se especifican en la línea de comandos se solicitará su entrada por teclado en la propia consola. Para emplear el esquema local síncrono de la arquitectura *DLA* tan sólo hay que indicar la palabra `local` en el parámetro `DLAServer_Host`, ignorándose en dicho caso el parámetro `DLAServer_Port`.

Una vez ejecutada la aplicación se analizan todos los casos optimizados y se genera un archivo con la base de casos del sistema. El conjunto de casos optimizados es generado por las aplicaciones *Case2Eval* y *Case2Cluster*.

[10] Este parámetro se corresponde con la distancia mínima entre clases *Dist_Cluster* descrito en el apartado 2.5 del capítulo 5.

TraceLabel

Esta aplicación etiqueta todos los casos utilizados durante la operación del sistema con su correspondiente nombre, y es utilizada como herramienta de análisis de los resultados.

La ejecución de la aplicación *TraceLabel* se realiza mediante el siguiente comando:

```
[%] ./TraceLabel [Trace_File] [CBR_File] [DLAServer_Host] [DLAServer_Port]
```

donde el parámetro `Trace_File` indica el nombre del archivo *Trace* con las trazas a procesar, el parámetro `CBR_File` indica el nombre del archivo *CBR* con la base de casos a procesar, y los parámetros `DLAServer_Host` y `DLAServer_Port` de la línea de comandos indican la dirección IP y el puerto de acceso respectivamente donde se encuentra el servidor central *DLAServer*. Si estos parámetros no se especifican en la línea de comandos se solicitará su entrada por teclado en la propia consola. Para emplear el esquema local síncrono de la arquitectura *DLA* tan sólo hay que indicar la palabra `local` en el parámetro `DLAServer_Host`, ignorándose en dicho caso el parámetro `DLAServer_Port`.

Una vez ejecutada la aplicación se analizan todos los casos de la traza de la trayectoria recorrida por el agente durante el proceso de navegación, y se etiqueta cada caso con el nombre asignado en la base de casos del sistema. El nombre de cada caso coincide con el nombre de la traza de la cual se derivó, por lo que se puede identificar rápidamente la traza utilizada en cada uno de los casos devueltos por el sistema. Las trazas utilizadas son generadas por el módulo *Navigation* durante la operación del sistema, y la base de casos utilizada es generada por las aplicaciones *Trace2CBR* y *Cluster2CBR*.

3.4 Sistema de navegación

En el directorio `Software/Navigation` del CD se encuentra el software de los distintos módulos necesarios para implementar el sistema de navegación en entornos dinámicos no estructurados. Estos módulos son los siguientes:

- ***Start***: módulo responsable de definir todos los parámetros necesarios por los distintos módulos del sistema.
- ***IFRobot***: módulo responsable de acceder a la plataforma robótica para recoger la información captada del entorno por medio de los sensores y enviar los comandos de movimiento adecuados.
- ***Navigation***: módulo responsable de desarrollar el sistema de evitación de obstáculos.
- ***MapMetric***: módulo responsable de generar la representación métrica del entorno.
- ***PathPlanning***: módulo responsable de calcular un camino libre de obstáculos sobre la representación disponible del entorno para alcanzar el destino final.

- ***MapTopo*:** módulo responsable de calcular una ruta con las regiones a atravesar sobre la representación disponible del entorno para alcanzar el destino final.
- ***IFUser*:** módulo responsable de implementar el interfaz de usuario para capturar los comandos especificados por el usuario y mostrar la información del sistema.

Gracias al uso de la arquitectura *DLA* todos estos módulos pueden ser ejecutados y detenidos en cualquier orden. De hecho, dependiendo de la capacidad de navegación deseada en el sistema pueden no ejecutarse todos los módulos. Así mismo, la incorporación de nuevos módulos que incrementen la funcionalidad del sistema es directa sin necesidad de modificar los módulos existentes.

Se describe a continuación la utilización de estos módulos.

Start

Existen una gran cantidad de parámetros en el sistema desarrollado para configurar los distintos módulos que operan en el mismo. Muchos de estos parámetros son utilizados por varios módulos, por lo que es evidente que su especificación puede llegar a ser bastante tediosa, pues requiere la modificación conjunta de todos los parámetros en todos los módulos para la correcta operación del sistema.

