Arquitectura híbrida para navegación mediante esquemas de memoria compartida distribuida

E.J. Pérez, C.Urdiales, F. Sandoval
Dpto. Tecnología Electrónica – E.T.S.I. Telecomunicación
Universidad de Málaga, Campus de Teatinos, 29071, Málaga
Email: edu@dte.uma.es

Resumen

Este trabajo presenta una nueva arquitectura híbrida para la navegación de agentes autónomos móviles. La arquitectura propuesta se basa en la descomposición jerárquica del comportamiento emergente de navegación en sucesivas capas, las cuales proporcionan comportamientos cada vez más complejos según una filosofía *bottom-up*. La comunicación entre capas se ha optimizado para la distribución de procesos concurrentes entre una o varias máquinas y la carga computacional se controla activando o inhibiendo selectivamente los procesos necesarios en un momento dado según su horizonte temporal. La arquitectura propuesta ha sido satisfactoriamente probada en un robot real.

1. Introducción

El desarrollo de la robótica en las últimas décadas ha supuesto la necesidad de organizar los recursos computacionales de las máquinas implicadas para manejar la enorme cantidad de información necesaria para llevar a cabo tareas cada vez más complejas. A efectos de tratar con la creciente complejidad de estos sistemas el campo de las arquitecturas de control ha adquirido una creciente importancia en los últimos años. Mientras que la algoritmia entra en el campo de la estructura de la arquitectura, los mecanismos de intercambio de información y gestión de recursos computacionales entran en el campo del estilo de la arquitectura [1]. Los sistemas actuales requieren trabajo concurrente en tiempo real que difícilmente puede ser implementado con técnicas de programación convencionales. A este respecto, han ido surgiendo arquitecturas de control más o menos específicas que no obstante se pueden reducir a alguno de estos paradigmas [1]: i) delibertativo; ii) reactivo; iii) híbrido.

El más extendido es el paradigma híbrido, el cual pretende solucionar los problemas de los paradigmas deliberativo y reactivo. No obstante, no existe una metodología claramente establecida para la implementación de una arquitectura híbrida, por lo que han aparecido numerosas variantes adaptadas al problema particular que pretenden resolver.

El objetivo del presente trabajo es la integración de distintos módulos de navegación para un agente autónomo móvil, tal como se describen en el apartado 2. Dicha arquitectura deberá ser lo suficientemente flexible para permitir la comunicación entre módulos con diferente horizonte temporal y requisitos computacionales, así como para soportar la rápida integración de nuevos módulos y sensores. Las prestaciones que caracterizan una arquitectura de este tipo se justifican y presentan en el apartado 3. El apartado 4 presenta pruebas con un robot real *Pioneer P2-AT* en un entorno sin explorar para demostrar que el estilo soporta de forma correcta la interacción entre todos los módulos deseados. Finalmente, el apartado 5 incluye conclusiones y líneas futuras de investigación.

2. Estructura

Una de las ventajas de las arquitecturas modulares es la posibilidad de desarrollar y probar los módulos que las constituyen independientemente. Así, se han desarrollado de forma aislada: i) sistema de navegación reactivo basado en *Case Based Reasoning* y cuya principal novedad es la capacidad de aprender de un supervisor humano sin necesidad de establecer un modelo analítico ni de efectuar estudios cinemáticos [2]; ii) sistema

jerárquico de representación del entorno capaz de extraer un modelo topológico de una rejilla de ocupación preservando la información geométrica; de esta forma, el proceso de desambiguación de la información topológica es sencillo y la actualización del modelo inmediata [3]; iii) método de cálculo de trayectorias jerárquico capaz de operar a nivel topológico para después seguir el camino a nivel métrico, con el consecuente ahorro de carga computacional [4].

El objetivo del presente trabajo es la integración de todos estos módulos mediante una arquitectura cuyo estilo permita un intercambio ágil de información, minimizando el tráfico necesario y que permita manejar de forma eficiente y sencilla todos los módulos desarrollados y los que puedan desarrollarse en un futuro.