Para facilitar la especificación de todos estos parámetros se ha implementado una estrategia centralizada más eficiente, que permite configurar todos los parámetros simultáneamente de una forma sencilla. Para ello se ha creado un archivo de configuración en formato texto que contiene todos los parámetros necesarios por los distintos módulos del sistema, y se ha ubicado dicho archivo de configuración en el servidor central *DLAServer*. Así pues, todos los módulos del sistema que necesiten la especificación de algún parámetro para su correcta operación simplemente tienen que acceder al archivo de configuración ubicado en el servidor central y leer el correspondiente valor para dicho parámetro. Al existir un único archivo de configuración centralizado con todos los parámetros necesarios en el sistema estos pueden ser modificados muy fácilmente independientemente de la ubicación donde se encuentren los módulos que los utilicen.

El módulo *Start* es el responsable de ubicar el archivo de configuración que contiene todos los parámetros necesarios por los distintos módulos del sistema en el servidor central *DLAServer*. Si bien todos los módulos pueden ser ejecutados en cualquier orden, hasta que no se ejecute el módulo *Start* ningún módulo comenzará su ejecución [11]. Evidentemente este comportamiento es el deseado, pues para la correcta operación de cualquier módulo es necesario especificar los parámetros necesarios para su operación.

La ejecución del módulo *Start* se realiza mediante el siguiente comando:

```
[%] ./Start [Config_File] [DLAServer_Host] [DLAServer_Port]
```

[11] Todos los módulos poseen un buzón de configuración en el que quedan bloqueados a la espera de que el módulo *Start* inicie su actividad. Tras iniciar su actividad dicho módulo escribe en el buzón de configuración, siendo activados el resto de módulos del sistema.

donde el parámetro `Config_File` indica el nombre del archivo de configuración que contiene los parámetros del sistema, y los parámetros `DLAServer_Host` y `DLAServer_Port` de la línea de comandos indican la dirección IP y el puerto de acceso respectivamente donde se encuentra el servidor central *DLAServer*. Si estos parámetros no se especifican en la línea de comandos se solicitará su entrada por teclado en la propia consola. Para emplear el esquema local de la arquitectura de control *DLA* tan sólo hay que indicar la palabra `local` en el parámetro `DLAServer_Host`, ignorándose en dicho caso el parámetro `DLAServer_Port`.

Se puede detener la ejecución del módulo *Start* mediante la combinación de teclas `<CTRL>+<C>` sobre la consola en la que se encuentra ejecutándose. Una vez que todos los módulos han iniciado su actividad quedan configurados los parámetros necesarios por los mismos, por lo que este módulo no vuelve a ser necesario hasta que alguno de ellos se deba volver a ejecutar. Si se desea modificar algún parámetro en el sistema es necesario detener el módulo *Start* y todos los módulos, modificar el archivo de configuración que contiene los parámetros del sistema, y volver a ejecutar el módulo *Start* y el resto de módulos para que vuelvan a configurar los nuevos parámetros.

Todos los parámetros utilizados en el sistema están descritos en los archivos de configuración `Nomad200.cfg` y `P2AT.cfg` [12]. Estos parámetros se han agrupado según los módulos en los cuales se ha definido su significado. Es importante notar que esta clasificación es meramente una división para facilitar su explicación, pues cualquier parámetro definido en el archivo de configuración puede se utilizado por cualquier módulo del sistema. Al describir la utilización de cada módulo en los siguientes apartados se describirá el significado de cada uno de los parámetros del archivo de configuración asociados al mismo.

Los parámetros del archivo de configuración asociados al módulo *Start* son los siguientes:

`NUM_SONARS`: número de sensores sonar del agente
`ANG_SONARi`: ángulo del sonar "i" (décimas de grado; positivo = contrario agujas reloj)
`SIZE_ROBOT`: tamaño del agente (diámetro de la circunferencia circunscrita) (milímetros)
`HOST_ROBOT`: dirección IP del servidor hardware *RobotServer*
`PORT_ROBOT`: puerto de acceso del servidor hardware *RobotServer*
`HOST_CBR`: dirección IP del servidor *CBRServer*
`PORT_CBR`: puerto de acceso del servidor *CBRServer*

Es importante matizar que el sistema permite utilizar un número menor de sensores que los que posee el propio agente, gracias a los parámetros `NUM_SONARS` y `ANG_SONARi`. Estos parámetros indican la configuración sensorial que se quiere utilizar en el sistema, no teniendo que coincidir con la configuración sensorial de la plataforma robótica. Incluso es posible especificar distintos ángulos para los sensores a utilizar de los ángulos que posee la plataforma robótica, aunque en este caso el sistema adaptará automáticamente la configuración sensorial especificada con la disponible más cercana.

[12] En estos archivos se utiliza el carácter ";" al principio de una línea para indicar que se trata de un comentario.

IFRobot

El módulo *IFRobot* es el responsable de acceder a la plataforma robótica para recoger la información captada del entorno por medio de los sensores y enviar los comandos de movimiento adecuados. La información compartida por este módulo deberá ser procesada por el resto de módulos para generar la respuesta adecuada a los estímulos captados. Es evidente que la operación del resto de módulos carece de sentido si este módulo no está operativo y recoge la adecuada información del entorno, por lo que al igual que ocurre con el módulo *Start* hasta que no se ejecute el módulo *IFRobot* ningún módulo comenzará su ejecución [13].

La ejecución del módulo *IFRobot* se realiza mediante el siguiente comando:

```
[%] ./IFRobot [DLAServer_Host] [DLAServer_Port]
```

donde los parámetros `DLAServer_Host` y `DLAServer_Port` de la línea de comandos indican la dirección IP y el puerto de acceso respectivamente donde se encuentra el servidor central *DLAServer*. Si estos parámetros no se especifican en la línea de comandos se solicitará su entrada por teclado en la propia consola. Para emplear el esquema local síncrono de la arquitectura *DLA* tan sólo hay que indicar la palabra `local` en el parámetro `DLAServer_Host`, ignorándose en dicho caso el parámetro `DLAServer_Port`.

Se puede detener la ejecución del módulo *IFRobot* mediante la combinación de teclas `<CTRL>+<C>` sobre la consola en la que se encuentra ejecutándose.

Los parámetros del archivo de configuración asociados al módulo *IFRobot* son los siguientes:

`U_SEC`: distancia de seguridad para detener al agente (milímetros)
`VEL_TRANS_MAX`: máxima velocidad de traslación (milímetros por segundo)
`VEL_ROT_MAX`: máxima velocidad de rotación (décimas de grado por segundo)

El parámetro `U_SEC` se corresponde con el parámetro U_{sec} descrito en el apartado 1.1 del capítulo 4. Los parámetros `VEL_TRANS_MAX` y `VEL_ROT_MAX` determinan la máxima velocidad que será enviada al agente. Si algún comando genera una velocidad mayor que las indicadas será ignorado, enviando en dicho caso el valor de velocidad máxima.

El resto de parámetros del archivo de configuración utilizados por el módulo *IFRobot* así como los módulos que los definen son los siguientes:

[13] Todos los módulos poseen un buzón de inicialización en el que quedan bloqueados a la espera de que el módulo *IFRobot* inicie su actividad. Tras iniciar su actividad dicho módulo escribe en el buzón de inicialización, siendo activados el resto de módulos del sistema.

`NUM_SONARS`: número de sensores sonar del agente [**Start**]
`ANG_SONARi`: ángulo del sonar "i" (décimas de grado; positivo = contrario agujas reloj) [**Start**]
`HOST_ROBOT`: dirección IP del servidor hardware *RobotServer* [**Start**]
`PORT_ROBOT`: puerto de acceso del servidor hardware *RobotServer* [**Start**]
`U_PROX`: distancia para evitación de obstáculos (milímetros) [**Navigation**]
`G_REP`: coeficiente de ponderación de la fuerza repulsiva [**Navigation**]
`RANGE_MAX`: máxima distancia a actualizar del mapa métrico (milímetros) [**MapMetric**]

Navigation

El módulo *Navigation* es el responsable de desarrollar el sistema de evitación de obstáculos. Se ha configurado de forma que se puede seleccionar entre el sistema *CBR* y el esquema de los Campos Potenciales para evitar los obstáculos del entorno. Si bien el sistema *CBR* ha demostrado ser más útil y versátil, el esquema de los Campos Potenciales no precisa etapa de entrenamiento ni el servidor *CBRServer* para su operación, por lo que requiere menos recursos y puede ser más rápido de utilizar para aplicaciones no muy exigentes en cuanto a capacidad de navegación.