Los módulos disponibles hasta ahora se encuentran representados en la Figura 1. Cabe resaltar que el módulo de localización actualmente se basa únicamente en odometría.

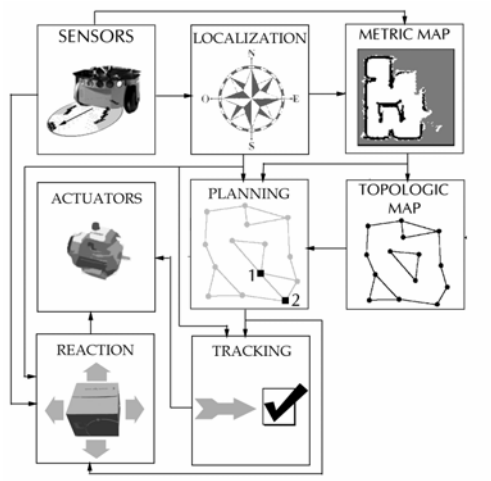

Figura 1. Estructura de la arquitectura

3. Estilo

Aunque existen diversas recomendaciones para implementar arquitecturas eficientes para el control de sistemas robóticos [1] [5], la mayoría de los sistemas tienden a ser implementados mediante una filosofía *ad hoc*, puesto que estos sistemas están fuertemente influenciados por el hardware del robot y por la aplicación final que se desee implementar. Posiblemente, el factor más crítico a la hora de implementar un sistema robótico sea el de manejar la creciente complejidad de las interacciones entre los distintos módulos que lo componen. Desde el punto de vista de la ingeniería esta complejidad se aborda con la *modularidad*, es decir, mediante la división del sistema en subsistemas menores (conocidos como módulos) que, combinados convenientemente, desarrollen la aplicación que se desea implementar. Estos subsistemas deben operar concurrentemente, y para ello es imprescindible mantener un continuo intercambio de información entre ellos. En este sentido, es necesario proporcionar los mecanismos adecuados de intercambio de información que garanticen y faciliten el funcionamiento de estos subsistemas. Existen numerosos esquemas para facilitar los mecanismos de intercambio de información entre módulos, como conexiones punto a punto [6], agentes enrutadores de tráfico [7][8][9] o esquemas de memoria compartida [10].

3.1. Filosofía

Si se requiere una alta flexibilidad a la hora de ubicar los distintos módulos es preferible usar un esquema de memoria compartida [11], por lo que éste ha sido el esquema escogido para la implementación de la arquitectura. En el sistema propuesto, los diferentes módulos se distribuyen en distintas máquinas, las cuales pueden poseer diferentes sistemas operativos. Además, los módulos pueden ser desarrollados en distintos lenguajes de programación y distintas plataformas. Esto dota al sistema de una gran flexibilidad a la hora de ser desarrollado. El intercambio de información entre los distintos módulos se posibilita mediante un servidor central, al cual se conectan todos y envían los datos que quieren compartir, almacenados en zonas compartidas conocidas como *conexiones*. Este servidor es el encargado de controlar el acceso a los datos compartidos, implementando un eficiente modelo de *memoria compartida distribuida*. Este esquema posee numerosas ventajas, como son:

a) *Flexibilidad*: los módulos pueden ser aisladamente desarrollados y depurados, al

ser completamente independientes del resto de módulos del sistema, ya que únicamente interaccionan con el servidor central.

b) *Extensibilidad*: al ser cada módulo completamente independiente del resto de módulos, nuevos módulos pueden ser añadidos al sistema sin necesidad de rescribir el resto de módulos.

c) *Escalabilidad*: al seguir una filosofía distribuida, si es necesaria mayor capacidad de cómputo se pueden añadir nuevos módulos en nuevos ordenadores sin modificar los demás. Incluso se pueden ejecutar varios servidores centrales para distribuir el tráfico generado si se detecta que es excesivo para un único servidor central.

d) *Portabilidad*: los módulos pueden ser desarrollados sobre distintas plataformas y lenguajes de programación, según las necesidades de cada uno.