Este módulo también salva la traza de la trayectoria recorrida por el agente para un posterior análisis, la cual contiene información sobre la posición, la orientación y la lectura de todos los sensores del agente. Dicha traza se salva en el archivo de texto `Probe.txt`. Si dicho archivo ya existe simplemente se añade la traza al final del mismo, posibilitando la utilización de un único archivo para salvar distintas pruebas. Adicionalmente, se crea el archivo `Probe.dat` que almacena el número de pruebas realizadas sobre el agente. Este número es utilizado para etiquetar cada traza con un nombre distinto, añadiendo además del nombre la fecha y hora de la prueba realizada y la configuración sensorial del agente que se ha utilizado. Este archivo de traza puede ser posteriormente utilizado con las aplicaciones *Trace2RAW* y *TraceLabel* descritas en el apartado 3.3.

Por último, si se utiliza el sistema *CBR* para evitar los obstáculos del entorno este módulo también salva todos los casos adaptados que se vayan produciendo durante el proceso de navegación. Dichos casos se salvan en el archivo de texto `CBR_adap.txt` con la descripción de los casos a incorporar en la base de casos, los cuales pueden ser directamente incluidos en cualquier base de casos mediante la aplicación *CBRSequencer* descrita en el apartado 3.3.

La ejecución del módulo *Navigation* se realiza mediante el siguiente comando:

```
[%] ./Navigation [DLAServer_Host] [DLAServer_Port]
```

donde los parámetros `DLAServer_Host` y `DLAServer_Port` de la línea de comandos indican la dirección IP y el puerto de acceso respectivamente donde se encuentra el servidor central *DLAServer*. Si estos parámetros no se especifican en la línea de comandos se solicitará su entrada por teclado en la propia consola. Para emplear el esquema local síncrono de la arquitectura *DLA* tan sólo hay que indicar la palabra `local` en el parámetro `DLAServer_Host`, ignorándose en dicho caso el parámetro `DLAServer_Port`.

Se puede detener la ejecución del módulo *Navigation* mediante la combinación de teclas <CTRL>+<C> sobre la consola en la que se encuentra ejecutándose.

Los parámetros del archivo de configuración asociados al módulo *Navigation* son los siguientes:

U_ADAP: distancia para adaptar el caso devuelto
U_PROX: distancia para evitación de obstáculos (milímetros)
G_ATTRAC: coeficiente de ponderación de la fuerza atractiva
G_REP: coeficiente de ponderación de la fuerza repulsiva
TYPE_AVOID: esquema de los Campos Potenciales (1) o sistema CBR (2)
DIST_TARGET: distancia para considerar el destino alcanzado (milímetros)

El parámetro U_ADAP se corresponde con el parámetro U_{adap} descrito en el apartado 2.4.2 del capítulo 5. El parámetro U_PROX se corresponde con el parámetro U_{prox} descrito en el apartado 3.1 del capítulo 5. Los parámetros G_ATTRAC y G_REP se corresponden con los parámetros G_{attrac} y G_{rep} descritos en el apartado 3.1 del capítulo 5 respectivamente. El parámetro TYPE_AVOID permite seleccionar el esquema de evitación de obstáculos deseado. El parámetro DIST_TARGET indica la distancia mínima a la cual se considera que el destino ha sido alcanzado.

El resto de parámetros del archivo de configuración utilizados por el módulo *Navigation* así como los módulos que los definen son los siguientes:

NUM_SONARS: número de sensores sonar del agente [**Start**]
HOST_CBR: dirección IP del servidor *CBRServer* [**Start**]
PORT_CBR: puerto de acceso del servidor *CBRServer* [**Start**]

MapMetric

El módulo *MapMetric* es el responsable de generar la representación métrica probabilística del entorno.

La ejecución del módulo *MapMetric* se realiza mediante el siguiente comando:

```
[%] ./MapMetric [DLAServer_Host] [DLAServer_Port]
```

donde los parámetros DLAServer_Host y DLAServer_Port de la línea de comandos indican la dirección IP y el puerto de acceso respectivamente donde se encuentra el servidor central *DLAServer*. Si estos parámetros no se especifican en la línea de comandos se solicitará su entrada por teclado en la propia consola. Para emplear el esquema local síncrono de la arquitectura *DLA* tan sólo hay que indicar la palabra local en el parámetro DLAServer_Host, ignorándose en dicho caso el parámetro DLAServer_Port.

Se puede detener la ejecución del módulo *MapMetric* mediante la combinación de teclas <CTRL>+<C> sobre la consola en la que se encuentra ejecutándose.

Los parámetros del archivo de configuración asociados al módulo *MapMetric* son los siguientes:

NUM_COL: número de columnas del mapa métrico
NUM_ROW: número de filas del mapa métrico
SIZE_CELL: tamaño de la celda del mapa métrico (milímetros)
UNC_ARC: anchura del lóbulo principal del sensor sonar (décimas de grado)
P_OCC: incremento de probabilidad para celdas ocupadas (%)
P_FREE: reducción de probabilidad para celdas libres (%)
U_PATH: celdas modificadas para planificación de caminos
U_TOPO: celdas modificadas para planificación de rutas
RANGE_MAX: máxima distancia a actualizar del mapa métrico (milímetros)
MAX_OCC: máximo número de veces para incrementar la misma celda ocupada
MAX_FREE: máximo número de veces para reducir la misma celda libre

Los parámetros NUM_COL y NUM_ROW se corresponden con los parámetros Num_Col y Num_Row descritos en el apartado 2.1 del capítulo 4 respectivamente. El parámetro SIZE_CELL se corresponde con el parámetro $Size\_Cell$ descrito en el apartado 2.1 del capítulo 4. El parámetro UNC_ARC se corresponde con el parámetro 2β descrito en el apartado 2.1 del capítulo 4. Los parámetros P_OCC y P_FREE se corresponden con los parámetros P_{occ} y P_{free} descritos en el apartado 2.1 del capítulo 4 respectivamente. Los parámetros U_PATH y U_TOPO se corresponden con los parámetros U_{path} y U_{topo} descritos en el apartado 2.2 del capítulo 4 respectivamente. El parámetro RANGE_MAX indica la máxima distancia del mapa métrico que se desea actualizar en cada una de las lecturas de los sensores. Los parámetros MAX_OCC y MAX_FREE indican el número de veces consecutivas que se puede incrementar o reducir la probabilidad de ocupación de la misma celda respectivamente. Si su valor es igual a -1 no tienen ningún efecto en la generación del mapa métrico. Estos parámetros pretenden limitar el efecto de lecturas erróneas de los sensores sonar, pues debido a los ángulos de incidencia y reflexión algunos obstáculos del mapa métrico pueden ser totalmente borrados si no se detectan correctamente.

El resto de parámetros del archivo de configuración utilizados por el módulo *MapMetric* así como los módulos que los definen son los siguientes:

NUM_SONARS: número de sensores sonar del agente **[Start]**
SIZE_ROBOT: tamaño del agente (diámetro de la circunferencia circunscrita) (milímetros) **[Start]**
P_OBST: umbral de probabilidad para obstáculos (%) **[PathPlanning]**

PathPlanning

El módulo *PathPlanning* es el responsable de calcular un camino libre de obstáculos sobre la representación disponible del entorno para alcanzar el destino final.

La ejecución del módulo *PathPlanning* se realiza mediante el siguiente comando:

```
[%] ./PathPlanning [DLAServer_Host] [DLAServer_Port]
```

donde los parámetros DLAServer_Host y DLAServer_Port de la línea de comandos indican la dirección IP y el puerto de acceso respectivamente donde se encuentra el servidor central *DLAServer*. Si estos parámetros no se especifican en la línea de comandos se solicitará su entrada por teclado en la propia consola. Para emplear el esquema local síncrono de la arquitectura *DLA* tan

sólo hay que indicar la palabra `local` en el parámetro `DLAServer_Host`, ignorándose en dicho caso el parámetro `DLAServer_Port`.

Se puede detener la ejecución del módulo *PathPlanning* mediante la combinación de teclas `<CTRL>+<C>` sobre la consola en la que se encuentra ejecutándose.