3.2. Prestaciones

Este esquema fue propuesto originalmente por Dulimarta [10]. Sin embargo, en su desarrollo se empleó un único servidor para gestionar el intercambio de información entre todos los módulos del sistema, por lo que existía una única cola en la que todas las peticiones debían esperar según el orden de llegada. Esta característica posee un importante inconveniente: conforme el tráfico intercambiado incrementa el tiempo de respuesta del servidor aumenta, por lo que existe una creciente penalización en la atención a cualquiera de los módulos del sistema. Los módulos más perjudicados suelen ser aquellos que poseen conexiones pequeñas, pues aunque el intercambio de información que precisan es pequeño deben esperar a que todas las peticiones anteriores que se encuentran en la cola de espera sean atendidas, aun cuando estas peticiones pueden corresponder a conexiones grandes con un considerable tiempo de respuesta. Para mejorar la concurrencia del sistema, el esquema propuesto sigue esta filosofía de memoria compartida distribuida, pero añade un servidor específico para cada una de las conexiones existentes. El servidor central será el encargado de crear y destruir estos servidores específicos según sean necesarios. De esta forma, existirán tantas colas como conexiones, por lo que todas podrán ser atendidas concurrentemente disminuyendo el tiempo de respuesta de cada una de ellas. Este nuevo esquema se ha denominado *DLA ("Distributed and Layered Architecture")*, y se puede observar en la Figura 2.

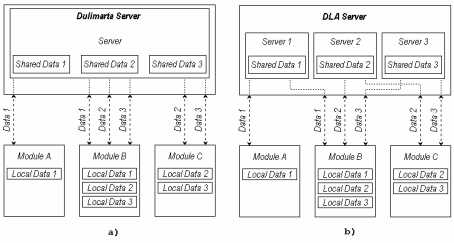

Figura 2. Esquema de la arquitectura de control: a) Dulimarta; b) DLA

Con el fin de verificar las mejores prestaciones obtenidas con el esquema propuesto *DLA* frente al originalmente propuesto por Dulimarta, se han realizado algunas pruebas específicas. La Tabla 1 muestra el retardo medio en milisegundos que sufre una conexión pequeña de 16 bytes en ser atendida por el servidor central bajo distintas cargas de tráfico. Se han probado distintas distribuciones de tráfico (uniforme, exponencial y gaussiana), con resultados muy similares en todas ellas. Todas las distribuciones tienen un tiempo medio de 100 ms, y la gausiana tiene una desviación típica de 12.5 ms. Los errores se han calculado con un intervalo de confianza del 95%.

El Test 1 se corresponde con la situación donde únicamente existe la conexión de prueba de 16 bytes en el sistema, por lo que obviamente ambos esquemas poseen tiempos de respuesta parecidos. En el Test 2 coexisten en el sistema la conexión de prueba de 16 bytes y 4 conexiones más también de 16 bytes, por lo que comienzan a aparecer pequeñas diferencias entre ambos esquemas. En los Test sucesivos, se ha ido incrementando considerablemente el tráfico, coexistiendo en el sistema la conexión de prueba de 16 bytes y 1 conexión de 64 Kbytes para el Test 3, 2 conexiones de 64 Kbytes para el Test 4, 3 conexiones de 64 Kbytes para el Test 5 y 4 conexiones de 64 Kbytes para el Test 6.

Bajo estas últimas pruebas, donde el tráfico del sistema es considerablemente alto, se puede apreciar claramente la mejora del esquema *DLA* frente al esquema de Dulimarta.

Uniforme	DLA (ms)	Dulimarta (ms)
Test 1	16.52 ± 0.17	16.27 ± 0.19
Test 2	16.3 ± 0.4	20.5 ± 0.8
Test 3	24 ± 3	28 ± 4
Test 4	32 ± 5	52 ± 6
Test 5	40 ± 6	78 ± 9
Test 6	48 ± 6	122 ± 13
Exponencial	**DLA (ms)**	**Dulimarta (ms)**
Test 1	16.05 ± 0.17	16.68 ± 0.25
Test 2	17.6 ± 1.0	19.8 ± 0.7
Test 3	21.8 ± 2.3	25 ± 3
Test 4	32 ± 4	46 ± 5
Test 5	37 ± 6	72 ± 8
Test 6	51 ± 7	126 ± 13
Guasiana	**DLA (ms)**	**Dulimarta (ms)**
Test 1	16.19 ± 0.17	16.29 ± 0.19
Test 2	16.4 ± 0.4	20.4 ± 0.7
Test 3	26 ± 3	24.0 ± 2.2
Test 4	33 ± 4	48 ± 5
Test 5	43 ± 6	78 ± 9
Test 6	50 ± 6	117 ± 13

Tabla 1. Comparativa de prestaciones entre el esquema *DLA* y el de Dulimarta

La Figura 3 muestra estos resultados gráficamente, permitiendo observar más claramente la mejora del esquema propuesto.

En algunos sistemas suele ser necesario, además de proporcionar mecanismos de intercambio de información entre módulos, poseer algún mecanismo de sincronización entre los mismos. El esquema propuesto por Dulimarta poseía semáforos compartidos que los distintos módulos utilizaban para sincronizarse. El esquema propuesto *DLA* utiliza una nueva estructura de datos más abstracta conocida como *mailbox*, que se podría traducir como buzón. Estos buzones son más sencillos de utilizar que los semáforos, por lo que facilitan enormemente la tarea de sincronización entre módulos. Constituyen un mecanismo directo de intercambio de mensajes entre módulos, ya que cualquier módulo puede leer y escribir en cualquier buzón existente en el sistema. Su diferencia con las conexiones radica en que los buzones permiten que el módulo que accede al mismo se bloquee si lo desea hasta que algún otro módulo escriba un nuevo dato en dicho buzón. De esta forma, los distintos módulos pueden desactivarse y quedar dormidos a la espera de que algún otro módulo los active escribiendo un mensaje en el buzón correspondiente. Este comportamiento además permite la optimización de los recursos del sistema, al quedar inactivos los módulos que no son necesarios hasta que se activen bajo demanda.

Por útlimo, se ha dotado al esquema *DLA* de una potente funcionalidad que no está presente en el esquema de Dulimarta. En la distribución de módulos a lo largo del sistema es común que algunos de estos módulos residan en la misma máquina. En este caso es claramente ineficiente conectarlos entre sí a través del servidor central mediante *sockets*, puesto que sería mucho más eficiente conectarlos entre sí localmente mediante memoria compartida. Por ello, se han implementado las mismas funcionalidades de la arquitectura (conexiones y buzones) de forma local, por lo que aquellos módulos que residan en la misma máquina pueden utilizar estos mecanismos de comunicación mucho más rápidos y eficientes.