Los parámetros del archivo de configuración asociados al módulo *PathPlanning* son los siguientes:

`P_OBST`: umbral de probabilidad para obstáculos (%)
`DIST_FREE`: distancia a los obstáculos (milímetros)

El parámetro `P_OBST` se corresponde con el parámetro P_{obst} descrito en el apartado 2.1 del capítulo 6. El parámetro `DIST_FREE` está relacionado con el parámetro *Obst_Grow* descrito en el apartado 2.1 del capítulo 6. Recuérdese que el parámetro *Obst_Grow* indica el número de celdas utilizadas en el crecimiento de obstáculos del proceso de planificación de caminos. No obstante, en lugar de indicar este parámetro directamente se ha preferido indicar la distancia que se quiere mantener entre el agente y los obstáculos del entorno mediante el parámetro `DIST_FREE`. La posterior traducción de este parámetro al necesario *Obst_Grow* se realiza muy fácilmente conociendo el tamaño de la celda del mapa métrico *Size_Cell*:

$$Obst\_Grow = DIST\_FREE / Size\_Cell \tag{A.1}$$

redondeando el valor resultante hacia el mayor entero más cercano.

El resto de parámetros del archivo de configuración utilizados por el módulo *PathPlanning* así como los módulos que los definen son los siguientes:

`NUM_SONARS`: número de sensores sonar del agente [**Start**]
`DIST_TARGET`: distancia para considerar el destino alcanzado (milímetros) [**Navigation**]
`NUM_COL`: número de columnas del mapa métrico [**MapMetric**]
`NUM_ROW`: número de filas del mapa métrico [**MapMetric**]
`SIZE_CELL`: tamaño de la celda del mapa métrico (milímetros) [**MapMetric**]

MapTopo

El módulo *MapTopo* es el responsable de calcular una ruta con las regiones a atravesar sobre la representación disponible del entorno para alcanzar el destino final.

La ejecución del módulo *MapTopo* se realiza mediante el siguiente comando:

```
[%] ./MapTopo [DLAServer_Host] [DLAServer_Port]
```

donde los parámetros `DLAServer_Host` y `DLAServer_Port` de la línea de comandos indican la dirección IP y el puerto de acceso respectivamente donde se encuentra el servidor central *DLAServer*. Si estos parámetros no se especifican en la línea de comandos se solicitará su entrada por teclado en la propia consola. Para emplear el esquema local síncrono de la arquitectura *DLA* tan sólo hay que indicar la palabra `local` en el parámetro `DLAServer_Host`, ignorándose en dicho

caso el parámetro `DLAServer_Port`.

Se puede detener la ejecución del módulo *MapTopo* mediante la combinación de teclas `<CTRL>+<C>` sobre la consola en la que se encuentra ejecutándose.

Los parámetros del archivo de configuración asociados al módulo *MapTopo* son los siguientes:

`A_MIN`: número mínimo de celdas para formar una región
`DIST_MAX`: distancia máxima entre regiones

El parámetro `A_MIN` se corresponde con el parámetro A_{min} descrito en el apartado 3.2 del capítulo 6. El parámetro `DIST_MAX` se corresponde con el parámetro $Dist\_Max$ descrito en el apartado 3.2 del capítulo 6.

El resto de parámetros utilizados en el proceso de generación de la representación topológica del entorno descritos en el apartado 3.2 del capítulo 6 (U_{occ}, U_{free}, C_{occ}, C_{free}, C_{not} y $Obst\_Grow$) se han incluido al principio del archivo `Pyramid.h`:

```
/* ----- PYRAMID ----- */
#define U_OCC 60
#define U_FREE 40
#define C_OCC 100
#define C_FREE 0
#define C_NOT 50
#define OBST_GROW 1
/* ----- PYRAMID ----- */
```

Si es necesario modificar estos valores hay que volver a compilar este módulo, según se ha descrito en el proceso de instalación del apartado 2.4.

El resto de parámetros del archivo de configuración utilizados por el módulo *MapTopo* así como los módulos que los definen son los siguientes:

`NUM_SONARS`: número de sensores sonar del agente [**Start**]
`DIST_TARGET`: distancia para considerar el destino alcanzado (milímetros) [**Navigation**]
`NUM_COL`: número de columnas del mapa métrico [**MapMetric**]
`NUM_ROW`: número de filas del mapa métrico [**MapMetric**]
`SIZE_CELL`: tamaño de la celda del mapa métrico (milímetros) [**MapMetric**]

IFUser

El módulo *IFUser* es el responsable de implementar el interfaz de usuario para capturar los comandos especificados por el usuario y mostrar la información del sistema.