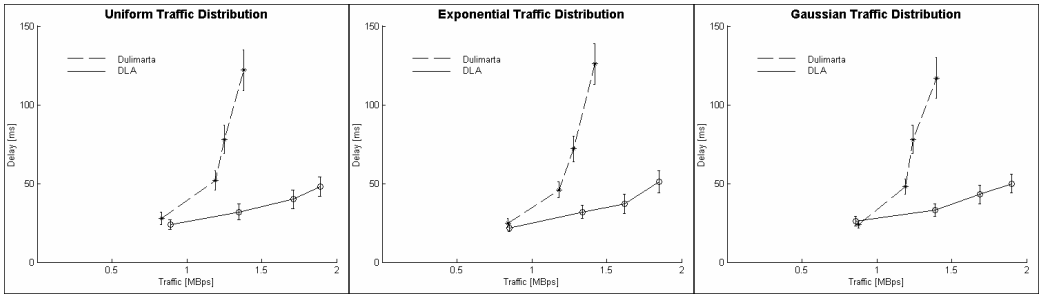

Figura 3. Comparativa de prestaciones entre el esquema *DLA* y el de Dulimarta

El uso de recursos locales libera además de carga al servidor central, por lo que redunda en una mejora de prestaciones en el sistema global. Un caso particular de esta situación ocurre cuando se quieren ubicar todos los módulos del sistema sobre una plataforma móvil con un PC integrado, pues en este caso todos los módulos deben coexistir en la misma máquina y esta funcionalidad de conexión local aporta un grado de optimización muy destacable. Realizando pruebas sencillas de conexión entre módulos locales los tiempos típicos de respuesta para tranferir datos entre módulos es de unos 20 µs para conexiones pequeñas de 16 bytes y de unos 750 µs para conexiones grandes de 64 Kbytes, tiempos claramente reducidos en comparación con los expuestos en la Tabla 1. Este tiempo, además, es independiente del número de módulos y conexiones existentes en el sistema, ya que el tiempo de acceso a memoria compartida sólo depende de la cantidad de datos que sean transferidos.

3.3. Componentes

Estructuralmente, el esquema DLA propuesto está compuesto por tres componentes:

a) *DLAServer*: servidor central encargado de gestionar la creación y destrucción de las distintas conexiones y buzones así como el acceso a los mismos.
b) *DLAAdmin*: elemento que permite la monitorización y configuración remota del servidor central.
c) *DLALibrary*: librería de uso para comunicarse con el servidor central e intercambiar datos entre los distintos módulos.

El servidor central *DLAServer* constituye el núcleo de la arquitectura. El administrador remoto *DLAAdmin* es una herramienta que permite por una parte configurar el funcionamiento del servidor, y por otra monitorizar y modificar los datos almacenados en el servidor central, es decir, el flujo de datos que se intercambian los distintos módulos entre sí. Esta facilidad proporciona un mecanismo muy versátil y potente para depurar el sistema bajo desarrollo. La librería *DLALibrary* permite la utilización de la arquitectura. Ha sido desarrollada bajo *C* para *Linux*, si bien con el tiempo se ha ido transcribiendo a *C* para *Windows*, *JAVA* y *Matlab*. Esto implica que cualquier módulo programado en alguno de estos lenguajes y plataformas puede usar la arquitectura. Puesto que la arquitectura se basa principalmente en el uso de *sockets*, cualquier plataforma y lenguaje de programación que los soporte es susceptible de usarla si se transcribe la librería de uso. Es importante destacar que la librería *DPALibrary* es la misma independientemente de que se utilicen el esquema remoto mediante el acceso al servidor central o el esquema local mediante memoria compartida, siendo este proceso totalmente transparente al usuario al ser las mismas funciones las que se utilizan en ambos casos. Este hecho simplifica más aun la utilización de la arquitectura.

4. Pruebas

La arquitectura propuesta ha sido probada en un entorno real del laboratorio del departamento sobre un robot *Pioneer P2-AT*, robot equipado con 8 sensores sonar y un AMD K6 400 MHz con 128 Mbytes de RAM. Se utilizaron además 3 ordenadores Pentium III 550 MHz con 128 Mbytes de RAM, uno para albergar al servidor central de la arquitectura *DLAServer* y los otros dos para distribuir los distintos módulos necesarios para implementar la aplicación.

4.1. Estilo

Para comparar las prestaciones obtenidas con el esquema *DLA* frente al esquema propuesto por Dulimarta en una aplicación real, se han realizado pruebas para medir la latencia de la capa reactiva y la latencia del sistema total (capa reactiva y capa deliberativa) en ambos escenarios. Se define la latencia como el intervalo de tiempo transcurrido desde que se recogen los datos del entorno a través del robot hasta que se calculan y se envían al mismo los comandos de movimiento adecuados a dichos estímulos. En la Figura 4 se comparan los resultados obtenidos en la latencia del sistema siguiendo el esquema *DLA* y siguiendo el esquema propuesto por Dulimarta. Los valores de latencia medios para el sistema reactivo son de 390±60 ms para Dulimarta y 178±11 ms para *DLA*, y para el sistema total de 2.8±0.7 s para Dulimarta y 1.49±0.16 s para *DLA*. El esquema *DLA* es claramente superior en todos los casos.