La ejecución del módulo *IFUser* se realiza mediante el siguiente comando:

```
[%] ./IFUser [DLAServer_Host] [DLAServer_Port]
```

donde los parámetros `DLAServer_Host` y `DLAServer_Port` de la línea de comandos indican la dirección IP y el puerto de acceso respectivamente donde se encuentra el servidor central *DLAServ-*

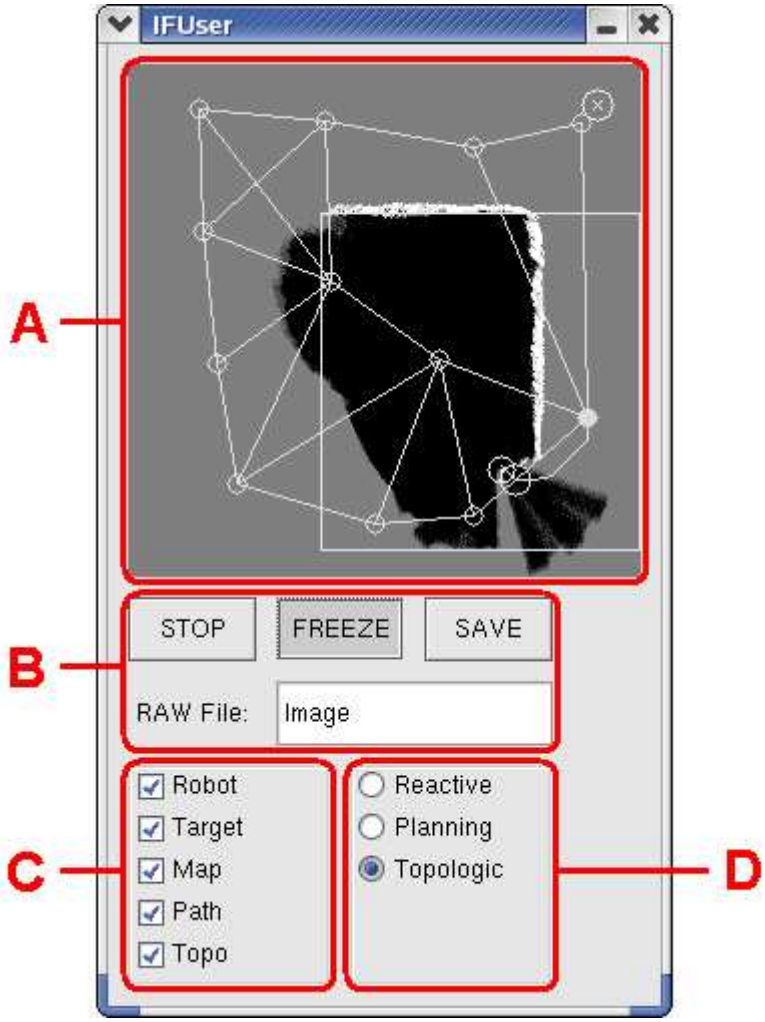

Figura A.3: Interfaz de usuario del módulo *IFUser*.

er. Si estos parámetros no se especifican en la línea de comandos se solicitará su entrada por teclado en la propia consola. Para emplear el esquema local síncrono de la arquitectura *DLA* tan sólo hay que indicar la palabra `local` en el parámetro `DLAServer_Host`, ignorándose en dicho caso el parámetro `DLAServer_Port`.

Una vez ejecutado el módulo *IFUser* presenta el interfaz visual que se muestra en la figura A.3, cuyo uso es bastante intuitivo. La utilidad de las distintas zonas que lo componen es la que se describe a continuación:

- **Zona A:** en esta zona se muestra toda la información relevante del sistema mediante una imagen:
  - **Robot:** posición actual y orientación del agente en el entorno. El agente se representa mediante un círculo con una línea blanca para indicar la orientación del mismo. El tamaño del círculo es proporcional al tamaño del agente (parámetro `SIZE_ROBOT`).
  - **Target:** posición del destino final en el entorno. El destino final se representa me-

diante un círculo con una "X". El tamaño del círculo es proporcional a la distancia especificada para considerar el destino alcanzado (parámetro `DIST_TARGET`).

- **Map:** mapa métrico del entorno. En el mapa métrico se representan los obstáculos en color blanco, las zonas libres en color negro y las zonas no exploradas en color gris.
- **Path:** *bounding box* del mapa métrico utilizado, camino calculado y siguiente destino intermedio. La *bounding box* se representa mediante un rectángulo, el camino calculado se representa mediante un conjunto de líneas, y el siguiente destino intermedio se representa mediante un círculo. El tamaño del círculo es proporcional a la distancia especificada para considerar el destino alcanzado (parámetro `DIST_TARGET`).
- **Topo:** mapa topológico del entorno y ruta calculada. El mapa topológico se representa mediante un grafo compuesto por círculos para los nodos y líneas para los enlaces entre los mismos, y la ruta calculada se representa rellenando de color blanco los nodos que pertenecen a la misma.

Esta zona sirve también para especificar el destino final que se desea alcanzar. Para ello únicamente hay que hacer un *click* con el botón izquierdo del ratón en la posición de la imagen que se desee alcanzar. Una vez especificado el destino final el agente comenzará a navegar hacia el mismo según el nivel de navegación seleccionado en la Zona D del interfaz visual.

- **Zona B:** en esta zona aparecen una serie de botones que permiten realizar algunas operaciones básicas:
  - **Botón STOP:** este botón sirve para detener al agente, anulando el comando de navegación actual.
  - **Botón FREEZE:** este botón sirve para detener temporalmente al agente, aunque no anula el comando de navegación actual. Si se vuelve a pulsar este botón el agente reanuda su operación de acuerdo al comando de navegación actual. Cuando se activa este botón no es posible modificar el comando de navegación actual. Esta funcionalidad es útil para la depuración del sistema, al permitir detener el agente temporalmente y analizar los resultados obtenidos, tras lo cual se puede reanudar su operación normal.
  - **Botón SAVE:** este botón sirve para salvar en un archivo gráfico en formato *RAW* la imagen mostrada en la zona A del interfaz visual. El nombre del archivo generado es el que se muestra en el campo "`RAW File:`". No es necesario especificar la extensión "`.raw`", pues es automáticamente añadida al nombre del archivo indicado en el campo anterior. Esta funcionalidad es útil para la depuración del sistema, al permitir salvar los resultados obtenidos para su posterior análisis.
- **Zona C:** en esta zona aparecen una serie de casillas de verificación que permiten seleccionar la información que se desea mostrar en la imagen de la zona A:
  - **Robot:** posición actual y orientación del agente en el entorno.
  - **Target:** posición del destino final en el entorno.

  - **Map:** mapa métrico del entorno.
  - **Path:** *bounding box* del mapa métrico utilizado, camino calculado y siguiente destino intermedio. Esta información sólo se muestra en los niveles de Navegación Planificada o de Navegación Topológica.
  - **Topo:** mapa topológico del entorno y ruta calculada. Esta información sólo se muestra en el nivel de Navegación Topológica.

- **Zona D:** en esta zona aparecen una serie de casillas de selección que permiten escoger el nivel de navegación deseado en el sistema:

  - **Reactive:** esta casilla sirve para utilizar el nivel de Navegación Reactiva.
  - **Planning:** esta casilla sirve para utilizar el nivel de Navegación Planificada.
  - **Topologic:** esta casilla sirve para utilizar el nivel de Navegación Topológica.

Se puede detener la ejecución del módulo *IFUser* cerrando el interfaz visual o mediante la combinación de teclas `<CTRL>+<C>` sobre la consola en la que se encuentra ejecutándose.

Los parámetros del archivo de configuración utilizados por el módulo *IFUser* así como los módulos que los definen son los siguientes:

`NUM_SONARS`: número de sensores sonar del agente [**Start**]
`SIZE_ROBOT`: tamaño del agente (diámetro de la circunferencia circunscrita) (milímetros) [**Start**]
`DIST_TARGET`: distancia para considerar el destino alcanzado (milímetros) [**Navigation**]
`NUM_COL`: número de columnas del mapa métrico [**MapMetric**]
`NUM_ROW`: número de filas del mapa métrico [**MapMetric**]
`SIZE_CELL`: tamaño de la celda del mapa métrico (milímetros) [**MapMetric**]