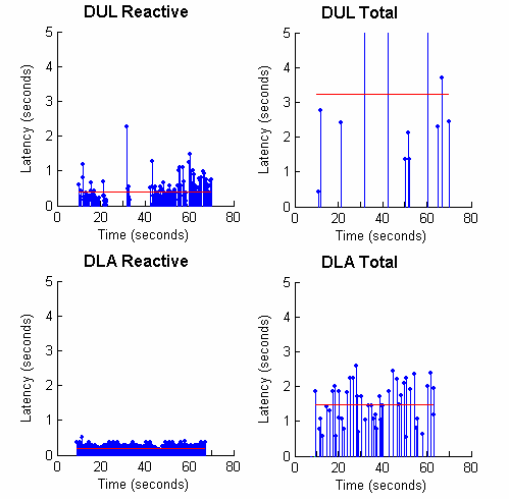

Figura 4. Latencia *DLA*-Dulimarta

4.2. Estructura

Las pruebas consistieron en proporcionar diferentes objetivos al robot en un entorno no explorado para forzar la navegación en el mismo. Durante la navegación, el robot tuvo que pasar cerca de numerosos obstáculos y a través de puertas, evitando la colisión mediante el funcionamiento de las capas reactivas. La representación del entorno mediante los mapas métrico y topológico se actualizaron continuamente, para mantener una representación lo más fidedigna posible del entorno del robot. La capa deliberativa de planificación de la ruta a seguir fue lanzada cada vez que era detectado un cambio significativo en el mapa topológico y se modificaban un número predeterminado de nodos en el mismo. Así pues, el robot tiende a seguir la ruta propuesta por la capa deliberativa. Si se detecta que el entorno se ha modificado sustancialmente, la capa deliberativa genera un nuevo plan mientras el robot continúa navegando sin colisionar con los obstáculos circundantes gracias a la capa reactiva.

La Figura 5 muestra el entorno donde se han llevado a cabo las pruebas reales, así como el camino recorrido entre los puntos de origen (D) y destino (A). Para facilitar la explicación de la prueba se ha dividido el experimento en 4 etapas, que son las que se describen a continuación.

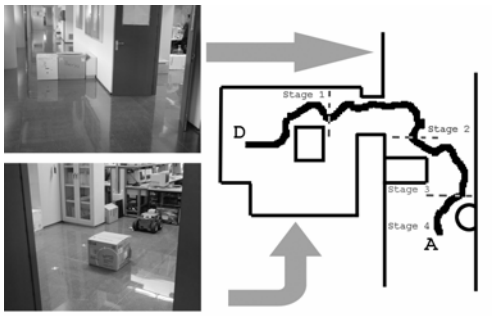

Figura 5. Entorno de pruebas

Etapa 1

Al inicio el robot está detenido esperando que se le introduzca algún destino al que dirigirse. En cuanto se proporciona un destino son activados los módulos reactivo, de localización y de planificación. La capa deliberativa opera con el mapa topológico disponible en ese momento, que será actualizado en cuanto el robot se mueva y se actualice el mapa métrico. Hasta que se obtiene una ruta óptima hasta el objetivo a partir del mapa topológico, no obstante, el robot navega hacia el objetivo en línea recta evitando obstáculos gracias a la capa reactiva. La Figura 6 muestra los mapas métrico y topológico del entorno en la etapa 1, así como la ruta calculada por la capa deliberativa. Esta ruta ha sido descompuesta en una serie de nodos intermedios que el robot deberá atravesar hasta alcanzar su destino final, conocidos como puntos de control. Estos puntos de control podrán variar conforme se calculen nuevas rutas sobre mapas topológicos más actualizados, y el robot simplemente se mueve entre sucesivos puntos de control evitando los obstáculos mediante la navegación que proporciona la capa reactiva.

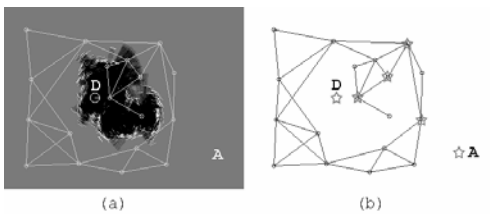

Figura 6. Etapa 1: a) mapas métrico y topológico; b) ruta calculada

Etapa 2

Al final de la etapa 1 el robot tuvo que esquivar una caja situada en el laboratorio. Tras ser esquivada mediante la capa reactiva, el mapa métrico fue actualizado con la parte inexplorada del entorno que se encontraba más allá de la caja. Puesto que los cambios en el mapa métrico fueron significativos, el mapa topológico fue recalculado, y tras detectar cambios importantes en el mismo la capa deliberativa fue relanzada para calcular otra nueva ruta hasta el destino final. La Figura 7 muestra los nuevos mapas métrico y topológico del entorno en la etapa 2, así como la nueva ruta calculada por la capa deliberativa.

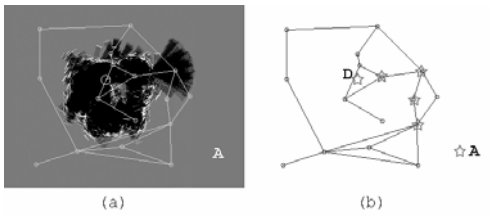

Figura 7. Etapa 2: a) mapas métrico y topológico; b) ruta calculada

Etapa 3

Una vez atravesada la puerta del laboratorio el robot entra en un pasillo totalmente inexplorado. Al comenzar a explorar esta nueva zona, los cambios en el mapa métrico son sustanciales, por lo que el mapa topológico es recalculado de nuevo. El nuevo mapa topológico también ha sido considerablemente actualizado, por lo que la capa deliberativa vuelve a ser lanzada y otra nueva ruta más óptima para alcanzar el destino final es calculada. La Figura 8 muestra los mapas métrico y topológico que posee el robot mientras navega por el entorno en la etapa 3, así como la nueva ruta calculada por la capa deliberativa. Es destacable observar los errores de la odometría que se empiezan a acumular en el desplazamiento del robot. Si la precisión de la aplicación así lo requiere estos errores deben ser corregidos mediante una técnica más compleja de localización, como puede ser el empleo de filtros de Kalman [12]. En las pruebas realizadas se intenta verificar el correcto funcionamiento de la arquitectura y del sistema de navegación compuesto por la fusión de los paradigmas métrico y topológico, por lo que estos errores no han sido corregidos al no interferir sustancialmente con los resultados que se pretenden poner de manifiesto.

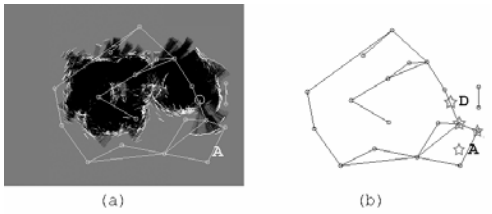

Figura 8. Etapa 3: a) mapas métrico y topológico; b) ruta calculada

Etapa 4

La Figura 9 muestra los mapas métrico y topológico actualizados al navegar por la etapa 4, así como la ruta planificada por la capa deliberativa. El camino final seguido por el robot se muestra en la Figura 5, donde se puede apreciar que dicha trayectoria es bastante razonable. Las oscilaciones que presenta se debe a que en determinadas áreas el proceso de exploración es bastante profundo, por lo que las rutas calculadas deben ser debidamente modificadas para adaptarse a las nuevas circunstancias del entorno.

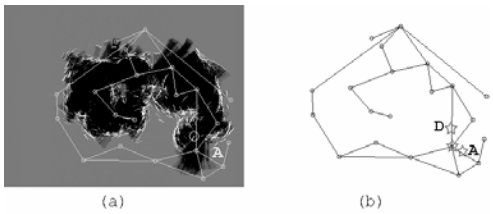

Figura 9. Etapa 4: a) mapas métrico y topológico; b) ruta calculada

5. Conclusiones y líneas futuras

Este trabajo ha presentado un sistema completo de navegación mediante una arquitectura híbrida que integra paradigmas reactivos y deliberativos. El estilo de la arquitectura se basa en un modelo de memoria compartida distribuido, el cual ha sido

desarrollado para ofrecer diversas facilidades y prestaciones de forma óptima,. La estructura integra distintas capas cuyas respuestas combinadas permiten implementar un sistema de navegación rápido y seguro con planificación. La capa reactiva usa el paradigma *Case Based Reasoning* como sistema de navegación, por ser un sistema que soporta entrenamiento del comportamiento de navegación deseado y proporcionar una respuesta rápida antes obstáculos inesperados del entorno. La capa deliberativa usa un mapa topológico con información de las regiones del entorno y su conectividad, para reducir el tiempo de cómputo de la ruta a seguir. El sistema se ha probado satisfactoriamente en entornos no explorados.

Entre las líneas futuras para mejorar el presente trabajo se pretende utilizar nuevas tecnologías de software distribuido como *CORBA* para aumentar la potencia y portabilidad del sistema, así como incorporar un sistema de localización eficiente que corrija los crecientes errores de odometría.

Agradecimientos

Este trabajo fue parcialmente financiado por la Comisión Interministerial de Ciencia y Tecnología (CICyT), proyecto Nº TIN2004-07741.

Referencias

[1] E. Coste-Manière, R. Simmons: "Architecture: the Backbone of Robotic Systems". Proc. *IEEE International Conference on Robotics and Automation* (ICRA2000), pp. 67-72. San Francisco, 2000.

[2] C. Urdiales, E.J. Pérez, F. Sandoval, J. Vázquez-Salceda: "A hybrid architecture for autonomous navigation in dynamic environments". Proc. *IEEE/WIC International Conference on Intelligent Agent Technology* (IAT03). Halifax, Canada, 2003.

[3] C. Urdiales, A. Bandera, E.J. Pérez, A. Poncela, F. Sandoval: "Hierarchical Planning in a Mobile Robot for Map Learning and Navigation". Autonomous Robotic Systems: Soft Computing and Hard Computing Methodologies and Applications, vol. 116, pp. 165-188. Physica-Verlag. Eds. C. Zhou, D. Maravall, D. Ruan. 2003.

[4] A. Poncela, E.J. Pérez, C. Urdiales, F. Sandoval: "A Hybrid Path Planning Technique for Partially Unknown Indoor Environments". Intelligent Automation and Soft Computing 2005, vol. 11, no. 3, pp. 155-166.

[5] A. Orëback, H. Christensen: "Evaluation of Architectures for Mobile Robotics". Autonomous Robots 2003, vol. 14, pp. 33-49.

[6] H. Chocon: "Object-Oriented Design and Distributed Implementation of a Mobile Robot Control System". Proc. *Workshop on Architecture for Intelligent Control System*. Nice, France, 1992.

[7] J.D. Wise, L. Ciscon: "TelRIP: Distributed Application Environment Operation Manual. Version 1.6". Technical Report 9103. Universities Space Automation/Robotics Consortium, 1992.

[8] R. Simmons: "Structured Control for Autonomous Robots". IEEE Transactions on Robotics and Automation 1994, vol. 10, no. 1, pp. 34-43.

[9] M. Lindström, A. Orëback, H. Christensen: "BERRA: A Research Architecture for Service Robots". Proc. *IEEE International Conference on Robotics and Automation* (ICRA2000). San Francisco, 2000.

[10] H.S. Dulimarta, A.K. Jain: "A Client/Server Control Architecture for Robot Navigation". Pattern Recognition 1996, vol. 29, no. 8, pp. 1259-1284.

[11] M. Shaw, P. Clements: "A Field Guide to Boxology: Preliminary Classification of Architectural Styles for Software Systems". Proc. *Computer Software and Applications Conference* (COMPSAC97), pp. 6-13. 1997.

[12] J.M. Pérez, R. Vázquez, P. Núñez, E.J. Pérez, F. Sandoval: "A Hough-based Method for Concurrent Mapping and Localization in Indoor Environments". Proc. *IEEE Conference on Robotics, Automation and Mechatronics* (RAM04). Singapore, 2004.